21st International Workshop on Algorithms in Bioinformatics

WABI 2021, August 2–4, 2021, Virtual Conference

Edited by

Alessandra Carbone

Mohammed El-Kebir

Editors

Alessandra Carbone
UMR 7238 CNRS, Sorbonne Université, Paris, France
alessandra.carbone@sorbonne-universite.fr

Mohammed El-Kebir
University of Illinois at Urbana-Champaign, Champaign, IL, USA
melkebir@illinois.edu

ACM Classification 2012
Mathematics of computing → Discrete mathematics; Applied computing → Computational biology; Mathematics of computing → Information theory; Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-200-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-200-6.

Publication date
July, 2021

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.WABI.2021.0

ISBN 978-3-95977-200-6 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Regular Papers

21st International Workshop on Algorithms in Bioinformatics (WABI 2021).
Editors: Alessandra Carbone and Mohammed El-Kebir
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This proceedings volume contains papers presented at the 21st Workshop on Algorithms in Bioinformatics (WABI 2021), which was virtually held in Chicago, Illinois, USA, August 2–4, 2021.

The Workshop on Algorithms in Bioinformatics is an annual conference established in 2001 to cover all aspects of algorithmic work in bioinformatics, computational biology, and systems biology. The conference is intended as a forum for presentation of new insights about discrete algorithms and machine-learning methods that address important problems in biology (particularly problems based on molecular data and phenomena), that are founded on sound models, that are computationally efficient, and that have been implemented and tested in simulations and on real datasets. The meeting's focus is on recent research results, including significant work-in-progress, as well as identifying and exploring directions of future research.

WABI 2021 was co-located with ACM-BCB 2021. Because of the COVID-19 pandemic, the ACM-BCB Organizing Committee decided to run the affiliated conferences online exclusively. This did not affect the scientific aspects of WABI, with the exception of somewhat lower number of submissions that fortunately maintained the high quality of WABI standards. The activities of the Program Committee of WABI took place as usual: peer review, discussion, selection of accepted papers, and publishing of the proceedings have been accomplished in the same way as in all WABI editions. This year, for the first time, WABI had a poster session, allowing students to interact with and present their ongoing research to the WABI community.

In 2021, a total of 38 manuscripts were submitted to WABI from which 20 were selected for presentation at the conference and are included in this proceedings volume as full papers. Extended versions of selected papers have been invited for publication in a thematic series in the journal Algorithms for Molecular Biology (AMB), published by BioMed Central. The 20 papers selected for the conference underwent a thorough peer review, involving at least three (and often four or five) independent reviewers per submitted paper, followed by discussions among the WABI Program Committee members. The selected papers cover a wide range of topics including phylogenetic trees and networks, biological network analysis, sequence alignment and assembly, genomic-level evolution, sequence and genome analysis, RNA and protein structure, topological data analysis, and more. They are ordered randomly within this volume.

We thank all the authors of submitted papers for making this conference possible. A special thanks goes to all the members of the WABI 2021 Program Committee and their subreviewers for their participation in a very active review process with numerous exchanges that culminated in constructive reviews reports for the authors. We are also grateful to the WABI Steering Committee (Bernard Moret, Vincent Moulton, Jens Stoye and Tandy Warnow) for their availability, help and advice. We thank all the conference participants, session chairs, and speakers who contributed to a great scientific program. In particular, we are indebted to the keynote speaker of the conference, Mona Singh (Princeton University), for her presentation. We thank the ACM-BCB 2021 Organizing Committee for setting up the event in these complicated times due to the pandemic emergency. Finally, we thank Lechuan Li for maintaining the WABI 2021 website.

21st International Workshop on Algorithms in Bioinformatics (WABI 2021).
Editors: Alessandra Carbone and Mohammed El-Kebir
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Previous proceedings of WABI appeared in LNCS/LNBI volumes 2149 (WABI 2001, Aarhus), 2452 (WABI 2002, Rome), 2812 (WABI 2003, Budapest), 3240 (WABI 2004, Bergen), 3692 (WABI 2005, Mallorca), 4175 (WABI 2006, Zurich), 4645 (WABI 2007, Philadelphia), 5251 (WABI 2008, Karlsruhe), 5724 (WABI 2009, Philadelphia), 6293 (WABI 2010, Liverpool), 6833 (WABI 2011, Saarbrücken), 7534 (WABI 2012, Ljubljana), 8126 (WABI 2013, Sophia Antipolis), 8701 (WABI 2014, Wroclaw), 9289 (WABI 2015, Atlanta), and 9838 (WABI 2016, Aarhus). As of 2016, they appeared in LIPICS volumes 88 (WABI 2017, Boston), 113 (WABI 2018, Helsinki), 143 (WABI 2019, Boston) and 172 (WABI 2020, Pisa).

Alessandra Carbone & Mohammed El-Kebir

Program Committee

Alessandra Carbone (Co-Chair)
Sorbonne Universite, France

Mohammed El-Kebir (Co-Chair)
University of Illinois at Urbana-Champaign, USA

Tatsuya Akutsu
Kyoto University, Japan

Marco Antoniotti
Universita di Milano-Bicocca, Italy

Anne Bergeron
Universite du Quebec a Montreal, Canada

Paola Bonizzoni
Universita di Milano-Bicocca, Italy

Christina Boucher
University of Florida, USA

Rayan Chikhi
Institut Pasteur, France

Leonid Chindelevitch
Imperial College London, UK

Lenore Cowen
Tufts University, USA

Daniel Doerr
University of Duesseldorf, Germany

Nadia El-Mabrouk
University of Montreal, Canada

Anna Gambin
Warsaw University, Poland

Bjarni Halldorsson
deCODE genetics and Reykjavik University, Iceland

Katharina Huber
University of East Anglia, UK

Carl Kingsford
Carnegie Mellon University, USA

Gunnar Klau
Heinrich Heine University Dusseldorf, Germany

Gregory Kucherov
University of Paris Est, France

Manuel Lafond
Universite de Sherbrooke, Canada

Elodie Laine
Sorbonne Universite, France

Yu Lin
Austrialia National University, Australia

Stefano Lonardi
University of California Riverside, USA

Veli Makinen
University of Helsinki, Finland

Guillaume Marcais
Carnegie Mellon University, USA

Tobias Marschall
Heinrich Heine University, Germany

Erin Molloy
University of Maryland College Park / UCLA, US

Bernard Moret
Ecole Polytechnique Federale de Lausanne, Switzerland

Vincent Moulton
University of East Anglia, UK

Francesca Nadalin
EMBL-EBI / ITT, Italy, England

Mihai Pop
University of Maryland, USA

Teresa Przytycka
NCBI / NIH, USA

William Stafford Noble
University of Washington, USA

Aida Ouangraoua
Universite de Sherbrooke, Canada

Ion Petre
Turku University, Finland

21st International Workshop on Algorithms in Bioinformatics (WABI 2021).
Editors: Alessandra Carbone and Mohammed El-Kebir
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Most Parsimonious Reconciliation Problem in the Presence of Incomplete Lineage Sorting and Hybridization Is NP-Hard

Matthew LeMay
Department of Mathematics, Harvey Mudd College, Claremont, CA, USA

Yi-Chieh Wu[1]
Department of Computer Science, Harvey Mudd College, Claremont, CA, USA

Ran Libeskind-Hadas
Department of Computer Science, Harvey Mudd College, Claremont, CA, USA

Abstract

The maximum parsimony phylogenetic reconciliation problem seeks to explain incongruity between a gene phylogeny and a species phylogeny with respect to a set of evolutionary events. While the reconciliation problem is well-studied for species and gene trees subject to events such as duplication, transfer, loss, and deep coalescence, recent work has examined species phylogenies that incorporate hybridization and are thus represented by networks rather than trees. In this paper, we show that the problem of computing a maximum parsimony reconciliation for a gene tree and species network is NP-hard even when only considering deep coalescence. This result suggests that future work on maximum parsimony reconciliation for species networks should explore approximation algorithms and heuristics.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases phylogenetics, reconciliation, deep coalescence, hybridization, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.1

Related Version *Previous Version*: `https://www.biorxiv.org/content/10.1101/2021.03.14.435321v1`

Funding *Matthew LeMay*: Supported by the National Science Foundation under Grant No. IIS-1751399.
Yi-Chieh Wu: Supported by the National Science Foundation under Grant No. IIS-1751399.
Ran Libeskind-Hadas: Supported by the National Science Foundation under Grant No. IIS-1905885.

Acknowledgements The authors thank Adam Walker and the anonymous reviewers for valuable comments that helped improve the paper.

1 Introduction

Genes evolve via several evolutionary processes operating at various evolutionary timescales. Nucleotides can mutate, and domains can recombine. Genes can be generated, lost, or replaced through gene duplication, gene loss, horizontal gene transfer, and gene conversion. Populations can diverge or combine through speciation and hybridization. In addition to these events, within a population, polymorphisms can persist across speciation events, leading to a phenomenon known as *incomplete lineage sorting (ILS)* [15, 18]. Thus, the history of a set of genes may differ from the history of the species in which they evolved [12].

[1] Corresponding author

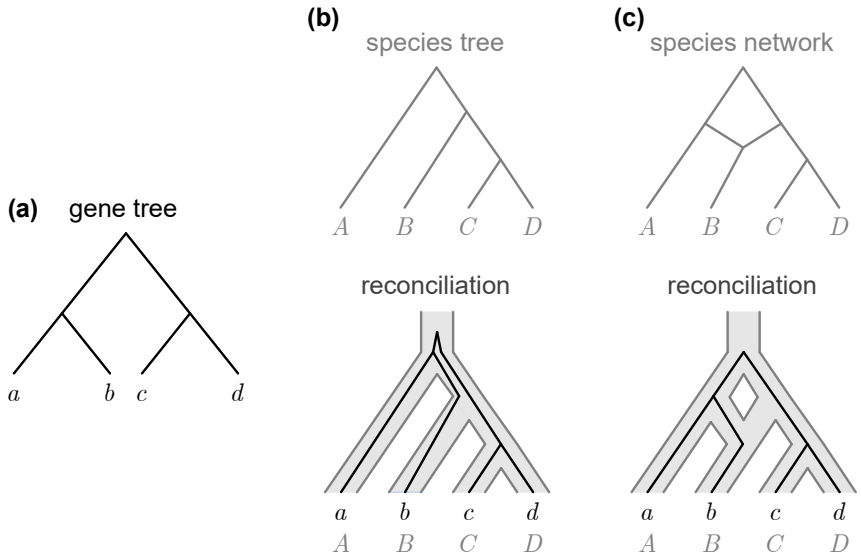


Figure 1 Gene trees, species trees, and species networks. (a) A gene tree. (b) A species tree and reconciliation. Under the multispecies coalescent model, the gene tree evolves within the species tree, and incongruence between the trees is due to ILS. (c) A species network and reconciliation. The same gene tree evolves within the species network, and no ILS is necessary.

In phylogenetics, *reconciliations* attempt to explain these differences by mapping gene histories within species histories to infer the evolutionary events that shaped that gene family (Figure 1a,b). The simplest and most common approach seeks a most parsimonious reconciliation (MPR) [7, 12, 14], in which each type of event in the model has an associated non-negative cost and the objective is to find a reconciliation of minimum total cost.

The time complexity of the MPR problem depends on the events being modeled and the set of constraints being considered. For example, the lowest common ancestor mapping, which can be computed in polynomial time [14, 26], solves the MPR problem when considering only duplications [8], only duplications and losses [8], and only deep coalescence [22]. When considering duplications, transfers, and losses, the MPR problem can be solved in polynomial time or is NP-hard depending on whether the species tree is undated, partially dated, or fully dated, and on whether the reconciliation is constrained to be time-consistent [13, 20]. Similarly, depending on details of the underlying model, the MPR problem can be solved in polynomial time when considering duplications, transfers, losses, and coalescence [4, 17], or is NP-hard when considering duplications, losses, and coalescence [2]. The MPR problem is also NP-hard when simultaneously modeling the evolution of domains, genes, and species [9].

Though the species history is often represented by a tree, hybridization is increasingly recognized as an important evolutionary process, requiring the use of species networks (Figure 1c). In eukaryotic species, hybridization encompasses two different processes: hybrid speciation, in which there is no underlying tree, and introgression, in which there is an underlying tree [5, 6]. Horizontal gene transfer in prokaryotic species can be considered a special case of introgression [16] and results in reticulate evolutionary histories as well.

Several authors have recently considered reconciliations with species networks. For example, several methods exist for the related problem of inferring a species network that minimizes deep coalescence [23, 24, 25]. However, the authors did not analyze the time complexity of their algorithms. Furthermore, their approaches require searching over the space of species network topologies, in contrast to the problem considered here, in which the species network is assumed to be known. Other authors have focused on the problem

of inferring MPRs between gene trees and species networks, showing that the problem can be solved in polynomial time when minimizing the duplication-transfer-loss cost [11] or the duplication-loss cost [19]. There is little previous work on the problem of inferring MPRs between a gene tree and species network in which incongruence is due to deep coalescence, and the question of whether this problem can be solved in polynomial time remained open.

In this paper, we show that the problem of inferring an MPR between a gene tree and species network in the presence of incomplete lineage sorting is NP-hard. Our results suggest that future work on this problem should focus on developing heuristics or approximation algorithms.

2 Definitions

We use the terms *node* and *vertex* interchangeably. A *rooted binary phylogenetic network* refers to a rooted directed acyclic graph with a single root with in-degree 0 and out-degree 2; additional internal nodes with either in-degree 1 and out-degree 2, called *branch nodes*, or in-degree 2 and out-degree 1, called *hybridization nodes*; and one or more leaves with in-degree 1 and out-degree 0. Edges leading to hybridization nodes are called *hybridization edges*. Given a network N, let $V(N)$ denote its node set and $E(N)$ denote its edge set. Let $L(N) \subset V(N)$ denote its leaf set, $I(N) = V(N) \setminus L(N)$ denote its set of internal nodes, and $r(N) \in I(N)$ denote its root node. For a node $v \in V(N)$, let $c(v)$ denote its set of children (the empty set if v is a leaf), let $p(v)$ denote its set of parents (the empty set if v is the root node), and, if v has a single parent, $e(v)$ denotes the edge $(p(v), v)$. The size of N, denoted by $|N|$, is equal to $|V(N)| + |E(N)|$.

Let $\leq_N$ ($<_N$) be the partial order on $V(N)$ such that $v \leq_N u$ ($v <_N u$) if and only if there exists a path in N from u to v ($v \neq u$); v is said to be *lower or equal to* (lower than) u, and v a (strict) *descendant* of u, and u a (strict) *ancestor* of v.

Given two nodes u and v of N such that $v \leq_N u$, a path from u to v in N is a sequence of contiguous edges from u to v in N. Note that if $u = v$, the path from u to v is empty. As there can be multiple paths between pairs of vertices in a network, let $paths_N(u, v)$ denote the set of all paths from u to v. Let $paths(N)$ denote the set of all paths in network N.

A binary phylogenetic tree is a binary phylogenetic network with no hybridization nodes; that is, a directed binary tree. In the remainder of this paper, we refer to rooted binary phylogenetic networks and rooted phylogenetic trees simply as *networks* and *trees*, respectively.

A *species network* S represents the evolutionary history of a set of species and a *gene tree* G represents the evolutionary history of a set of genes sampled from these species. A *leaf mapping* $Le\colon L(G) \to L(S)$ associates each leaf in the gene tree with a corresponding species from which the gene was sampled. The mapping need not be one-to-one nor onto. Note that gene phylogenies are assumed to be trees whereas species phylogenies may, in general, be networks.

In this paper, we assume that both the gene tree and species network are undated. Thus, the only temporal constraints on the nodes are those induced by ancestor-descendant relationships.

2.1 Reconciliations

A *reconciliation* for a given gene tree, species network, and leaf mapping comprises a pair of mappings: The *vertex mapping* $R_v : V(G) \to V(S)$ associates each node of G with a node of S. For each non-root node g of G, the *path mapping* $R_p : V(G) \to paths(S)$ associates a path in S from $R_v(p(g))$ to $R_v(g)$. The vertex mapping must be consistent with the given leaf

mapping and must satisfy temporal constraints; namely if a gene node g is mapped to species node s and a child g' of g is mapped to species node s', then s must be an ancestor of s'. The path mapping is required because S is a network, and thus there may be multiple paths between ancestors and descendants in the network. The formal definition of a reconciliation is given in Definition 1.

▶ **Definition 1** (Reconciliation). *Given a gene tree G, a species network S, and a leaf mapping Le, a* reconciliation[2] *R for (G, S, Le) is a pair of mappings (R_v, R_p) where $R_v : V(G) \to V(S)$ is a* vertex mapping *and $R_p : V(G) \to paths(S)$ is a* path mapping *subject to the following constraints:*

1. *If $g \in L(G)$, then $R_v(g) = Le(g)$.*
2. *If $g \in I(G)$, then for each $g' \in c(g)$, $R_v(g') \leq_S R_v(g)$.*
3. *If $g \neq r(G)$, then $R_p(g) \in paths_S(R_v(p(g)), R_v(g))$. Otherwise, $R_p(g) = \emptyset$.*

Constraint 1 asserts that R_v extends the leaf mapping Le. Constraint 2 asserts that R_v satisfies the temporal constraints implied by S. Constraint 3 asserts that the vertex mapping and path mapping are consistent. We note that some formulations of the reconciliation problem include an additional constraint asserting that no two paths in the path mapping use two different hybridization edges leading to the same hybridization node. While we do not explicitly enforce this constraint in Definition 1, the NP-hardness proof in the next section satisfies this additional constraint nonetheless.

In a multispecies coalescent process, evolution in the species network is viewed backward in time, from the leaves toward the root. Then, given a reconciliation R, we can count the number of gene lineages "passing through" each edge e of the species network. Specifically, given edge $e \in E(S)$,

$$\mathbf{L}_R(e) = |\{g \in V(G) : e \in R_p(g)\}|,$$

and the number of "extra lineages" is defined to be

$$\mathbf{XL}_R(e) = \max(0, \mathbf{L}_R(e) - 1).$$

Note, for example, that if two gene paths pass through a species edge, there is one extra lineage on that edge.

Finally, the *deep coalescence cost* of a reconciliation is the sum of extra lineages across all edges of the species network:

$$\mathbf{DC}_R = \sum_{e \in E(S)} \mathbf{XL}_R(e).$$

This value is the *reconciliation cost* in this model.

Finally, we formalize these optimization and decision problems:

▶ **Problem 2** (Most Parsimonious Reconciliation (MPR)). *Given a gene tree G, a species network S, and a leaf mapping Le, find a reconciliation R for (G, S, Le) such that the deep coalescence cost $\mathbf{DC}_R$ is minimized.*

▶ **Problem 3** (Most Parsimonious Reconciliation Decision Problem (MPRD)). *Given a gene tree G, a species network S, a leaf mapping Le, and an integer k, is there a reconciliation R for (G, S, Le) such that $\mathbf{DC}_R \leq k$?*

[2] When explaining topological incongruence through only deep coalescence, a reconciliation is sometimes called a *coalescent history* [5].

3 NP-hardness

▶ **Theorem 4.** *MPRD is NP-hard.*

In the proof that follows, it will be convenient to consider the gene tree as the collection of all paths P from its root to its leaves. For a given leaf mapping $Le : L(G) \to L(S)$, a *lineage mapping* with respect to Le is a mapping M from each path $p_\ell \in P$ whose endpoint is leaf ℓ to a path in S from some fixed node $v \in V(S)$ to $Le(\ell)$. Each reconciliation has a corresponding lineage mapping, M. Specifically, let $R = (R_v, R_p)$ be a reconciliation and let $r(G), g_1, \ldots, g_k$, $g_k = \ell$, denote the nodes on the unique path from $r(G)$ to leaf ℓ. Then, M associates this path in G with path $R_p(g_1)R_p(g_2)\ldots R_p(g_k)$ in S. Note that multiple different reconciliations may induce the same lineage mapping since there are, in general, different mappings of nodes of G to nodes of S than induce the same set of paths. For simplicity, when referring to lineage mappings we use the notation $M(\ell)$ in lieu of $M(p_\ell)$.

Finally, we use the notation (A, B) to represent a binary tree with a root and children A and B, either of which may be leaves or trees themselves.

Proof. Our proof is by a reduction from 3SAT. In particular, we consider the most general version of 3SAT in which the literals in a clause need not be unique and clauses need not be unique. Consider an instance of 3SAT with n variables and m clauses and, without loss of generality, assume that n and m are both powers of 2. (If not, the 3SAT instance can be padded with dummy variables and clauses to construct an equivalent instance in polynomial time by simply introducing new variables as needed and repeating clauses as needed.)

Construction. The gene tree G is constructed as follows: A *gene variable gadget* for a variable x_i comprises a tree G_i^x with three leaves, labelled a_i, b_i, y_i, with the topology $((a_i, b_i), y_i)$. The root of this gadget is labelled x_i. These n variable gadgets are connected via a perfect binary tree G^x. A *gene clause gadget* for a clause c_j consists simply of a single leaf ℓ_j. These m leaves are connected via a perfect binary tree G^c. A *k-caterpillar* of length k is a binary tree constructed from a path of length k ($k+1$ vertices and k edges) where each of the first k vertices on that path has two children: one is the next vertex on the path, and another is a leaf. In total a k-caterpillar has $k+1$ leaves. The root of the gene tree G has two children: one is the root of the tree G^x and the other is the root of a $2n$-caterpillar called e. One of the two deepest leaves of caterpillar e is the root of G^c and the remaining leaves are labeled $e_1, \ldots, e_{2n}$ in order of depth from the root. The structure of the gene tree is depicted in Figure 2.

The species network S is constructed as follows: A *species variable gadget* for variable x_i consists of a subtree S_i^x with four children labeled T_i, A_i, B_i, and F_i, with topology $((T_i, A_i), (F_i, B_i))$. A_i and B_i are leaves. The root of this gadget is labelled X_i. T_i and F_i are the first vertices of paths which correspond to setting x_i to true or false, respectively. We henceforth refer to these paths as the *variable setting paths* for T_i and F_i, respectively. The remaining vertices on these variable settings paths are described in the next paragraph; ultimately these paths join at a hybridization node which has a single leaf child Y_i. The roots of these n variable gadgets are joined via a perfect binary tree S^x.

A *species clause gadget* S_j^c for clause C_j is constructed as follows. Let z_1, z_2, and z_3 denote the three literals in that clause. If literal z_1 is the unnegated variable x_i then a vertex $U_{1,j}$ and its child $V_{1,j}$ are introduced on the variable setting path for F_i in the species variable gadget for x_i. Conversely, if literal z_1 is the negation of x_i, then vertex $U_{1,j}$ and its child $V_{1,j}$ are introduced on the variable setting path for T_i. The analogous process is

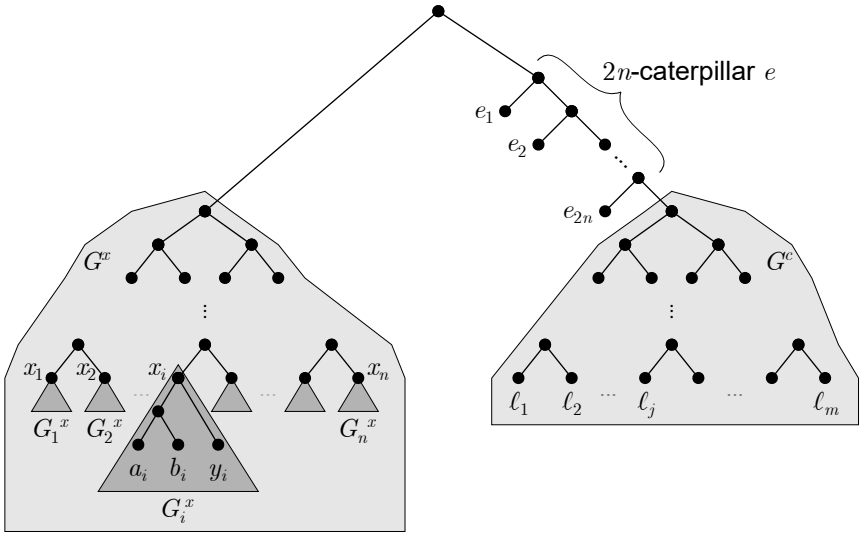


Figure 2 The gene tree in the NP-hardness construction. The shaded subtree G^x on the left contains a variable gadget G_i^x for each variable x_i in the 3SAT instance, and the shaded subtree G^c on the right contains a single leaf ℓ_j for each clause j in the 3SAT instance.

used to introduce a pair of vertices $U_{2,j}$ and $V_{2,j}$ for the variable setting path for literal z_2 and a pair of vertices $U_{3,j}$ and $V_{3,j}$ for the variable setting path for literal z_3. If the clause contains repeated literals, we will add the vertices to paths in numerical order based on their subscripts. For example, if $z_1 = z_2$, then we will introduce the vertex $U_{2,j}$ immediately after the vertex $V_{1,j}$ on the corresponding variable setting path, so that $V_{1,j}$ will be one of the parents of $U_{2,j}$. The root of the species clause gadget for clause C_j is a vertex U_j with $U_{1,j}$ as one child and U'_j as the second child whose children are $U_{2,j}$ and $U_{3,j}$. Note that $U_{1,j}$, $U_{2,j}$, and $U_{3,j}$ are hybridization nodes since they each have two parents. Node $V_{1,j}$ has a second child V_j, $V_{2,j}$ and $V_{3,j}$ have a child V'_j (a hybridization node), V'_j is another parent of V_j (a hybridization node) which, in turn, has a single child L_j, a leaf of the species network. The roots of the m species clause gadgets are connected with a perfect binary tree S^c.

Finally, the root of the species network has two children: one is the root of the species variable tree S^x and the other is the root of a $2n$-caterpillar called E. One of the two deepest leaves of caterpillar E is the root of the clause gadget tree S^c and the remaining leaves are labeled $E_1, \ldots, E_{2n}$ in increasing depth from the root. A representation of the species network is shown in Figure 3.

The leaf mapping Le is as follows: For the leaves in the variable gadgets, $Le(a_i) = A_i$, $Le(b_i) = B_i$, and $Le(y_i) = Y_i$, for $1 \leq i \leq n$. For the leaves of the $2n$-caterpillar, $Le(e_i) = E_i$, for $1 \leq i \leq 2n$. For the leaves in the clause gadgets, $Le(\ell_i) = L_i$, for $1 \leq i \leq m$.

Finally, the value of k in the decision problem is set to be n, the number of variables in the 3SAT instance. It is easily seen that this construction can be performed in time polynomial in the size of the 3SAT instance.

Correctness. We prove that the constructed MPRD instance has a reconciliation with deep coalescence cost no more than n if and only if there is a satisfying assignment of the variables in the given 3SAT instance.

We begin with several observations. First, in any reconciliation, $r(G)$ must be mapped to $r(S)$ because the lowest common ancestor of the leaves in the variable gadgets and clause gadgets in G is $r(G)$ while the lowest common ancestor in S of their images under the

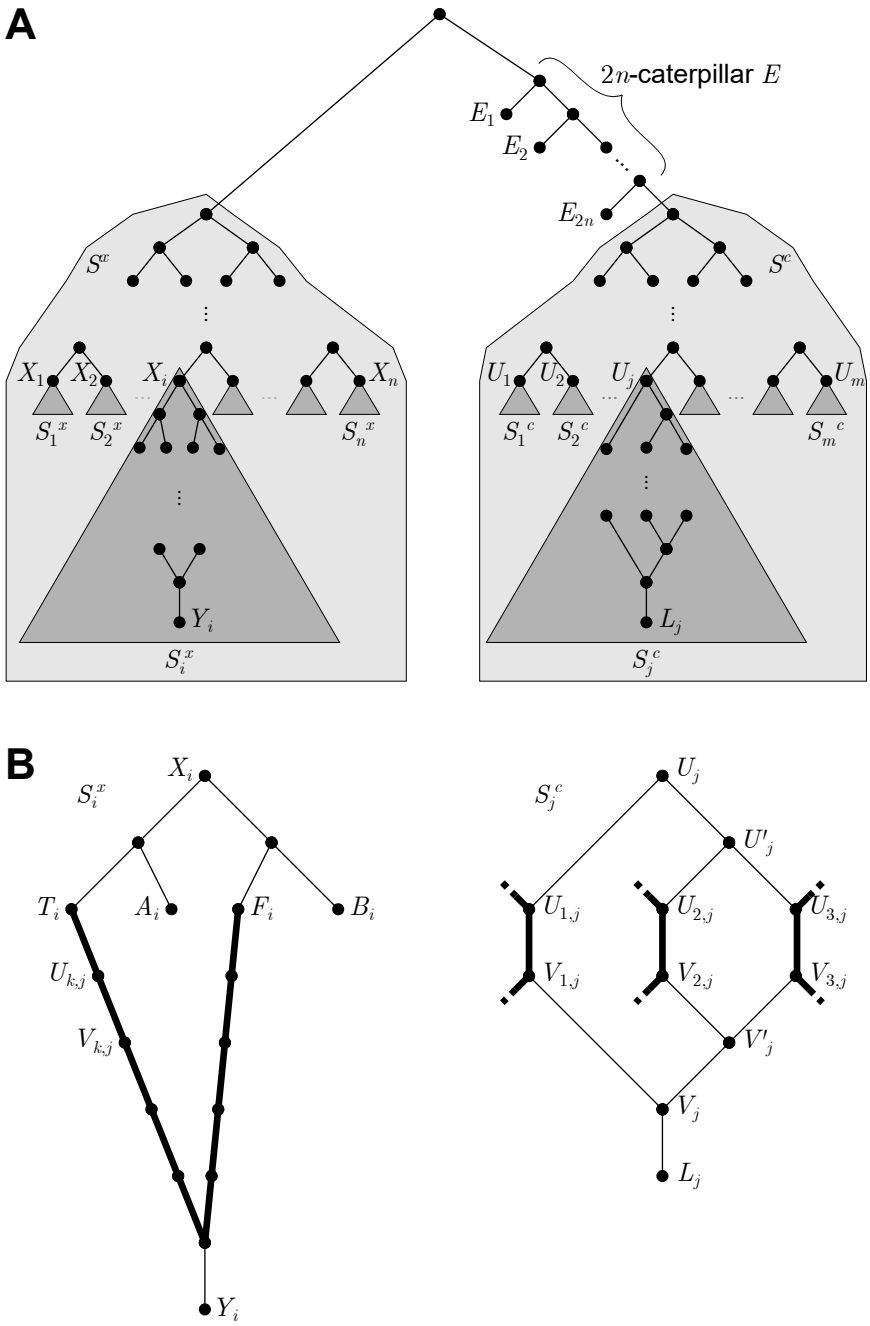


Figure 3 The species network in the NP-hardness reduction. (A) The shaded subtree S^x on the left contains a variable gadget S_i^x for each variable x_i in the 3SAT instance, and the subtree S^c on the right contains a clause gadget S_j^c for each clause j in the 3SAT instance. (B) The variable gadget on the left is shown in detail. Vertices T_i and F_i are the first vertices on the variable setting paths (indicated in bold) for variable x_i. The clause gadget on the right is shown in detail for clause j. The bold edges are from variable setting paths for the three variables in clause j. Note that the edge $U_{k,j}, V_{k,j}$ indicated on the left is the k^{th} bold edge in the clause gadget for clause j if and only if variable x_i is the k^{th} variable in clause j. In this example, $U_{k,j}, V_{k,j}$ appears on a true variable setting path, indicating that variable x_i appears negated in clause j.

leaf mapping is $r(S)$. The species network S has unique paths from its root to the leaves $A_1, ..., A_n$, $B_1, ..., B_n$, and $E_1, ..., E_{2n}$. Therefore, all lineage mappings and have the same unique paths $M(\ell)$ for all leaves ℓ among $a_1, \ldots, a_n$, $b_1, \ldots, b_n$, and $e_1, \ldots, e_{2n}$. The only leaves $\ell \in L(G)$ for which $M(\ell)$ has more than one possible path are $y_1, ..., y_n$ and $\ell_1, ..., \ell_m$.

Note that in order for a reconciliation to have cost no more than n, the induced lineage mapping M must satisfy the property that $M(y_i)$ contains the node X_i, the root of the gadget S_i^x. To see this, suppose by way of contradiction that there is a variable leaf y_i such that the path $M(y_i)$ (a path from $r(S)$ to Y_i) does not contain the vertex X_i. The only paths from $r(S)$ to Y_i that do not contain X_i are through clause gadgets, and therefore $M(y_i)$ must pass through the E caterpillar. Since there is a unique path from $r(S)$ to A_i in S and that path does not pass through the E caterpillar, $M(a_i)$ cannot contain nodes from the E caterpillar. Therefore, $M(y_i)$ must diverge from $M(a_i)$ at $r(S)$, and therefore also diverges from $M(e_{2n})$ at $r(S)$ since e_{2n} is more distantly related to y_i than a_i is to y_i. But then each of the $2n$ internal nodes of the caterpillar E has at least two lineages, one from $M(y_i)$ and one from $M(e_{2n})$, contributing a cost of at least $2n > n$, contradicting the assumed cost bound.

There are, therefore, only two possibilities for $M(y_i)$ – it either includes the variable setting path for T_i or the variable setting path for F_i in the variable gadget for x_i. Both of these options contribute at least one to the total cost since $M(a_i)$ and $M(b_i)$ must diverge at (or above) X_i and, since y_i is more distantly related to a_i and b_i than a_i and b_i are to one another, $M(y_i)$ must diverge from $M(a_i)$ and $M(b_i)$ at (or above) X_i. Thus, $M(y_i)$ must contribute an extra lineage on an edge shared with $M(a_i)$ or $M(b_i)$. Since there are n variables, this contributes a cost of n, so these are necessarily the only extra lineages.

For any clause j, $M(\ell_j)$ must contain at least one of the edges $(U_{k,j}, V_{k,j})$ for $k \in \{1, 2, 3\}$. This is a consequence of the fact that without these three edges, there is no path in S from the root to L_j. Therefore $M(\ell_j)$ shares an edge with at least one species variable setting path corresponding to the negation of a literal in clause j.

Now suppose there is a satisfying assignment of the variables in the 3SAT instance. Then construct a lineage mapping M with respect to Le as follows: $M(y_i)$ contains the variable setting path T_i if the variable x_i is set to true, and the variable setting path F_i if the variable x_i is set to false. For a clause j, let z_k, $k \in \{1, 2, 3\}$, denote one of three literals in that clause that evaluates to true with respect to the given satisfying assignment. If z_k is an unnegated variable x_i, then, by construction, the F_i variable setting path contains the edge $(U_{k,j}, V_{k,j})$ in the clause gadget S_j^c. We then construct $M(\ell_j)$ so that it follows the unique path from $r(S)$ to U_j, and then passes through the clause gadget via that edge. If z_k is a negated variable $\neg x_i$, then, by construction, the T_i variable setting path contains the edge $(U_{k,j}, V_{k,j})$ in the clause gadget S_j^c and $M(\ell_j)$ is chosen to pass through the clause gadget via that edge. A reconciliation inducing this lineage mapping is trivial since each vertex in the gene tree has a corresponding vertex in the species network. The only cost incurred by this reconciliation is one for each variable gadget as noted above. The total cost is therefore n and thus this is a "yes" instance of MPRD.

Conversely, suppose there is some reconciliation with cost at most n and let M be the corresponding lineage mapping. Then, we induce a setting of each variable x_i based on whether $M(y_i)$ contains the T_i or F_i variable setting path. As noted previously, this induces a cost of n and thus the remaining paths cannot contribute any additional cost. Therefore, for each clause C_j, $M(\ell_j)$ must pass through an otherwise unused edge $(U_{k,j}, V_{k,j})$, $k \in \{1, 2, 3\}$, implying that, by construction, the k^{th} literal in clause C_j has a setting that satisfies that clause. Therefore, the 3SAT instance is satisfied. ◀

Finally, we note that for simplicity, the reduction above did not seek to ensure that the species network has a temporal representation, meaning that there is a consistent timing of events in that network. It is always possible to add additional nodes to the species network to satisfy the temporal representation property [1] and it is easily verified that our reduction holds after adding these nodes.

4 Discussion

In this work, we have shown that the problem of inferring an MPR between a gene tree and species network in the presence of incomplete lineage sorting is NP-hard. These results suggest several important directions for future research. First, approximation algorithms and exact fixed-parameter tractable algorithms should be explored for the MPR problem. Second, the problem may be solved effectively in many instances using satisfiability solvers or integer linear programming, as has been done for phylogenetic reconciliation in other event models [3, 10, 21]. Third, heuristics can be explored and tested experimentally.

References

1 Mihaela Baroni, Charles Semple, and Mike Steel. Hybrids in real time. *Syst Biol*, 55(1):46–56, 2006. doi:10.1080/10635150500431197.

2 Daniel Bork, Ricson Cheng, Jincheng Wang, Jean Sung, and Ran Libeskind-Hadas. On the computational complexity of the maximum parsimony reconciliation problem in the duplication-loss-coalescence model. *Algorithm Mol Biol*, 12(6), 2017. doi:10.1186/s13015-017-0098-8.

3 Morgan Carothers, Joseph Gardi, Gianluca Gross, Tatsuki Kuze, Nuo Liu, Fiona Plunkett, Julia Qian, and Yi-Chieh Wu. An integer linear programming solution for the most parsimonious reconciliation problem under the duplication-loss-coalescence model. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3388440.3412474.

4 Yao-ban Chan, Vincent Ranwez, and Céline Scornavacca. Inferring incomplete lineage sorting, duplications, transfers and losses with reconciliations. *J Theor Biol*, 432:1–13, 2017. doi:10.1016/j.jtbi.2017.08.008.

5 R. A. Leo Elworth, Huw A. Ogilvie, Jiafan Zhu, and Luay Nakhleh. Advances in computational methods for phylogenetic networks in the presence of hybridization. In Tandy Warnow, editor, *Bioinformatics and Phylogenetics: Seminal Contributions of Bernard Moret*, pages 317–360. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-10837-3_13.

6 Ryan A. Folk, Pamela S. Soltis, Douglas E. Soltis, and Robert Guralnick. New prospects in the detection and comparative analysis of hybridization in the tree of life. *Am J Bot*, 105(3):364–375, 2018. doi:10.1002/ajb2.1018.

7 Morris Goodman, John Czelusniak, G. William Moore, A.E. Romero-Herrera, and Genji Matsuda. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Syst Zool*, 28(2):132–163, 1979. doi:10.1093/sysbio/28.2.132.

8 Paweł Górecki and Jerzy Tiuryn. Dls-trees: A model of evolutionary scenarios. *Theoret Comput Sci*, 359(1–3):378–399, 2006. doi:10.1016/j.tcs.2006.05.019.

9 L. Li and M. S. Bansal. An integrated reconciliation framework for domain, gene, and species level evolution. *IEEE/ACM Trans Comput Biol Bioinform*, 16(1):63–76, 2019. doi:10.1109/TCBB.2018.2846253.

10 Lei Li and Mukul S. Bansal. An integer linear programming solution for the domain-gene-species reconciliation problem. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, page 386–397, Washington, DC, USA, 2018. Association for Computing Machinery. doi:10.1145/3233547.3233603.

11 R Libeskind-Hadas and M Charleston. On the computational complexity of the reticulate cophylogeny reconstruction problem. *J Comput Biol*, 16:105–117, 2009. doi:10.1089/cmb.2008.0084.

12 Wayne P. Maddison. Gene trees in species trees. *Syst Biol*, 46(3):523–536, 1997. doi:10.1093/sysbio/46.3.523.

13 Y. Ovadia, D. Fielder, C. Conow, and R. Libeskind-Hadas. The cophylogeny reconstruction problem is NP-complete. *J Comput Biol*, 18(1):59–65, 2011. doi:10.1089/cmb.2009.0240.

14 Roderic D.M. Page. Maps between trees and cladistic analysis of historical associations among genes,organisms, and areas. *Syst Biol*, 43(1):58–77, 1994. doi:10.1093/sysbio/43.1.58.

15 P. Pamilo and M. Nei. Relationships between gene trees and species trees. *Mol Biol Evol*, 5(5):568–583, September 1988. doi:10.1093/oxfordjournals.molbev.a040517.

16 Roswitha Schmickl, Sarah Marburger, Sian Bray, and Levi Yant. Hybrids and horizontal transfer: introgression allows adaptive allele discovery. *J Exp Bot*, 68(20):5453–5470, 2017. doi:10.1093/jxb/erx297.

17 Maureen Stolzer, Han Lai, Minli Xu, Deepa Sathaye, Benjamin Vernot, and Dannie Durand. Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. *Bioinformatics*, 28(18):409–415, 2012. doi:10.1093/bioinformatics/bts386.

18 Fumio Tajima. Evolutionary relationship of DNA sequences in finite populations. *Genetics*, 105(2):437–460, 1983. doi:10.1093/genetics/105.2.437.

19 Thu-Hien To and Celine Scornavacca. Efficient algorithms for reconciling gene trees and species networks via duplication and loss events. *BMC Genomics*, 16(10):S6, October 2015. doi:10.1186/1471-2164-16-S10-S6.

20 Ali Tofigh, Michael Hallett, and Jens Lagergren. Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Trans Comput Biol Bioinform*, 8(2):517–535, March 2011. doi:10.1109/TCBB.2010.14.

21 Nicolas Wieseke, Tom Hartmann, Matthias Bernt, and Martin Middendorf. Cophylogenetic reconciliation with ILP. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 12(6):1227–1235, November 2015. doi:10.1109/TCBB.2015.2430336.

22 Taoyang Wu and Louxin Zhang. Structural properties of the reconciliation space and their applications in enumerating nearly-optimal reconciliations between a gene tree and a species tree. *BMC Bioinf*, 12(Suppl 9):S7–, 2011. doi:10.1186/1471-2105-12-S9-S7.

23 Yun Yu, R. Matthew Barnett, and Luay Nakhleh. Parsimonious inference of hybridization in the presence of incomplete lineage sorting. *Syst Biol*, 62(5):738–751, 2013. doi:10.1093/sysbio/syt037.

24 Yun Yu, Nikola Ristic, and Luay Nakhleh. Fast algorithms and heuristics for phylogenomics under ILS and hybridization. *BMC Bioinformatics*, 14(15):S6, October 2013. doi:10.1186/1471-2105-14-S15-S6.

25 Yun Yu, Cuong Than, James H. Degnan, and Luay Nakhleh. Coalescent histories on phylogenetic networks and detection of hybridization despite incomplete lineage sorting. *Syst Biol*, 60(2):138–149, 2011. doi:10.1093/sysbio/syq084.

26 Christian M. Zmasek and Sean R. Eddy. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics*, 17(9):821–828, September 2001. doi:10.1093/bioinformatics/17.9.821.

Efficient Privacy-Preserving Variable-Length Substring Match for Genome Sequence

Yoshiki Nakagawa
Department of Computer Science and Engineering, Waseda University, Tokyo, Japan

Satsuya Ohata
Digital Garage, Inc., Tokyo, Japan

Kana Shimizu[1]
Department of Computer Science and Engineering, Waseda University, Tokyo, Japan
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Abstract

Finding a similar substring that commonly appears in query and database sequences is an essential task for genome data analysis. This study proposes a secure two-party variable-length string search protocol based on secret sharing. The unique feature of our protocol is that time, communication, and round complexities are not dependent on the database length N, after the query input. This property brings dramatic performance improvements in search time, since N is usually quite large in an actual genome database, and the same database is repeatedly used for many queries. Our concept hinges on a technique that efficiently applies the compressed full-text index (FOCS 2000) for a secret-sharing scheme. We conducted an experiment using a human genomic sequence with the length of 10 million as the database and a query with the length of 100 and found that the query response time of our protocol was at least three orders of magnitude faster than a well-designed baseline protocol under the realistic computation/network environment.

2012 ACM Subject Classification Theory of computation → Pattern matching; Security and privacy → Privacy-preserving protocols; Theory of computation → Cryptographic protocols; Applied computing → Genomics

Keywords and phrases Private Genome Sequence Search, Secure Multiparty Computation, Secret Sharing, FM-index, Suffix Tree, Maximal Exact Match

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.2

Supplementary Material *Software (Source Code)*: `https://waseda.box.com/v/wabi2021-suppl-ppgs-src`

Funding This work is partially supported by JST CREST grant number JPMJCR19F6.
Kana Shimizu: Supported part in MEXT/JSPS KAKENHI Grant Number 19K12209 and 21H04871.

1 Introduction

The dramatic reduction in the cost of genome sequencing has prompted increased interest in personal genome sequencing over the last 15 years. Extensive collections of personal genome sequences have been accumulated both in academic and industrial organizations, and there is now a global demand for sharing the data to accelerate scientific research [13, 24]. As discussed in previous studies, disclosing personal genome information has a high privacy risk [10], so it is crucial to ensure that individuals' privacy is protected upon data sharing. At present, the most popular approach for this is to formulate and enforce a privacy policy, but it is a time-consuming process to reach an agreement, especially among stakeholders with different legal backgrounds, which slows down the pace of research. Therefore, there is a

[1] Corresponding author

strong demand for privacy-preserving technologies that can potentially compensate for or even replace the traditional policy-based approach [3, 21]. One important application that needs a privacy-preserving technology is private genome sequence search, where different stakeholders respectively hold a query sequence and a database sequence and the goal is to let the query holder know the result while simultaneously keeping the query and the database private. Many studies have addressed the problem of how to compute exact or approximate edit distance or the longest common substring (LCS) through techniques based on homomorphic encryption [17, 8, 22] and secure multi-party computation (MPC) [15, 31, 33, 7, 2, 26, 23], or how to compute sequence similarity based on private set intersection [4]. While these studies can evaluate global sequence similarity for two sequences of similar length, other studies address the problem of finding a substring between a query and a long genome sequence or a set of long genome sequences, with the aim of evaluating local sequence similarity [28, 16, 30, 29, 18, 6, 25]. [28] proposed an approach to combine an additive homomorphic encryption and index structures such as FM-index [11] and the positional Burrows-Wheeler transform [9] to find the longest prefix of a query that matches a database (LPM) and a set-maximal match for a collection of haplotypes. [30] used a similar approach and improved the time and communication complexities for LPM on a protein sequence by using a wavelet matrix. [16] improved the round complexity of a set-maximal match, though the search time was more than one order of magnitude slower than [28] due to the heavy computational cost caused by the fully homomorphic encryption. [29] used the Goldreich-Micali-Wigderson protocol to build a suffix tree for a set-maximal match. According to experiments by [18], the search time of [29] is one order of magnitude slower than [28] and [18]. [18] used a garbled circuit to build a suffix tree for substring match and a set-maximal match under a different security assumption such that the tree-traversal pattern is leaked to the cloud server. [6] and [25] found fixed-length substring matches using a one-way hash function or homomorphic encryption on a public cloud under a security assumption such that the database is a public sequence and a query is leaked to a private cloud server.

In this study, we aim to improve privacy-preserving substring match under the security assumption such that both the query and the database sequence are strictly protected. We first propose a more efficient method for finding LPM, and then extend it to find the longest maximal exact match (LMEM), which is more practically important in bioinformatics. We designed the protocol for LMEM for ease of explanation, and the protocol can be applied to similar problems such as finding all maximal exact matches (MEMs) with a small modification. To our knowledge, this is the first study to address the problem of securely finding MEMs.

Our Contribution

The time complexity of the previous studies [28, 30] include the factor of N, and thus they do not scale well to a large database. For a similar reason, using secure matching protocols (e.g., [32]) for the shares (or tags in searchable encryption) of all substrings in a query and database is even worse in terms of time complexity. To achieve a real-time search on an actual genome database, we propose novel secret-sharing-based protocols that do not include the factor of N in the time, communication, and round complexities for the search time (i.e., the time after the input of a query until the end of the search).

The basic idea of the protocols is to represent the database string by a compressed index [11, 12] and store the index as a lookup table. LPM and MEMs are found by at most ℓ and 2ℓ table lookups respectively, where ℓ is the length of the query. More specifically, the table V is referenced in a recursive manner; i.e., one needs to obtain $V[j]$, where $j = V[i]$, given i. To ensure security, we need to compute $V[j]$ without seeing any element of V.

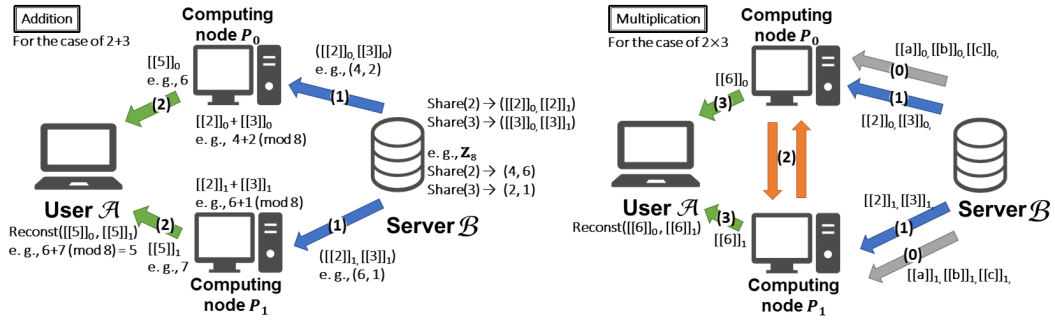


Figure 1 Arithmetic addition and multiplication over secret sharing.

The key technical contribution of this study is an efficient protocol that achieves this type of recursive reference. We named the protocol secret-shared recursive oblivious transfer (ss-ROT). While the previous studies requires $O(N)$ time complexity to ensure security, the time, communication, and round complexities of ss-ROT are all $O(\ell)$ for ℓ recursive table lookups, except for the preparation of the table and generation of shares before the query input. Since the entire protocols mainly consist of ℓ table lookups for LPM, and 2ℓ table lookups and 2ℓ inner product computations for LMEM, the search times for LPM and LMEM do not depend on the database size.

We implemented the proposed protocol and tested it on substrings of a human genome sequence 10^3 to 10^7 in length and confirmed that the actual CPU time and data transfer overhead were in good agreement with the theoretical complexities. We also found that the search time of our protocol was three orders of magnitude faster than that of the previous method [28, 30]. For conducting further performance analysis, we designed and implemented baseline protocols using major techniques of secret-sharing-based protocols. The results showed that the search times of our protocols were at least two orders of magnitude faster than those of the baseline protocols.

2 Preliminaries

2.1 Secure Computation based on Secret Sharing

Here, we explain the 2-out-of-2 additive secret sharing ($(2,2)$-SS) scheme and how to securely compute arithmetic/Boolean gates (Figure 1).

Secret Sharing and Secure Computation

In t-out-of-n secret sharing (e.g., [27]), we split the secret value x into n pieces, and can reconstruct x by combining more or an equal number of t pieces. We call the split pieces "share". The basic security notion for secret sharing is that we cannot obtain any information about x even if we gather less than or equal to $(t-1)$ shares. In this paper, we consider a case with $(t,n)=(2,2)$. A 2-out-of-2 secret sharing ($(2,2)$-SS) scheme over $\mathbb{Z}_{2^n}$ consists of two algorithms: Share and Reconst. Share takes as input $x \in \mathbb{Z}_{2^n}$ and outputs $(\llbracket x \rrbracket_0, \llbracket x \rrbracket_1) \in \mathbb{Z}_{2^n}^2$, where the bracket notation $\llbracket x \rrbracket_i$ denotes the arithmetic share of the i-th party (for $i \in \{0,1\}$). We denote $\llbracket x \rrbracket = (\llbracket x \rrbracket_0, \llbracket x \rrbracket_1)$ as their shorthand. Reconst takes as inputs $\llbracket x \rrbracket_0$ and $\llbracket x \rrbracket_1$ and outputs x. For arithmetic sharing $\llbracket x \rrbracket_i$ and Boolean sharing $\llbracket x \rrbracket_i^B$, we consider power-of-two integers n (e.g., $n=16$) and $n=1$, respectively.

Depending on the secret sharing scheme, we can compute arithmetic/Boolean gates over shares; that is, we can execute some kind of processing related to x without x. This means it is possible to perform some computation without violating the privacy of the secret data,

Table 1 Secure subprotocols used in this paper.

	Input	Output
Equality	$[\![x]\!]$, $[\![y]\!]$	$[\![z]\!]^B$ s.t. $z = 1$ if $x = y$ otherwise $z = 0$
Comp	$[\![x]\!]$, $[\![y]\!]$	$[\![z]\!]^B$ s.t. $z = 1$ if $x < y$ otherwise $z = 0$
CastUp	$[\![x]\!] \in \mathbb{Z}_{2^n}$, n'	$[\![x]\!] \in \mathbb{Z}_{2^{n'}}$ $(n < n')$
B2A	$[\![x]\!]^B$	$[\![x]\!]$
Choose	$[\![x]\!]$, $[\![y]\!]$, $[\![e \in \{0,1\}]\!]$	$[\![z]\!]$ s.t. $z = x$ if $e = 1$, otherwise $(e = 0)$ $z = y$

and is called secure (multi-party) computation. It is known that we can execute arbitrary computation by combining basic arithmetic/Boolean gates. In the following paragraphs, we show how to concretely compute these gates over shares.

Semi-Honest Secure Two-Party Computation Based on $(2, 2)$-Additive SS

We use a standard $(2, 2)$-additive SS scheme, defined by

- $\mathsf{Share}(x)$: randomly choose $r \in \mathbb{Z}_{2^n}$ and let $[\![x]\!]_0 = r$ and $[\![x]\!]_1 = x - r$.
- $\mathsf{Reconst}([\![x]\!]_0, [\![x]\!]_1)$: output $[\![x]\!]_0 + [\![x]\!]_1$.

Note that one of the shares of x ($[\![x]\!]_0$ or $[\![x]\!]_1$) does not reveal any information about x. In Figure 1, the secret value $x = 2$ is split into $[\![x]\!]_0 = 4$ and $[\![x]\!]_1 = 6$. These are valid $(2, 2)$-additive shares because $4 + 6 \equiv 2 \pmod 8$ holds. Even if we can see $[\![x]\!]_0 = 4$, we cannot decide the value of x since we execute a split of x uniformly at random. This means, in Figure 1, computing nodes P_0 and P_1 cannot obtain any information about x as long as these two nodes do not collude. On the other hand, we can compute arithmetic ADD/MULT gates over shares as follows:

- $[\![z]\!] \leftarrow \mathsf{ADD}([\![x]\!], [\![y]\!])$ can be done locally by just adding each party's share on x and on y. In Figure 1 (left), we show an example of secure addition. P_0/P_1 obtain shares 6/7 by adding their two shares. In this process, P_0/P_1 cannot find they are computing $2 + 3$.
- Multiplication is more complex than addition. There are various methods for multiplication over shares, most of which require communication between computing nodes. In this paper, we use the standard method for $[\![w]\!] \leftarrow \mathsf{MULT}([\![x]\!], [\![y]\!])$ based on Beaver triples (BT) [5]. Such a triple consists of $\mathsf{bt}_0 = (a_0, b_0, c_0)$ and $\mathsf{bt}_1 = (a_1, b_1, c_1)$ such that $(a_0 + a_1)(b_0 + b_1) = (c_0 + c_1)$. Hereafter, a, b, and c denote $a_0 + a_1$, $b_0 + b_1$, and $c_0 + c_1$, respectively. We use these BTs as auxiliary inputs for computing MULT. Note that we can compute them in advance (or in offline phase) since they are independent of inputs $[\![x]\!]$ and $[\![y]\!]$. We adopt a trusted initializer setting (e.g., [19, 20]); that is, BTs are generated by the party other than two computing nodes and then distributed. In the online phase of MULT, each i-th party P_i $(i \in \{0, 1\})$ can compute the multiplication share $[\![z]\!] = [\![xy]\!]$ as follows:
  1. P_i first computes $([\![x]\!]_i - a_i)$ and $([\![y]\!]_i - b_i)$, and sends them to P_{1-i}.
  2. P_i reconstructs $x' = x - a$ and $y' = y - b$.
  3. P_0 computes $[\![z]\!]_0 = x'y' + x'b_0 + y'a_0 + c_0$, and P_1 computes $[\![z]\!]_1 = x'b_1 + y'a_1 + c_1$. Here, $[\![z]\!]_0$ and $[\![z]\!]_1$ calculated with the above procedures are valid shares of xy; that is, $\mathsf{Reconst}([\![z]\!]_0, [\![z]\!]_1) = xy$. We shorten the notations and write the ADD and MULT protocols simply as $[\![x]\!] + [\![y]\!]$ and $[\![x]\!] \cdot [\![y]\!]$, respectively.

We also write $\mathsf{ADD}(\mathsf{ADD}([\![x_\mathrm{A}]\!], [\![x_\mathrm{B}]\!]), [\![x_\mathrm{C}]\!])$ as $\Sigma_{c=\{\mathrm{A},\mathrm{B},\mathrm{C}\}}[\![x_c]\!]$. Note that, similarly to the ADD protocol, we can also locally compute multiplication by constant c, denoted by $c \cdot [\![x]\!]$. We can easily extend the above protocols to Boolean gates. By converting $+$ and $-$ into

$\oplus$ in the arithmetic ADD and MULT protocols, we can obtain the XOR and AND protocols, respectively. We can construct NOT and OR protocols from the properties of these gates. When we compute $\mathsf{NOT}([\![x]\!]_0^B, [\![x]\!]_1^B)$, P_0 and P_1 output $\neg[\![x]\!]_0^B$ and $[\![x]\!]_1^B$, respectively. When we compute $\mathsf{OR}([\![x]\!]^B, [\![y]\!]^B)$, we compute $\neg\mathsf{AND}(\neg[\![x]\!]^B, \neg[\![y]\!]^B)$. We shorten the notations and write XOR, AND, NOT, and OR simply as $[\![x]\!] \oplus [\![y]\!]$, $[\![x]\!] \wedge [\![y]\!]$, $\neg[\![x]\!]$, and $[\![x]\!] \vee [\![y]\!]$, respectively. By combining the above gates, we can securely compute higher-level protocols. The functionality of the secure subprotocols [23] used in this paper are shown in Table 1. Due to space limits, we omit the details of their construction. Note that we can compute Choose by $[\![z]\!] = [\![y]\!] + [\![e]\!] \cdot ([\![x]\!] - [\![y]\!])$. In this paper, we consider the standard simulation-based security notion in the presence of semi-honest adversaries (for 2PC), as in [14]. We show the definition in Appendix B. Roughly speaking, this security notion guarantees the privacy of the secret under the condition that computing nodes do not deviate from the protocol; that is, although computing nodes are allowed to execute arbitrary attacks in their local, they do not (maliciously) manipulate transmission data to other parties. The building blocks we adopt in this paper satisfy this security notion. Moreover, as described in [14], the composition theorem for the semi-honest model holds; that is, any protocol is privately computed as long as its subroutines are privately computed.

2.2 Index Structure for String Search

Notation and Definition

Σ denotes a set of ordered symbols. A string consists of symbols in Σ. We denote a lexicographical order of two strings S and S' by $S \leq S'$ (i.e., A < C < G < T and AAA < AAC). We denote the i-th letter of a string S by $S[i]$ and a substring starting from the i-th letter to the j-th letter by $S[i, j]$. The index starts with 0. The length of S is denoted by $|S|$. A reverse string of S (i.e., $S[|S|-1], \ldots, S[0]$) is denoted by $\hat{S}$. We consider a direction from the i-th position to the j-th position as rightward if $i < j$ and leftward otherwise.

Given a query w and a database S, we define the longest prefix that matches a database string (LPM) by $\max_{(0,j)}\{j | w[0, \ldots, j] = S[k, \ldots, l]\}$, where $0 \leq j < \ell$ and $0 \leq k \leq l < N$, and the longest maximal exact match (LMEM) by $\max_{(i,j)}\{j - i | w[i, \ldots, j] = S[k, \ldots, l]\}$, where $0 \leq i \leq j < \ell$ and $0 \leq k \leq l < N$.

FM-Index and related data structures

FM-Index [11] and related data structures [12] are widely used for genome sequence search. Given a query string w of length ℓ and a database string S of length N, [11] enables LPM to be found in $O(\ell)$ time regardless of N, and it also enables LMEM to be found in $O(\ell)$ if auxiliary data structures are used [12]. Given all the suffixes of a string S: $S[0, \ldots, |S|-1]$, $S[1, \ldots, |S|-1], \ldots, S[|S|-1]$, a suffix array is an array of positions $(p_0, \ldots, p_{|S|-1})$ such that $S[p_0, \ldots, |S|-1] \leq S[p_1, \ldots, |S|-1] \leq S[p_2, \ldots, |S|-1], \ldots, \leq S[p_{|S|-1}, \ldots, |S|-1]$. We denote the suffix array of S by SA and denote its i-th element by $SA[i]$. A Burrows-Wheeler transform (BWT) is a permutation of the sequence S such that its i-th letter becomes $S[SA[i]-1]$. We denote a BWT of S by L and denote its i-th letter by $L[i]$. Let us define a rank of S for a letter $c \in \Sigma$ at position t by $\mathsf{Rank}_c(t, S) = |\{j | S[j] = c, 0 \leq j < t\}|$ and a count of occurrences of letters that are lexicographically smaller than c in S by $\mathsf{CF}_c(S) = \sum_{r<c} \mathsf{Rank}_r(|S|, S)$, and the operation $\mathsf{LF}_c(i, S) = \mathsf{CF}_c(L) + \mathsf{Rank}_c(i, L)$. The match between w and S is reported as a form of left-closed and right-open interval on SA, and the lower and upper bounds of the interval are respectively computed by LF. Given a

letter c and an interval $[f, g)$ that corresponds to suffixes that share the prefix x (i.e., $[f, g)$ reports the locations of the substring x in S), we can find a new interval that corresponds to all suffixes that share the prefix cx (i.e., locations of the substring cx) by

$$[f', g') = [\mathsf{LF}_c(f, S), \mathsf{LF}_c(g, S)). \tag{1}$$

The leftward extension of the match is called a backward search, which is the main functionality of FM-Index. By starting the search with the initial interval $[0, N)$ and conducting the backward searches for $w[\ell-1], w[\ell-2], \ldots$, the longest suffix match is detected when $f = g$. Rank and CF are precomputed and stored in an efficient form that can be searched in constant time. Therefore, the longest suffix match can be computed in $O(\ell)$ time. LPM is found if the search is conducted on $\hat{S}$ and match is extended by $w[0], w[1], \ldots, w[\ell-1]$.

Searching LMEM by repeating LPM for $w[0, \ldots, \ell-1], w[1, \ldots, \ell-1], w[2, \ldots, \ell-1], \ldots, w[\ell-1]$ takes $O(\ell^2)$ time. We can improve it to $O(\ell)$ time by using the longest common prefix (LCP) array and related data structures [12]. The LCP array, denoted by LCP, is an array that stores the length of the longest prefix of $S[\mathsf{SA}[i-1], |S|-1]$ and $S[\mathsf{SA}[i], |S|-1]$ in $\mathsf{LCP}[i]$ for $0 < i \leq N$. The lcp-interval $[i, j)$ of lcp-value d is an interval such that it satisfies $\mathsf{LCP}[i] < d$, $\mathsf{LCP}[j] < d$, $\mathsf{LCP}[k] > d$ for all $k \in \{i+1, \ldots, j-1\}$, and $\mathsf{LCP}[k] = d$ for at least one $k \in \{i+1, \ldots, j-1\}$, and is denoted by $d-[i, j)$. $d-[i, j)$ corresponds to all the suffixes that share the prefix $S[SA[i], \ldots, SA[i]+d-1]$. The parent interval of $d-[i, j)$ is the lcp-interval $h-[m, n)$ such that $h < d$ and $0 \leq m \leq i < j \leq n < N$, and there is no other lcp-interval $t-[r, s)$ such that $h < t < d$ and $0 \leq m \leq r \leq i < j \leq s \leq n < N$. The parent of the lcp-interval $[f, g)$ can be found by

$$[f', g') = \begin{cases} [\mathsf{PSV}[f_i], \mathsf{NSV}[f_i]) & \mathsf{LCP}[g_i] \leq \mathsf{LCP}[f_i] \\ [\mathsf{PSV}[g_i], \mathsf{NSV}[g_i]) & (otherwise)\ , \end{cases} \tag{2}$$

where $\mathsf{PSV}[i] = \max\{j | 0 \leq j < i \wedge \mathsf{LCP}[j] < \mathsf{LCP}[i]\}$ and $\mathsf{NSV}[i] = \min\{j | i \leq j < N \wedge \mathsf{LCP}[j] < \mathsf{LCP}[i]\}$. By finding a parent interval using PSV and NSV whenever it fails to extend the match, we can avoid useless backward searches, and thus LMEM is found at most 2ℓ backward searches. LCP, PSV and NSV are precomputed and stored in an efficient form that can be searched in constant time, so we can find LMEM in $O(\ell)$ time. See section 5.2 of [12] for more details of the data structures. Examples of the search by FM-Index, LCP, PSV, and NSV are provided in Appendix A.

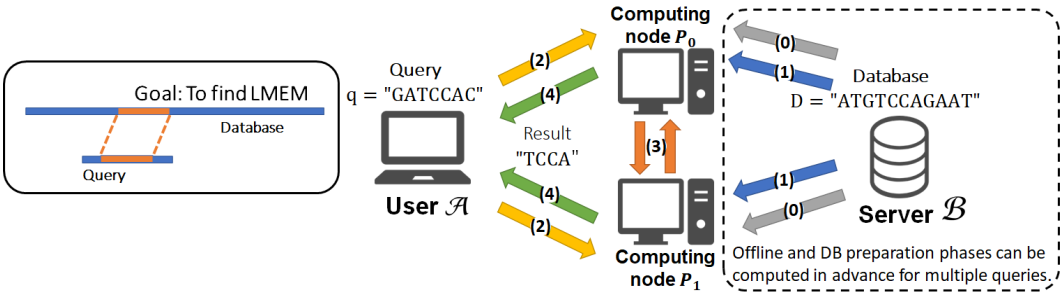


Figure 2 Schematic view of our goal and model. (0) Server (DB holder) distributes Beaver triples. (A reliable third party can serve as the trusted initializer instead.) (1) Server distributes shares of the database. (2) User (query holder) distributes shares of the query. (3) The computing nodes jointly calculate shares of the result. (4) The results are sent to User. The offline phase is (0), DB preparation phase is (1), and Search phase consists of (2)–(4).

Table 2 Summary of complexities for our protocols and related protocols. BTime and Bsize are generation time and size of BTs. Dtime and Dsize are generation time for the shares of the database and size of the shares. Stime is the time for Search phase. Comm. is the size of data exchanged between computing nodes. Round is the number of data exchanges.

	Btime	Bsize	Dtime	Dsize	Stime	Comm.	Round
ss-ROT (ours)	0	0	ℓN	ℓN	ℓ	ℓ	ℓ
Secure LPM (ours)	ℓ	ℓ	ℓN	ℓN	ℓ	ℓ	ℓ
[30, 28] (LPM by AHE)	–	–	–	–	ℓN	$\ell\sqrt{N}$	ℓ
Baseline (LPM)	$\ell^2 N$	$\ell^2 N$	N	N	$\ell^2 N$	$\ell^2 N$	$\log \ell + \log N$
Secure LMEM (ours)	ℓ^2	ℓ^2	ℓN	ℓN	ℓ^2	ℓ^2	ℓ
Baseline (LMEM)	$\ell^3 N$	$\ell^3 N$	N	N	$\ell^3 N$	$\ell^3 N$	$\log \ell + \log N$

3 Proposed protocols

Problem Setting and Outline of Our Protocols

We assume that a query holder $\mathcal{A}$, a database holder $\mathcal{B}$, and two computing nodes P_0 and P_1 participate the protocol. $\mathcal{A}$ holds a query string w of length ℓ and $\mathcal{B}$ holds a database string T of length N. After the protocol is run, only $\mathcal{A}$ knows LPM or LMEM between w and T. P_0 and P_1 do not obtain any information of w and T, except for ℓ and N.

Our protocol consists of offline, DB preparation, and Search phases. In the offline phase, $\mathcal{B}$ generates BTs (correlated randomness used for multiplication) and sends them to P_0 and P_1. In the DB preparation phase, $\mathcal{B}$ creates a lookup table and distributes its shares to P_0 and P_1. In the Search phase, $\mathcal{A}$ generates shares of the query and sends them to P_0 and P_1, and P_0 and P_1 jointly compute the result without obtaining any information of the lookup table. Finally, $\mathcal{A}$ obtains the results. Figure 2 shows the schematic view of our goal and model. Note that the Offline and DB preparation phases do not depend on a query string, so they can be computed in advance for multiple queries.

In Section 3.1, we propose the important building block ss-ROT that enables recursive reference to a lookup table. In Section 3.2, we describe how to design the lookup table based on FM-Index, and propose an efficient protocol for LPM by using the lookup table and ss-ROT. In Section 3.3, we describe the additional table design for auxiliary data structures, and propose the complete protocol for LMEM. Table 2 summarizes the theoretical complexities of the three protocols. For comparison, the complexities of the baseline protocols and a previous method for LPM based on an additive homomorphic encryption [28, 30] are shown. As we mentioned in Section 1, the baseline protocols are designed using major techniques of secret-sharing-based protocols. The detailed algorithms are described in Appendix C.

3.1 Secret-shared Recursive Oblivious Transfer

We define a problem called a secret-shared recursive oblivious transfer (ss-ROT) as follows.

▶ **Definition 1.** *We assume a database holder $\mathcal{B}$ and two computing nodes P_0 and P_1 participate the protocol. $\mathcal{B}$ holds a vector V of length N and $0 \le V[i] < N$. Given the initial position p_0 and the depth of recursion ℓ $(2 \le \ell)$, the secret-shared recursive oblivious transfer protocol outputs shares of*

$$\underbrace{V[V[\cdots V}_{\ell}[p_0]\cdots]] \tag{3}$$

without leaking V to P_0 and P_1.

Protocol 1 Secret-shared Recursive Oblivious Transfer (ss-ROT).

Input: Public input: p_0
Input: Private input of server: $R^j[i]$ $(i = 0, \ldots, N-1,\ j = 0, \ldots, \ell - 1)$
1: (Preparation by $\mathcal{B}$) $\mathcal{B}$ generates and distributes $[\![R^j[i]]\!]_0$ and $[\![R^j[i]]\!]_1$ to P_0 and P_1
2: **for** $0 \le j \le \ell - 2$ **do** ▷ **Step 1**
3: P_0 and P_1 obtain a position p_{j+1} by $p_{j+1} = \mathsf{Reconst}([\![R^j[p_j]]\!]_0, [\![R^j[p_j]]\!]_1)$.
4: **end for**
5: P_0 and P_1 output $[\![R^{\ell-1}[p_{\ell-1}]]\!]_0$ and $[\![R^{\ell-1}[p_{\ell-1}]]\!]_1$. ▷ **Step 2**

For simplicity, we denote the recursion of Eq. 3 by $V^{(\ell)}[p_0]$ (e.g., $V[V[p_0]]$ is denoted by $V^{(2)}[p_0]$). In our protocol, all the random values are uniformly generated from $\mathbb{Z}_{2^n}$.

DB Preparation Phase. $\mathcal{B}$ generates $\ell - 1$ random values $r^0, \ldots, r^{\ell-2}$ and computes the following vectors $R^0, \ldots, R^{\ell-1}$. Each vector R^j has N elements.

$$R^j[i] = \begin{cases} (V[i] + r^j) \bmod N & (j = 0) \\ (V[(i - r^{j-1}) \bmod N] + r^j) \bmod N & (1 \le j \le \ell - 2) \\ (V[(i - r^{j-1}) \bmod N]) \bmod N & (j = \ell - 1) \end{cases} \tag{4}$$

$\mathcal{B}$ computes $\mathsf{Share}(R^j[i])$ and sends $[\![R^j[i]]\!]_0$ and $[\![R^j[i]]\!]_1$ to P_0 and P_1, for $i = 0, \ldots, N-1$ and $j = 0, \ldots, \ell - 1$.

Search Phase. The Search phase consists of two steps and is described in Lines 2–5 of Protocol 1. The input is the initial position p_0 and shares of R. The output is $[\![V^{(\ell)}[p_0]]\!]$. An example of a search is illustrated in Figure 3.

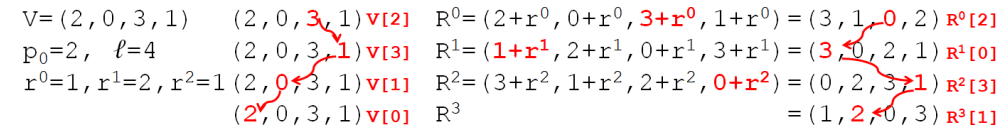


Figure 3 Example of a search when $V = (2, 0, 3, 1)$, $p_0 = 2$, and $\ell = 4$. The goal is to compute $[\![V^{(4)}[2]]\!] = [\![2]\!]$. Here we assume $\mathcal{B}$ generates $r^0 = 1, r^1 = 2, r^2 = 1$. In Step 1 of Search phase, P_0 and P_1 jointly compute $\mathsf{Reconst}([\![R^0[2]]\!]_0, [\![R^0[2]]\!]_1)$ to obtain $R^0[2] = 0$. ($R^0[2]$ is randomized by r^0, so any element of V is leaked.) In a similar way, P_0 and P_1 compute $R^1[0] = 3$ and $R^2[3] = 1$. In Step 2, P_0 and P_1 output $[\![R^3[1]]\!]_0$ and $[\![R^3[1]]\!]_1$ respectively. Since $R^0[2] = V[2] + r^0$, $R^1[V[2] + r^0] = V[V[2] + r^0 - r^0] + r^1$, $R^2[V[V[2]] + r^1] = V[V[V[2]] + r^1 - r^1] + r^2$, and $R^3[V[V[V[2]]] + r^2] = V[V[V[V[2]]] + r^2 - r^2]$, ss-ROT successfully computes $[\![V^{(4)}[2]]\!]$.

3.1.1 Security and Complexities

▶ **Theorem 2.** *ss-ROT is correct and secure in the semi-honest model.*

Due to space limits, the proof is shown in Appendix D.

In the DB preparation phase, $\mathcal{B}$ generates shares of V of length N for ℓ times. Therefore, time and communication complexities are $O(\ell N)$. For the Search phase, $\mathsf{Reconst}$ is computed ℓ times in Step 1. Since the time, communication, and round complexities of $\mathsf{Reconst}$ are $O(1)$, those of the Search phase become $O(\ell)$.

3.2 Secure LPM

Construction of Lookup Table. The goal is to find LPM securely. To apply FM-Index for a prefix search, the reverse string of T (i.e., $\hat{T}$) is used. The backward search of FM-Index is formulated by Eq. 1. If we precompute $\mathsf{LF}_c(i, \hat{T})$ for $i = 0, \ldots, N$ and $c \in \{\mathrm{A},\mathrm{T},\mathrm{G},\mathrm{C}\}$, and store them in a lookup table that consists of four vectors: V_A, V_C, V_G, and V_T such that $V_c[i] = \mathsf{LF}_c(i, \hat{T})$, Eq. 1 is replaced by the following table lookup

$$f_{k+1} = V_{w[k]}[f_k], \qquad g_{k+1} = V_{w[k]}[g_k]. \tag{5}$$

I.e., starting with the initial interval $[f_0 = 0, g_0 = N)$, we can compute the match by recursively referring to the lookup table while $f < g$.

Protocol Overview. The key idea of Secure LPM is to refer to V by ss-ROT, i.e., P_0 and P_1 jointly refer to V ℓ times in a recursive manner. To achieve backward search, P_0 and P_1 need to select $V_x[\cdot]$ for each reference, where x is a query letter to be searched with. This is achieved by expressing the query letter by unary code (Eq. 7) and computing the inner product of Eq. 7 and $(V_\mathrm{A}[\cdot], V_\mathrm{C}[\cdot], V_\mathrm{G}[\cdot], V_\mathrm{T}[\cdot])$. To find LPM, P_0 and P_1 need to check $f = g$ for each reference. We use the subprotocol Equality to check it securely. Since V is randomized with different numbers for searching f and g, the difference of the random numbers is precomputed and removed securely upon the equality check. $\mathcal{A}$ receives only the result of each equality check to know LPM. For examples, LPM is the prefix of length $i-1$ when $f = g$ for the i-th reference. If $f \neq g$ for all references, LPM is the entire query.

DB Preparation Phase. $\mathcal{B}$ creates a lookup table and generates the following 4ℓ vectors in a similar manner to ss-ROT. For simplicity, we denote the length of V_c by $N' = N + 1$.

$$R^j_{c,f}[i] = \begin{cases} (V_c[i] + r^j_f) \bmod N' & (j = 0) \\ (V_c[(i - r^{j-1}_f) \bmod N'] + r^j_f) \bmod N' & (1 \le j < \ell) \end{cases} \tag{6}$$

$R^j_{c,f}[i]$ is used for computing the lower bound f of the interval $[f, g)$. We also generate $R^j_{c,g}[i]$ for the upper bound g. R consists of 8ℓ vectors, each of length N'. Since the longest match is found when $f = g$, $\mathcal{B}$ also generates a vector $r'[j] = r^j_f - r^j_g$ that is used for equality check of f and g. Then, $\mathcal{B}$ sends shares of $R^j_{c,f}[i]$, $R^j_{c,g}[i]$, and $r'[j]$ to P_0 and P_1.

Search Phase. Protocol 2 describes the algorithm in detail. $\mathcal{A}$ generates four vectors q_A, q_C, q_G, q_T, each of length ℓ, as follows.

$$q_c[j] = \begin{cases} 1 & (c = w[j]) \\ 0 & (c \neq w[j]) \end{cases} \tag{7}$$

For each j, $(q_\mathrm{A}[j], q_\mathrm{C}[j], q_\mathrm{G}[j], q_\mathrm{T}[j])$ encodes $w[j]$ (e.g., $(q_\mathrm{A}[j], q_\mathrm{C}[j], q_\mathrm{G}[j], q_\mathrm{T}[j]) = (1, 0, 0, 0)$ if $w[j] = \mathrm{A}$). The aim of the encode is to compute $[\![R_x[j]]\!] = [\![\sum_{c\in\Sigma} q_c[j] \cdot R_c[j]]\!]$ when $w[j] = x$. Figure 4 illustrates an example of the table lookup.

$\mathcal{A}$ generates shares of q_A, q_C, q_G, q_T and distributes them to P_0 and P_1. P_0 and P_1 compute $\mathsf{LF}_{w[j]}(f', \hat{T}) + r^j_f$ and $\mathsf{LF}_{w[j]}(g', \hat{T}) + r^j_g$ in Lines 5–8 without leaking f' and g', where $[f', g')$ corresponds to the match of $w[0, j]$ and $\hat{T}$. In Lines 10–12, the equality of f' and g' is examined for all rounds. Note that f_j and g_j are randomized by different values r^{j-1}_f and r^{j-1}_g in order to conceal f' and g', so our protocol computes the equality of $f_j - g_j - r'[j-1]$ and 0. In Lines 15–16, $\mathcal{A}$ receives all the results of equality checks (i.e., $[\![o[1]]\!]^B, \ldots, [\![o[\ell]]\!]^B$) from P_0 and P_1, and knows LPM by reconstructing them. For example, if w =GCT and $o = (0, 0, 1)$, $\mathcal{A}$ knows that LPM is GC.

Protocol 2 Secure LPM.

Input: Public input: N, $N' = N + 1$, ℓ, $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$, $f_0 = 0$, $g_0 = N$
Input: Private input of user: query $q_c \in \{0,1\}^\ell$ $(c \in \Sigma)$
Input: Private input of server: $R^j_{c,f}, R^j_{c,g} \in \mathbb{Z}_{N'}^{N'}$ $(c \in \Sigma, 0 \le j < \ell)$, $r' \in \mathbb{Z}^\ell_{N'}$

1: (Preparation by $\mathcal{B}$) $\mathcal{B}$ distributes $[\![R^j_{c,f}]\!]$, $[\![R^j_{c,g}]\!] (c \in \Sigma, 0 \le j < \ell)$, $[\![r']\!]$ to P_0 and P_1
2: (Preparation by $\mathcal{A}$) $\mathcal{A}$ distributes $[\![q_c]\!]$ $(c \in \Sigma)$ to P_0 and P_1.
3: (Computation by P_0 and P_1)
4: **for** $j = 0, \cdots, \ell - 1$ **do**
5: $\quad [\![f_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,f}[f_j]]\!], [\![q_c[j]]\!])$ ▷ Select $[\![R^j_{w[j],f}[f_j]]\!]$
6: $\quad [\![g_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,g}[g_j]]\!], [\![q_c[j]]\!])$ ▷ Select $[\![R^j_{w[j],g}[g_j]]\!]$
7: $\quad f_{j+1} \leftarrow \mathsf{Reconst}([\![f_j]\!]_0, [\![f_j]\!]_1)$ ▷ Update randomized lower bound
8: $\quad g_{j+1} \leftarrow \mathsf{Reconst}([\![g_j]\!]_0, [\![g_j]\!]_1)$ ▷ Update randomized upper bound
9: **end for**
10: **for** $j = 1, \cdots, \ell$ **parallelly do**
11: $\quad [\![o[j]]\!]^B \leftarrow \mathsf{Equality}([\![f_j]\!] - [\![g_j]\!] - [\![r'[j-1]]\!], [\![0]\!])$ ▷ Equality check of upper and lower bounds.
12: **end for**
13: P_0 and P_1 send $[\![o]\!]^B_0, [\![o]\!]^B_1$ to $\mathcal{A}$
14: (Verification by $\mathcal{A}$)
15: **for** $j = 1, \cdots, \ell$ **do**
16: $\quad o[j] \leftarrow \mathsf{Reconst}([\![o[j]]\!]^B_0, [\![o[j]]\!]^B_1)$
17: **end for**

Output: $\mathcal{A}$ outputs $o[1], \ldots, o[\ell]$ to determine LPM.

3.2.1 Security and Complexities

▶ **Theorem 3.** *Protocol 2 is correct and secure in the semi-honest model.*

Due to space limits, the proof is shown in Appendix E. $\mathcal{A}$ may reveal T by making many queries. Such a problem is called output privacy. Although output privacy is outside of the scope of this paper, we should mention here that $\mathcal{A}$ needs to make an unrealistically large number of queries for obtaining T by such a brute-force attack, considering that N is very long.

The DB preparation phase generates shares of $R^j_{c,f}$ and $R^j_{c,g}$ ($c \in \Sigma$ and $0 \le j < \ell$); i.e., $8 \times \ell$ vectors of length N'. Therefore, the time and communication complexities are $O(\ell N)$. For the Search phase, MULT and Reconst are computed twice in Lines 4–9 for ℓ rounds and Equality is computed once in Lines 10–12 for ℓ rounds. Each time, the communication and round complexities of these subprotocols are $O(1)$, so those of the Search phase become $O(\ell)$.

3.3 Secure LMEM

Construction of Lookup Table. As described in Section 2.2, we can find a parent interval by a reference to LCP, PSV, and NSV. Therefore, in addition to V_c defined in Section 3.2, we prepare lookup tables that simply store all the outputs of them; i.e., $V_{lcp}[i] = \mathsf{LCP}[i]$, $V_{psv}[i] = \mathsf{PSV}[i]$, and $V_{nsv}[i] = \mathsf{NSV}[i]$.

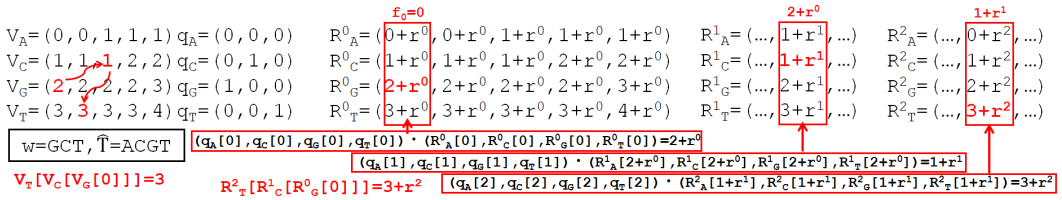


Figure 4 Example of a secure table lookup when w =GCT and $\hat{T}$=ACGT. Only the lookup for a lower bound is shown. For simplicity, $R^j_{c,f}$ and r^j_f are denoted by R^j_c and r^j. $\mathsf{LF}_{w[i]}(f_i, \hat{T})$ $(i = 0, 1, 2)$ is computed by $V_{\mathrm{G}}[0], V_{\mathrm{C}}[2]$, and $V_{\mathrm{T}}[1]$. V is referenced securely by using R. $R^0_{\mathrm{G}}[0]$ is computed by $\sum_{c\in\Sigma} q_c[0] \cdot R_c[0]$. $R^1_{\mathrm{C}}[2 + r^0]$ is computed by $\sum_{c\in\Sigma} q_c[1] \cdot R_c[2 + r^0]$. $R^2_{\mathrm{T}}[1 + r^1]$ is computed by $\sum_{c\in\Sigma} q_c[2] \cdot R_c[1 + r^1]$.

DB Preparation Phase. $\mathcal{B}$ generates randomized vectors $R_{c,f}$, $R_{c,g}$ and $r'[j] = r^j_f - r^j_g$ using the same algorithm in Section 3.2 for length 2ℓ. As shown in Eq. 2, V_{lcp} is referred by the upper and lower bounds of $[f, g)$. Therefore, $\mathcal{B}$ generates following circular permutations of V_{lcp} such that $W_{l,f}$ and $R_{c,f}$, and $W_{l,g}$ and $R_{c,g}$, are permutated by the same random values, respectively. I.e.,

$$W^j_{l,x}[i] = \begin{cases} V_{lcp}[i] & (j = 0) \\ V_{lcp}[(i - r^{j-1}_x) \bmod N] & (1 \le j < 2\ell) \end{cases},$$

where x is either f or g. V_{psv} is referred by both f and g, and is plugged in to f. Therefore, $\mathcal{B}$ generates $W^j_{p,f}$ and $W^j_{p,g}$ such that both of them are randomized by r^j_f, and $W^j_{p,f}$ is permutated by r^{j-1}_f and $W^j_{p,g}$ is permutated by r^{j-1}_g as follows.

$$W^j_{p,f}[i] = \begin{cases} (V_{psv}[i] + r^j_f) \bmod N & (j = 0) \\ (V_{psv}[(i - r^{j-1}_f) \bmod N] + r^j_f) \bmod N & (1 \le j < 2\ell) \end{cases}$$

$$W^j_{p,g}[i] = \begin{cases} (V_{psv}[i] + r^j_g) \bmod N & (j = 0) \\ (V_{psv}[(i - r^{j-1}_g) \bmod N] + r^j_f) \bmod N & (1 \le j < 2\ell) \end{cases}$$

Similarly, V_{nsv} is referred by both f and g, and is plugged in to g. Therefore, $\mathcal{B}$ generates $W^j_{n,f}[i]$ and $W^j_{n,g}[i]$ as follows.

$$W^j_{n,f}[i] = \begin{cases} (V_{nsv}[i] + r^j_f) \bmod N & (j = 0) \\ (V_{nsv}[(i - r^{j-1}_f) \bmod N] + r^j_g) \bmod N & (1 \le j < 2\ell) \end{cases}$$

$$W^j_{n,g}[i] = \begin{cases} (V_{nsv}[i] + r^j_g) \bmod N & (j = 0) \\ (V_{nsv}[(i - r^{j-1}_g) \bmod N] + r^j_g) \bmod N & (1 \le j < 2\ell) \end{cases}$$

$\mathcal{B}$ distributes shares of $R_{c,f}$, $R_{c,g}$, r', $W_{l,f}$, $W_{l,g}$, $W_{p,f}$, $W_{p,g}$, $W_{n,f}$, and $W_{n,g}$ to P_0 and P_1.

Search Phase. Protocol 3 describes the algorithm in detail. $\mathcal{A}$ generates query vectors q_{A}, q_{C}, q_{G}, q_{T} by Eq. 7 and distributes shares of the vectors to P_0 and P_1. In Line 6 of Protocol 3, $[\hat{f}, \hat{g})$ is computed by the reference to R (i.e., a search based on a backward search) similarly to Lines 5–6 of Protocol 2. In Line 9, $[f_{ex}, g_{ex})$ is computed by the reference to W (i.e., a search based on LCP, PSV and NSV). In Line 11, the interval is updated by either $[\hat{f}, \hat{g})$ or $[f_{ex}, g_{ex})$ based on the result of $f' = g'$ in Line 7, where $[f', g')$ corresponds to the interval that corresponds to a substring match.

In each round, we need to know a query letter to be searched with, so we need to maintain the right end position of the match in the query. The position moves toward the right while the match is extended, but remains the same when the interval is updated based on PSV and NSV. To memorize the position, we prepare shares of a unit bit vector u of length ℓ, in which the position t is memorized as $u[t] = 1$ and $u[i \neq t] = 0$. In Lines 14–18, u remains the same if the interval is updated based on PSV and NSV, and $u = (u, [\ell - 1], u[0], u[1], \ldots, u[\ell - 2])$ otherwise. In Lines 19–21, the inner product of q_c ($c \in \Sigma$) and u becomes the encode of $w[t]$ that is used for the next round.

We also maintain the left end position of the match. While the match is extended, the position remains the same and it moves toward the right when the interval is updated by $[f_{ex}, g_{ex})$. The new left end position can be computed by $p + m - c$ where p is the current position, m is the length of the current match, and c is the lcp-value of $[f_{ex}, g_{ex})$ (i.e., the longest common prefix length of suffixes contained in $[f_{ex}, g_{ex})$). The position is computed in Line 23. The match length is incremented by 1 for each extension. When the interval is updated by $[f_{ex}, g_{ex})$, the match length is reduced to the lcp-value of $[f_{ex}, g_{ex})$, which is computed by $\max(\mathsf{LCP}[f], \mathsf{LCP}[g])$. The match length is computed in Line 22. In Line 25, the longest match length and the corresponding left end position are updated. After all the positions in the query have been examined, LMEM and its left end position are sent to $\mathcal{A}$ in Line 27.

3.3.1 Security and Complexities

▶ **Theorem 4.** *Protocol 3 is correct and secure in the semi-honest model.*

Due to space limits, the proof is shown in Appendix F.

The DB preparation phase generates shares of $R^j_{c,f}$ and $R^j_{c,g}$ ($c \in \Sigma$, $0 \leq j < \ell$) and $W^j_{x,f}$ and $W^j_{x,g}$ ($x \in \{l, p, n\}$ and $0 \leq j < \ell$); $14 \times \ell$ vectors of length $N + 1$. Therefore, the time and communication complexities are $O(\ell N)$. For the Search phase, MULT is computed ℓ times in parallel in Lines 15–16. (These are not dependent on each other.) In Line 20, MULT is computed ℓ times in parallel, and Line 20 is computed in parallel four times in Lines 19–21. Lines 15–16 and Lines 19–21 are repeated for $2\ell - 1$ rounds. Other subprotocols are also computed for $2\ell - 1$ rounds. The time, communication, and round complexities are $O(1)$ for MULT, and independent computation of MULT for ℓ times does not increase the round complexity. The time, communication and round complexities are $O(1)$ for the other subprotocols used in Protocol 3. Therefore, the complexities of the Search phase are $O(\ell^2)$ for time and communication, and $O(\ell)$ for the number of rounds.

4 Experiment

We implemented Protocol 2 (Secure LPM) and Protocol 3 (Secure LMEM). For comparison, we also implemented baseline protocols (Baseline (LPM) and Baseline (LMEM)). Details of the baseline protocols are provided in Appendix C. All protocols were implemented by Python 3.5.2. The dataset was created from Chromosome 1 of the human genome. We extracted substrings of length $N = 10^3$, 10^4, 10^5, 10^6, and 10^7 for databases, and $\ell = 10$, 25, 50, 75, and 100 for queries. Share was run with $n = 16$ and $n = 32$ for $N < 10^5$ and $10^5 \leq N$ in the proposed protocols, and $n = 1$ for a Boolean share and $n = 8$ for an arithmetic share in the baseline protocols. We did not implement a data transfer module, and each protocol is implemented as a single program. Therefore, the search time of the protocols was measured by the time consumed by either one of P_0 and P_1. To assess the influence of communication

Protocol 3 Secure LMEM.

Input: Public input: N, $N' = N+1$, ℓ, $\Sigma = \{\mathrm{A}, \mathrm{T}, \mathrm{G}, \mathrm{C}\}$, $f_0 = 0$, $g_0 = N$
Input: Private input of user: query $q_c \in \{0,1\}^\ell$ $(c \in \Sigma)$
Input: Private input of server: $R^j_{c,f}, R^j_{c,g}, W^j_{l,f}, W^j_{l,g}, W^j_{p,f}, W^j_{p,g}, W^j_{n,f}, W^j_{n,g} \in \mathbb{Z}_{N'}^{N'}$ $(c \in \Sigma, 0 \le j < \ell)$, $r' \in \mathbb{Z}_{N'}^{\ell}$

(Preparation by $\mathcal{B}$) $\mathcal{B}$ generates shares of input vectors. $\mathcal{B}$ also generates shares of variables: $[\![u]\!] = [\![(1,0,\ldots,0)]\!]$, $[\![p_{max}]\!] = [\![0]\!]$, $[\![p]\!] = [\![0]\!]$, $[\![m_{max}]\!] = [\![0]\!]$, $[\![m]\!] = [\![0]\!]$. All shares are sent to P_0 and P_1.
(Preparation by $\mathcal{A}$) $\mathcal{A}$ generates and distributes $[\![q_c]\!]$ $(c \in \Sigma)$ to P_0 and P_1.
(Computation by P_0 and P_1)
Set shares of the initial letter $[\![z_c]\!] = [\![q_c[0]]\!]$ $(c \in \Sigma)$.
for $j = 0, \cdots, 2\ell - 1$ **do**
  $[\![\hat{f}_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,f}[f_j]]\!], [\![z[c]]\!])$, $[\![\hat{g}_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,g}[g_j]]\!], [\![z[c]]\!])$
  $[\![e1]\!]^B \leftarrow \mathsf{Equality}([\![\hat{f}_j]\!] - [\![\hat{g}_j]\!] - [\![r'[j-1]]\!], [\![0]\!])$, $[\![e1]\!] \leftarrow \mathsf{B2A}([\![e1]\!]^B)$ ▷ If $f = g$.
  $[\![e2]\!]^B \leftarrow \mathsf{Comp}([\![W^j_{l,f}[f_j]]\!], [\![W^j_{l,g}[g_j]]\!])$, $[\![e2]\!] \leftarrow \mathsf{B2A}([\![e2]\!]^B)$ ▷ If $\mathsf{LCP}[f] < \mathsf{LCP}[g]$
  $[\![f_{ex}]\!] \leftarrow \mathsf{Choose}([\![W^j_{p,g}[g_j]]\!], [\![W^j_{p,f}[f_j]]\!], [\![e2]\!])$, $[\![g_{ex}]\!] \leftarrow \mathsf{Choose}([\![W^j_{n,g}[g_j]]\!], [\![W^j_{n,f}[f_j]]\!], [\![e2]\!])$
  $[\![l_{ex}]\!] \leftarrow \mathsf{Choose}([\![W^j_{l,g}[g_j]]\!], [\![W^j_{l,f}[f_j]]\!], [\![e2]\!])$
  $[\![f_{j+1}]\!] \leftarrow \mathsf{Choose}([\![f_{ex}]\!], [\![\hat{f}_j]\!], [\![e1]\!])$, $[\![g_{j+1}]\!] \leftarrow \mathsf{Choose}([\![g_{ex}]\!], [\![\hat{g}_j]\!], [\![e1]\!])$
  $f_{j+1} \leftarrow \mathsf{Reconst}([\![f_{j+1}]\!]_0, [\![f_{j+1}]\!]_1)$, $g_{j+1} \leftarrow \mathsf{Reconst}([\![g_{j+1}]\!]_0, [\![g_{j+1}]\!]_1)$ ▷ Update f, g
  $[\![e']\!] \leftarrow \mathsf{B2A}(\mathsf{ADD}([\![e1]\!]^B, [\![1]\!]^B))$
  for $i = 0, \cdots, \ell - 1$ **parallelly do** ▷ Maintain right end of the match.
    $[\![u1[i]]\!] \leftarrow \mathsf{MULT}([\![u[i]]\!], [\![e1]\!])$ ▷ $u1 = u$ and $u2 = (0,\ldots,0)$ if $\hat{f} - \hat{g} - r' = 0$,
    $[\![u2[i]]\!] \leftarrow \mathsf{MULT}([\![u[i]]\!], [\![e']\!])$ $u1 = (0,\ldots,0)$ and $u2 = u$ otherwise.
    $[\![u[i]]\!] \leftarrow \mathsf{ADD}([\![u2[(i-1) \bmod \ell]]\!], [\![u1[i]]\!])$ ▷ u is incremented iff. $\hat{f} - \hat{g} - r' \neq 0$.
  end for
  for $c \in \Sigma$ **parallelly do**
    $[\![z_c]\!] \leftarrow \sum_{i\in\{0,\ldots,\ell-1\}} \mathsf{MULT}([\![q_c[i]]\!], [\![u[i]]\!])$ ▷ Select next letter to be searched with.
  end for
  $[\![m']\!] \leftarrow \mathsf{Choose}([\![l_{ex}]\!], \mathsf{ADD}([\![1]\!], [\![m]\!]), [\![e1]\!])$ ▷ Calculate match length
  $[\![p]\!] \leftarrow \mathsf{Choose}(\mathsf{ADD}(\mathsf{ADD}([\![p]\!], [\![m]\!]), [\![-l_{ex}]\!]), [\![p]\!], [\![e1]\!])$ ▷ Update left end position of match
  $[\![e3]\!] \leftarrow \mathsf{B2A}(\mathsf{Comp}([\![m']\!], [\![m_{max}]\!]))$, $[\![m]\!] \leftarrow [\![m']\!]$
  $[\![m_{max}]\!] \leftarrow \mathsf{Choose}([\![m_{max}]\!], [\![m]\!], [\![e3]\!])$, $[\![p_{max}]\!] \leftarrow \mathsf{Choose}([\![p_{max}]\!], [\![p]\!], [\![e3]\!])$ ▷ Update max
end for
P_0 and P_1 send $[\![m_{max}]\!]_0, [\![m_{max}]\!]_1, [\![p_{max}]\!]_0$, and $[\![p_{max}]\!]_1$ to $\mathcal{A}$
(Verification by $\mathcal{A}$) max $\leftarrow \mathsf{Reconst}([\![m_{max}]\!]_0, [\![m_{max}]\!]_1)$ and $p_{max} \leftarrow \mathsf{Reconst}([\![p_{max}]\!]_0, [\![p_{max}]\!]_1)$.
Output: $\mathcal{A}$ outputs m_{max} and p_{max} to report LMEM.

on a realistic environment, we theoretically estimated delays caused by network bandwidth and latency. We assume three environments: LAN (0.2 ms/10 Gbps), WAN_1 (10 ms/100 Mbps), and WAN_2 (50 ms/10 Mbps). During the run of Search phase, we stored all the data that were transferred from P_0 to P_1 in a file and measured the file size as an actual communication size. Note that the communication is symmetric and data transfer size from P_0 to P_1 is equal to that from P_1 to P_0. Based on the data transfer size D byte, we estimate the communication delay by $D/k + 1000 \times eT$, where k is bandwidth, e is latency and T is a round of communication. All the protocols were run with a single thread on the same machine equipped with Intel Xeon 2.2 GHz CPU and 256 GB memory. We also tested the

Table 3 Offline time (Time), offline size (Size), DB preparation time (Time), DB preparation size (Size), Search time on a local machine (Time), Search communication size (Size), estimated Search time for three environments: LAN (0.2 ms/10 Gbps), WAN_1 (10 ms/100 Mbps), and WAN_2 (50 ms/10 Mbps), for $N = 10^4$ (only for Baseline (LMEM)), $10^5, 10^6, 10^7$, and $\ell = 100$. The size unit is MB and the time unit is sec except for the cell describing "20 h<".

		Offline		DB preparation		Search		Estimated time on network		
	N	Time	Size	Time	Size	Time	Size	LAN	WAN_1	WAN_2
Secure	10^5	0.166	0.013	123	305	0.141	0.010	0.181	2.162	10.249
LPM	10^6	0.141	0.013	1248	3051	0.113	0.010	0.153	2.134	10.221
(ours)	10^7	0.150	0.013	12628	30517	0.126	0.010	0.167	2.147	10.234
	10^5	-	-	-	-	691	163	691	707	838
[30]	10^6	-	-	-	-	7817	517	7818	7863	8261
	10^7	-	-	-	-	20 h<	-	-	-	-
	10^5	3995	184	0.146	0.095	13	122	13	24	118
Baseline (LPM)	10^6	38767	1841	1.522	0.954	164	1227	165	268	1196
	10^7	20 h<	-	-	-	-	-	-	-	-
Secure	10^5	7.619	1.704	435	1068	4.817	0.999	5.577	42.900	195.654
LMEM	10^6	7.882	1.704	4467	10681	4.926	0.999	5.686	43.009	195.763
(ours)	10^7	8.457	1.704	46384	106811	5.740	0.999	6.501	43.824	196.578
Baseline	10^4	12747	611	0.015	0.010	46	407	46	80	389
(LMEM)	10^5	20 h<	-	-	-	-	-	-	-	-

C++ implementation of [30], which is based on AHE. The algorithm for LPM in [28] for the string with $|\Sigma| \leq 4$ (e.g., genome sequence) is the same as [30]. [30] is implemented as a server-client software, and the client and the server were run with individual single threads on the same machine. Therefore, the results of [30] do not include delays caused by bandwidth limitation and latency, so we also estimated delays based on the data transfer size and round of communication in the same manner. Each run of the program was terminated if the total runtime of all phases exceeded 20 hours.

Comparison to Baseline Protocols. Table 3 shows the offline time and size, DB preparation time and size, and Search time and communication size for $N = 10^5, 10^6, 10^7$, and $\ell = 100$. It also shows the result of Baseline (LMEM) for $N = 10^4$, as the runs for $N > 10^4$ did not finish within 20 hours. The Search times and communication sizes of Secure LPM and Secure LMEM are several orders of magnitudes faster and smaller than those of Baseline (LPM) and Baseline (LMEM). Since the round and communication complexities of the proposed protocols do not depend on N, their estimated Search time remains small even on WAN environments. The left panel of Figure 5 shows the estimated Search time on WAN_1 for $N = 10^3, 10^4, \ldots, 10^7$ and $\ell = 100$. The times of Secure LPM and Secure LMEM do not increase, while those of the baseline protocols increase linearly to N. The right panel of Figure 5 shows the estimated Search time on WAN_1 for $\ell = 10, 25, \ldots, 100$ for $N = 10^6$. We can not show the results of Baseline (LMEM) because none of its runs were finished within the time limit. As shown in the graph, the time of Secure LPM increases linearly to ℓ and that of Baseline (LPM) increases proportionally to ℓ^2, which are in good agreement with the theoretical complexities in Table 2. According to the graph, the time of Secure LMEM also increases linearly to ℓ though its time and communication complexities are $O(\ell^2)$. This is because the CPU times are much smaller than the delays caused by network latency that are influenced by the round complexity $O(\ell)$.

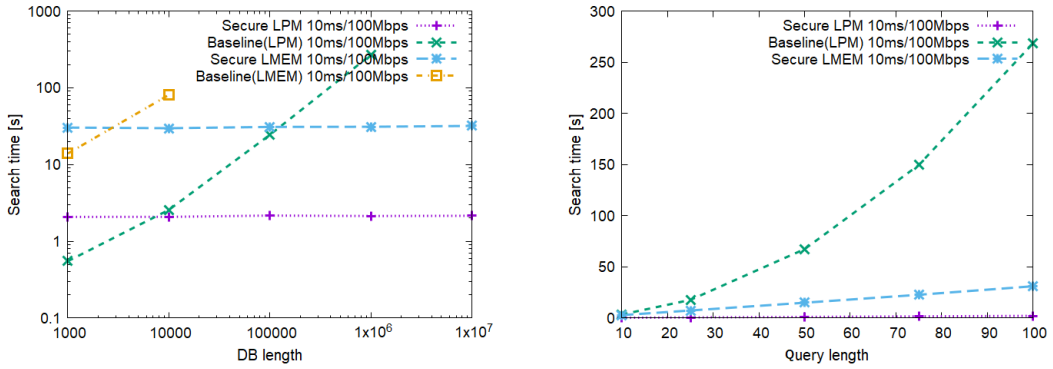


Figure 5 Estimated time (actual search time on a local machine + estimated data transfer time) for various N and ℓ.

We have preliminary results for testing Secure LPM and Baseline (LPM) on the actual network (10 ms/100 Mbps). The results were 40 sec for Secure LPM and 1739 sec for Baseline (LPM) when $N = 10^6$. Though both of the preliminary implementations have room for improvement in the performance of data transfer, the results also indicate that our protocol outperforms the baseline protocol and the previous study.

The time and size of Secure LPM and Secure LMEM are several orders of magnitude better than those of the baseline protocols for the offline phase, and vice versa for the DB preparation phase. The total time of the offline and DB preparation phases of our protocols are more than one order magnitude faster than that of baseline protocols. For the total size of the offline and DB preparation phases, Secure LMEM was better than Baseline (LMEM), but Baseline (LPM) was better than Secure LPM though the complexity is better for Secure LPM. This is because the majority of the shares were Boolean in the baseline protocols, while all of the shares were arithmetic in the proposed protocols.

Comparison to [30]. [30] is a two-party MPC based on AHE. Each homomorphic operation is time consuming and has no offline and DB preparation phases. As shown in Table 3, the Search time of Secure LPM is four orders of magnitude faster than [30] for $N = 10^6$. Since time complexity of [30] includes a factor of N, the difference in Search time becomes greater as N becomes large. Moreover, our protocols have a further advantage in communication for a query response when the network environment is poor, as the round complexity of [30] and our protocols are the same while [30] requires $O(\sqrt{N})$ communication size. The entire runtimes including all the phases are still six times faster for $N = 10^5$ and $N = 10^6$. We can compute LMEM by examining [30] for all the positions in a query string, but this approach consumed 3406 sec and 2.6 GByte of communication for $N = 10^4$.

5 Discussion

As clearly shown by the results, Search time of the proposed protocols are significantly efficient. Considering the importance of query response time for real applications, it is realistic to reduce Search time at the cost of DB preparation time. Since the total times for offline and DB preparation phases of the proposed protocols were significantly better than those of the well-designed baseline protocols, we consider the trade-off between Search and DB preparation times in our approach to be efficient. For further reduction of DB preparation time, parallelizing the share generation is a feasible approach. Regarding the DB preparation

phase, the data transfer between the server and the computing nodes is problematic when the number of queries and the length of the database are large. One potential solution to mitigate the problem is to use an AES-based random number generation that is similar to the technique used in [1]. To explain it briefly, when the server needs to distribute a share of x, (1) the server and P_0 generate the same randomness r using a pre-shared key and a pseudorandom function, and (2) the server computes $x - r$ and sends it to P_1. Although P_0's computation cost increases, we can remove the data transfer from the server to P_0. In our protocols, the generation of shares in the DB preparation phase cannot be outsourced because they are dependent on the database. Designing an efficient algorithm to outsource the share generation is an important open question.

References

1 Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proc. of CCS 2016*, pages 805–817, 2016. doi:10.1145/2976749.2978331.

2 Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. *PoPETs*, 2018(4):104–124, 2018. doi:10.1515/popets-2018-0034.

3 Md Momin Al Aziz, Md. Nazmus Sadat, Dima Alhadidi, Shuang Wang, Xiaoqian Jiang, Cheryl L. Brown, and Noman Mohammed. Privacy-preserving techniques of genomic data - a survey. *Briefings Bioinform.*, 20(3):887–895, 2019.

4 Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *Proc. of CCS 2011*, pages 691–702, 2011. doi:10.1145/2046707.2046785.

5 Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Proc. of CRYPTO 1991*, pages 420–432, 1991. doi:10.1007/3-540-46766-1_34.

6 Yangyi Chen, Bo Peng, XiaoFeng Wang, and Haixu Tang. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *Proc. of NDSS 2012*, 2012. URL: https://www.ndss-symposium.org/ndss2012/large-scale-privacy-preserving-mapping-human-genomic-sequences-hybrid-clouds.

7 Ke Cheng, Yantian Hou, and Liangmin Wang. Secure similar sequence query on outsourced genomic data. In *Proc. of AsiaCCS 2018*, pages 237–251, 2018. doi:10.1145/3196494.3196535.

8 Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. Homomorphic computation of edit distance. In *Proc. of FC 2015*, pages 194–212, 2015. doi:10.1007/978-3-662-48051-9_15.

9 Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.

10 Yaniv Erlich and Arvind Narayanan. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics*, 15(6):409–421, 2014.

11 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proc. of FOCS 2000*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

12 Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. An(other) entropy-bounded compressed suffix tree. In *Proc. of CPM 2008*, pages 152–165, 2008. doi:10.1007/978-3-540-69068-9_16.

13 Marc Fiume, Miroslav Cupak, Stephen Keenan, Jordi Rambla, Sabela de la Torre, Stephanie OM Dyke, Anthony J Brookes, Knox Carey, David Lloyd, Peter Goodhand, et al. Federated discovery and sharing of genomic data using beacons. *Nature biotechnology*, 37(3):220–224, 2019.

14 Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. doi:10.1017/CBO9780511721656.

15 Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proc. of USENIX 2011*, 2011. URL: http://static.usenix.org/events/sec11/tech/full_papers/Huang.pdf.

16 Y. Ishimaki, H. Imabayashi, K. Shimizu, and H. Yamana. Privacy-preserving string search for genome sequences with fhe bootstrapping optimization. In *Proc. of IEEE Big Data 2016*, pages 3989–3991, 2016.

17 Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *Proc. of IEEE S&P 2000*, pages 216–230, 2008. doi:10.1109/SP.2008.34.

18 Md Safiur Rahman Mahdi, Md Momin Al Aziz, Noman Mohammed, and Xiaoqian Jiang. Privacy-preserving string search on encrypted genomic data using a generalized suffix tree. *Informatics in Medicine Unlocked*, 23:100525, 2021.

19 Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2pc for mobile phones. *PoPETs*, 2016(2):82–99, 2016. doi:10.1515/popets-2016-0006.

20 Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Proc. of IEEE S&P 2017*, pages 19–38, 2017. doi:10.1109/SP.2017.12.

21 Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 48(1):1–44, 2015.

22 Koji Nuida, Satsuya Ohata, Shigeo Mitsunari, and Nuttapong Attrapadung. Arbitrary univariate function evaluation and re-encryption protocols over lifted-elgamal type ciphertexts. *IACR Cryptology ePrint Archive*, 2019:1233, 2019. URL: https://eprint.iacr.org/2019/1233.

23 Satsuya Ohata and Koji Nuida. Communication-efficient (client-aided) secure two-party protocols and its application. In *proc. of FC 2020*, pages 369–385, 2020.

24 Anthony A Philippakis, Danielle R Azzariti, Sergi Beltran, Anthony J Brookes, Catherine A Brownstein, Michael Brudno, Han G Brunner, Orion J Buske, Knox Carey, Cassie Doll, et al. The matchmaker exchange: a platform for rare disease gene discovery. *Human mutation*, 36(10):915–921, 2015.

25 Victoria Popic and Serafim Batzoglou. A hybrid cloud read aligner based on minhash and kmer voting that preserves privacy. *Nature communications*, 8(1):1–7, 2017.

26 Thomas Schneider and Oleksandr Tkachenko. EPISODE: efficient privacy-preserving similar sequence queries on outsourced genomic databases. In *Proc. of AsiaCCS 2019*, pages 315–327, 2019. doi:10.1145/3321705.3329800.

27 Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

28 Kana Shimizu, Koji Nuida, and Gunnar Rätsch. Efficient privacy-preserving string search and an application in genomics. *Bioinformatics*, 32(11):1652–1661, 2016. doi:10.1093/bioinformatics/btw050.

29 Katerina Sotiraki, Esha Ghosh, and Hao Chen. Privately computing set-maximal matches in genomic data. *BMC Medical Genomics*, 13(7):1–8, 2020.

30 H. Sudo, M. Jimbo, K. Nuida, and K. Shimizu. Secure wavelet matrix: Alphabet-friendly privacy-preserving string search for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(5):1675–1684, 2019.

31 Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proc. of CCS 2015*, pages 492–503, 2015. doi:10.1145/2810103.2813725.

32 Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In Ari Juels and Bryan Parno, editors, *Proc. of CCSW'13*, pages 65–76, 2013. doi:10.1145/2517488.2517497.

33 R. Zhu and Y. Huang. Efficient and precise secure generalized edit distance and beyond. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2020.

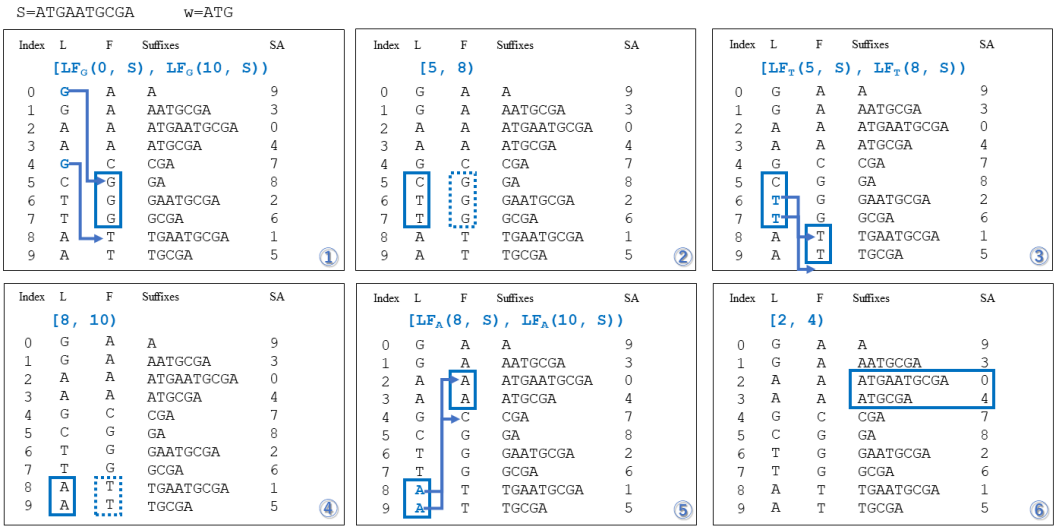


Figure A1 An example of search by FM-Index.

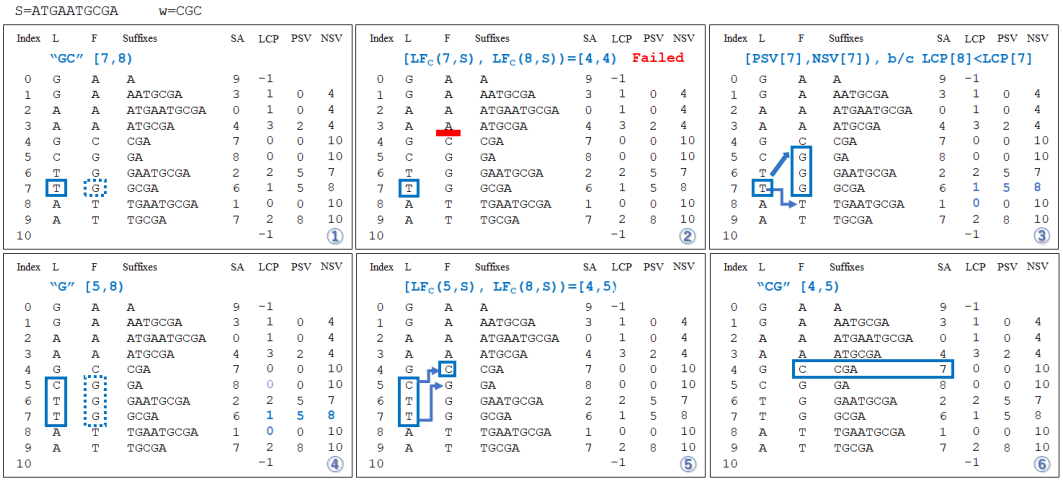


Figure A2 An example of search by FM-ndex, LCP array, PSV and NSV.

A Examples of a Search with FM-Index and Auxiliary Data Structures

Let us show examples of a search with FM-Index, LCP array, PSV and NSV. In addition to the data structures defined in Section 2.2, we also define a string F such that $F[i] = S[SA[i]]$. For the case of S =ATGAATGCGA, the indices become $SA = (9, 3, 0, 4, 7, 8, 2, 6, 1, 5)$, $L =$ GGAAGCTTAA, and $F =$ AAAACGGGTT. Figure A1 illustrates the example of a backward search to find the longest suffix of the query (ATG) that matches the database, and Figure A2 illustrates the search for MEMs with the query (CGC) by using LCP array, PSV, and NSV. As shown in the upper center panel of Figure A2, the search failed when the backward search with 'C' after finding the interval $[7, 8)$ that corresponds to GC. Since $\mathsf{LCP}[8] \leq \mathsf{LCP}[7]$, the parent lcp-interval becomes $[\mathsf{PSV}[7] = 5, \mathsf{NSV}[7] = 8)$, which corresponds to 'G'. The match CG is then searched with the backward search with 'C' from the parent lcp-interval.

B Semi-Honest Security

Here, we recall the simulation-based security notion in the presence of semi-honest adversaries (for two-party computation), as in [14].

▶ **Definition 5.** *Let* $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$ *be a probabilistic* 2*-ary functionality and* $f_i(\vec{x})$ *denote the* i*-th element of* $f(\vec{x})$ *for* $\vec{x} = (x_0, x_1) \in (\{0,1\}^*)^2$ *and* $i \in \{0,1\}$*;* $f(\vec{x}) = (f_0(\vec{x}), f_1(\vec{x}))$*. Let* Π *be a* 2*-party protocol to compute the functionality* f*. The view of party* P_i *for* $i \in \{0,1\}$ *during an execution of* Π *on input* $\vec{x} = (x_0, x_1) \in (\{0,1\}^*)$ *where* $|x_0| = |x_1|$*, denoted by* $\mathrm{VIEW}_i^{\Pi}(\vec{x})$*, consists of* $(x_i, r_i, m_{i,1}, \ldots, m_{i,t})$*, where* x_i *represents* P_i*'s input,* r_i *represents its internal random coins, and* $m_{i,j}$ *represents the* j*-th message that* P_i *has received. The output of all parties after an execution of* Π *on input* $\vec{x}$ *is denoted as* $\mathrm{OUTPUT}^{\Pi}(\vec{x})$*. Then, for each party* P_i*, we say that* Π privately computes f *in the presence of semi-honest corrupted party* P_i *if there exists a probabilistic polynomial-time algorithm* $\mathcal{S}$ *such that*

$$\{(\mathcal{S}(i, x_i, f_i(\vec{x})), f(\vec{x}))\} \equiv \{(\mathrm{VIEW}_i^{\Pi}(\vec{x}), \mathrm{OUTPUT}^{\Pi}(\vec{x}))\},$$

where the symbol $\equiv$ *means that the two probability distributions are statistically indistinguishable.*

As described in [14], the composition theorem for the semi-honest model holds; that is, any protocol is privately computed as long as its subroutines are privately computed.

C Our Secure Baseline LPM and LMEM

In this section, we show our secure baseline LCP and LMEM based on secret sharing. We explain how to construct LCP, since we can obtain LMEM by (parallelly) executing LCP for all positions in the query. Note that $\vec{x} = (x_1, x_2, \cdots)$, $\vec{x_i}$ denotes an i-th element of $\vec{x}$, $[\![\vec{t}]\!] = ([\![\vec{t}]\!]_0, [\![\vec{t}]\!]_1)$, and $(|\vec{x}|, |\vec{y}|) = (L, N)$. Here, we assume $N > L$. When $[\![\vec{x}]\!] = ([\![x_1]\!], [\![x_2]\!], \cdots, [\![x_p]\!])$, $[\![\vec{x}]\!] \gg 1$ means $([\![0]\!], [\![x_1]\!], \cdots, [\![x]\!]_{p-1})$. In our protocol, we use two subprotocols as follows:

- All-AND takes a list $[\![\vec{t}]\!]$ (with p Boolean shares) as input and outputs $[\![t_1 \wedge \cdots \wedge t_p]\!]^B$. We can compute this function with $\lceil p \rceil$ communication rounds (by appropriate parallelization) and $O(NL)$-bit data transfer. The total data transfer size for this All-AND is $\{-\frac{1}{3}L^3 + \frac{1}{2}(N+1)L^2 - \frac{1}{6}(3N+1)L\}$-bit since we execute All-AND at most $(N-1)$ times.
- All-OR takes a list $[\![\vec{u}]\!]$ (with p Boolean shares) as input and outputs $[\![u_1 \vee \cdots \vee u_p]\!]^B$. We can compute this function with $\lceil p \rceil$ communication rounds (by appropriate parallelization) and $O(N)$-bit data transfer. The total data transfer size for this All-OR is $\{-\frac{1}{2}L^2 + \frac{1}{2}(2N-1)L\}$-bit since we execute All-OR for L times.

Our protocol is as in Protocol A1.

In the following, we explain the details of our baseline longest common prefix search protocol using an example that strings $\vec{x}$ = "TGA" and $\vec{y}$ = "ATTGC". In this example, $w = 2$ since there exists "TG" in $\vec{y}$, but "TGA" does not. First, we check whether $w = 1$ or not. To achieve this functionality, we check whether the first character of $\vec{x}$ (i.e., "T" in the example) exists in $\vec{y}$ using Equality for N times. If there exists at least one True in this calculation result, it means $w = 1$ (or larger), and we can achieve this functionality using All-OR. Then, we check whether $w = 2$ or not; that is, we check whether there exists "TG" in the subsequence of $\vec{y}$. Here, we only need to consider ("AT", "TT", "TG", "GC") as subsequences in our example. We search for the perfect matching using Equality and

Protocol A1 Baseline Secure LPM.

Functionality: Compute the length of the longest common prefix w
Input: Strings $[\![\vec{x}]\!]$ and $[\![\vec{y}]\!]$, where $(|\vec{x}|, |\vec{y}|) = (L, N)$
Output: $[\![w]\!]$

for $i = 1, \cdots, L$ **do**
  for $j = 1, \cdots, N$ **do**
    $[\![\vec{s}_{i,j}]\!]^B = \mathsf{Equality}([\![\vec{x}_i]\!], [\![\vec{y}_j]\!])$
  end for
end for
for $i = 1, \cdots, L$ **do**
  P_I $(I \in 0, 1)$ locally generates an empty list $[\![\vec{u}]\!]_I$.
  for $j = 1, \cdots, N - i + 1$ **do**
    if $i = 1$ **then**
      P_I locally adds $[\![\vec{s}_{1,j}]\!]_I^B$ to $[\![\vec{u}]\!]_I$.
    else
      P_I locally generates an empty list $[\![\vec{t}]\!]_I$.
      for $k = 1, \cdots, i$ **do**
        P_I locally add $[\![\vec{s}_{k,k+j-1}]\!]_I^B$ to $[\![\vec{t}]\!]_I$.
      end for
      P_I adds $\mathsf{All\text{-}AND}([\![\vec{t}]\!])$ to $[\![\vec{u}]\!]_I$.
    end if
  end for
  $[\![\vec{v}_{L-i+1}]\!]^B = \mathsf{All\text{-}OR}([\![\vec{u}]\!])$
end for
$[\![\vec{v}]\!]^B = [\![\vec{v}]\!]^B \oplus ([\![\vec{v}]\!]^B \gg 1)$
$[\![\vec{v}]\!] = \mathsf{B2A}([\![\vec{v}]\!]^B)$
$[\![w]\!] = \sum_{\ell=1}^{L} [\![\vec{v}_\ell]\!] \cdot \ell$
return $[\![w]\!]$

All-AND in our protocol. The condition $w = 2$ (or larger) holds if there is at least one perfect matching, and we can achieve this functionality using All-OR. We can compute the cases of $w \leq 3$ using almost the same strategy. After that, we extract the number of the longest common prefix. In the above procedure, we can obtain (False, True, True) for $(w = 3, w = 2, w = 1)$, respectively. We can extract the leftmost True using 1-bit right-shift and XOR, which is a common technique for constructing secure protocols. Finally, we can obtain a final output $[\![w]\!]$ using B2A and the inner product (with constant numbers). Note that we can optimize Equality by replacing simple OR. This is because all characters in $\vec{x}$ and $\vec{y}$ are {"A", "T", "G", "C"}, and we can represent them using 2-bit arithmetic sharing. With an appropriate parallelization, we can execute Protocol A1 with $O(\log \lceil L \rceil + \log \lceil N \rceil)$ communication rounds.

D Proof of Theorem 2

Proof. Correctness and security of ss-ROT protocol are proved as follows.

Proof of correctness. We assume the following equation.

$$p_i = (V^{(i)}[p_0] + r^{i-1}) \bmod N \tag{8}$$

In Step 1, for $j = 0$, the protocol computes p_1 by reconstructing $R^0[p_0]$. From the definition of $R^j[i]$ in Eq. 4,

$$p_1 = R^0[p_0] = (V^{(1)}[p_0] + r^0) \bmod N. \tag{9}$$

For $j = k$, the protocol computes p_{k+1} by reconstructing $R^k[p_k]$. From the definition of $R^j[i]$ in Eq. 4 and the assumption of Eq. 8,

$$\begin{aligned} p_{k+1} = R^k[p_k] &= (V[\,(p_k - r^{k-1}) \bmod N\,] + r^k) \bmod N \\ &= (V[\,V^{(k)}[p_0]\,] + r^k) \bmod N \\ &= (V^{(k+1)}[p_0] + r^k) \bmod N. \end{aligned} \tag{10}$$

Eq. 8 holds for $i = 1$ by Eq. 9. It also holds for $i = k + 1$ under the assumption that Eq. 8 holds for $i = k$. Therefore by induction, Eq. 8 holds for $i = 1, \ldots, \ell - 1$.

In Step 2, P_0 and P_1 output $[\![R^{\ell-1}[p_{\ell-1}]]\!]$. Since Eq. 8 holds for $i = \ell - 1$,

$$R^{\ell-1}[p_{\ell-1}] = (V[(p_{\ell-1} - r^{\ell-2}) \bmod N]) \bmod N$$

is transformed into $(V^{(\ell)}[p_0]) \bmod N$ by plugging in $p_{\ell-1} = V^{(\ell-1)}[p_0] + r^{\ell-2}$. Therefore the final output of ss-ROT becomes $[\![(V^{(\ell)}[p_0]) \bmod N]\!]$. The above argument completes the proof of correctness of Theorem 2.

Proof of security. Due to space limits, we only show a sketch of the proof. In the DB preparation phase of ss-ROT, $\mathcal{B}$ does not disclose any private values, and P_0 and P_1 receive the shares. In the Search phase, all the messages exchanged between P_0 and P_1 are shares except for the result of Reconst in Step 1. In the j-th step of the loop in Step 1, $p_{j+1} = R^j[p_j] = (V^{(j+1)}[p_0] + r^j) \bmod N$ is reconstructed. Since the reconstructed value is randomized by r^j, no information is leaked. Note that for each vector R^j, all the elements $R^j[0], \ldots, R^j[N-1]$ are randomized by the same value r^j, but only one of them is reconstructed, and different random numbers $r^0, \ldots, r^{\ell-1}$ are used for $R^0, \ldots, R^{\ell-1}$. In Step 2, P_0 and P_1 output a result, and no information other than the result is leaked. ◀

E Proof of Theorem 3

Proof. Correctness and security of Protocol 2 are proved as follows.

Proof of correctness. The lookup table V simply stores all possible outputs of LF. Therefore, backward search (Eq. 1) is equivalent to Eq. 5. For the case of querying w, $V_{w[k-1]}[\cdots V_{w[0]}[p_0]\cdots]$ becomes lower bound f (for $p_0 = 0$) or upper bound g (for $p_0 = N$) of the interval that corresponds to the prefix match of length k. In Line 5 of Protocol 2, $[\![R^k_{\mathrm{A},f}[f_k] \times q_{\mathrm{A}}[k] + R^k_{\mathrm{C},f}[f_k] \times q_{\mathrm{C}}[k] + R^k_{\mathrm{G},f}[f_k] \times q_{\mathrm{G}}[k] + R^k_{\mathrm{T},f}[f_k] \times q_{\mathrm{T}}[k]]\!]$ is computed. Since $q_{w[j]}[j] = 1$ and $q_c[j] = 0$ $(c \neq w[j])$, it is equivalent to $[\![R^k_{w[k],f}[f_k]]\!]$. Line 6 computes $[\![R^j_{w[k],g}[g_k]]\!]$ in the same manner. Each vector $R^j_{c,f}$ in Eq. 6 is generated in the same manner as R^j in Eq. 4. Since Eq. 6 uses the common random values r^j_f and r^{j-1}_f for $R^j_{\mathrm{A},f}$, $R^j_{\mathrm{C},f}$, $R^j_{\mathrm{G},f}$, $R^j_{\mathrm{T},f}$, we can recursively reference V_c $(c \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\})$, which is obvious from the correctness of ss-ROT. Therefore, the recursion by Line 5 and Line 7 can compute $(V_{w[k-1]}[\cdots V_{w[0]}[f_0]\cdots] + r^{k-1}_f) \bmod N'$, and the recursion by Line 6 and Line 8 can also compute $(V_{w[k-1]}[\cdots V_{w[0]}[g_0]\cdots] + r^{k-1}_g) \bmod N'$.

The longest match is found when the interval width becomes 0. Since $f_k = (V_{w[k-1]}[\cdots V_{w[0]}[f_0]\cdots] + r_f^{k-1}) \bmod N'$ and $g_k = (V_{w[k-1]}[\cdots V_{w[0]}[g_0]\cdots] + r_g^{k-1}) \bmod N'$ are randomized, Line 11 computes $f_k - g_k - (r'[k-1] = r_f^{k-1} - r_g^{k-1})$ to obtain the correct interval width. Line 11 also computes the equality of 0 and the interval width for each round. By reconstructing all the results in Lines 15–17, $\mathcal{A}$ knows the round, in which the interval width becomes 0; i.e., he/she knows LPM. The above argument completes the proof of correctness of Theorem 3.

Proof of security. Due to space limitation, we only show a sketch of the proof. For Lines 1–2 of Protocol 2, $\mathcal{A}$ and $\mathcal{B}$ do not disclose any private values, and P_0 and P_1 receive the shares. For Lines 3–13, it is guaranteed by the subprotocols ADD, MULT, and Equality that all the messages exchanged between P_0 and P_1 are shares except for the output of Reconst in Lines 7–8. (See Section 2.1 for details of the subprotocols.) In Lines 7–8, reconstructed values are $R^k_{w[j],f}[f_j]$ and $R^k_{w[j],g}[g_j]$. Since the values are $(V_{w[j]}[f_j]+r_f^j) \bmod N'$ and $(V_{w[j]}[g_j] + r_g^j) \bmod N'$ according to Eq. 6, it is obvious that V is randomized for all rounds $j = 0, \ldots, \ell - 1$, and no information is leaked. For Lines 14–17, only the output of Equality at Line 11 is reconstructed. The reconstructed values are either 1 or 0 according to Equality, and no information other than the result is leaked. ◀

F Proof of theorem 4

Proof. Correctness and security of Protocol 3 are proved as follows.

Proof of correctness. V, R, r' and q are generated by the same algorithm used in Protocol 2. Therefore, Line 6 is equivalent to a backward search, and $e1$ is the result of the equality check of 0 and the width of the obtained interval in Line 7. The lookup tables V_{lcp}, V_{psv}, and V_{nsv} store all the outputs of LCP, PSV and NSV, and W_l, W_p, and W_n are generated based on V_{lcp}, V_{psv}, and V_{nsv}, respectively. Since $W^j_{l,f}$ and $W^j_{l,g}$ are circular permutations of V_{lcp} by the same random values r_f^{j-1} and r_g^{j-1} that are used for generating $R_{c,f}$ and $R_{c,g}$ ($c \in \Sigma$) respectively, Line 8 can compute $\mathsf{LCP}[g_j] \leq \mathsf{LCP}[f_j]$ and $e2$ holds the result. By using Choose and $e2$, either $[W^j_{p,f}[f_j], W^j_{n,f}[f_j])$ or $[W^j_{p,g}[g_j], W^j_{n,g}[g_j])$ is selected. $W^j_{p,f}$ and $W^j_{p,g}$ are permutated by r_f^{j-1} and r_g^{j-1}, but are randomized by the identical random value r_f^j. Similarly, $W^j_{n,f}$ and $W^j_{n,g}$ are permutated by r_f^{j-1} and r_g^{j-1}, but are randomized by r_g^j. Since $W_{p,f}[f_j]$ and $W^j_{n,g}[g_j]$ are generated in the same manner as $R_{c,f}$ and $R_{c,g}$, it is obvious that the reference by them is correct. The reference by $W^j_{n,f}[f_j]$ is transformed into

$$\begin{aligned} X_g^{j+1}[W^j_{n,f}[f_j]] &= V_x[W^j_{n,f}[f_j] - r_g^j] + r_g^{j+1} \\ &= V_x[V_{nsv}[f_j - r_f^{j-1}] + r_g^j - r_g^j] + r_g^{j+1} \\ &= V_x[\, V_{nsv}[f_j - r_f^{j-1}]\,] + r_g^{j+1} \end{aligned} \tag{11}$$

and the reference by $W^j_{p,f}[g_j]$ is transformed into

$$\begin{aligned} X_f^{j+1}[W^j_{p,g}[g_j]] &= V_x[W^j_{p,g}[g_j] - r_f^j] + r_f^{j+1} \\ &= V_x[V_{psv}[g_j - r_g^{j-1}] + r_f^j - r_f^j] + r_f^{j+1} \\ &= V_x[\, V_{psv}[g_j - r_g^{j-1}]\,] + r_f^{j+1} \end{aligned} \tag{12}$$

where X^{j+1} is any one of R_c^{j+1}, W_p^{j+1} and W_n^{j+1}, and V_x is the corresponding lookup table; i.e., either one of V_c, V_{psv} and V_{nsv}. Note that V_x could be a different table for each $j+1$, but we abuse the same notation for simplicity of notation. Since f_j and g_j are described in the form of $V_x^{(j)}[p_0] + r_f^{j-1}$ and $V_x^{(j)}[p_0'] + r_g^{j-1}$ based on Eq. 8, Eq. 11 and Eq. 12 are transformed into $V_x^{(j+2)}[p_0] + r_g^{j+1}$ and $V_x^{(j+2)}[p_0'] + r_f^{j+1}$, which also satisfy the recursion form of Eq. 8. Thus, the intervals $[W_{p,f}^j[f_j], W_{n,f}^j[f_j])$ and $[W_{p,g}^j[g_j], W_{n,g}^j[g_j])$ are correct intervals and Line 9 is equivalent to computing Eq. 2.

Lines 14–18, u remains the same if $e1 = 0$ and is permutated such that $u[i] = u[(i-1) \bmod \ell]$ otherwise. Therefore Lines 19–21 can choose the letter to be searched with. The match length and the start position are obtained based on $e1$ in Lines 22–23, and the longest value and the corresponding position are selected in Lines 24–25. The shares of the length and start position of LMEM are sent to $\mathcal{A}$, and $\mathcal{A}$ reconstructs them. Then, Protocol 3 outputs them. The above argument completes the proof of correctness of Theorem 4.

Proof of security. Due to space limits, we only show a sketch of the proof. For Lines 1–2 of Protocol 3, $\mathcal{A}$ and $\mathcal{B}$ do not disclose any private values, and P_0 and P_1 receive the shares. For Lines 3–27, it is guaranteed by the subprotocols ADD, MULT, Equality, and Choose that all the messages exchanged between P_0 and P_1 are shares except for the output of Reconst in Line 12. (See Section 2.1 for details of the subprotocols.) In Line 12, the reconstructed values are $f_{i+1} = V_x^{(j+1)}[p_0] + r_f^j$ and $g_{j+1} = V_x^{(j+1)}[p_0] + r_g^j$, according to Eq. 8, Eq. 11, and Eq. 12. Since f_{j+1} and g_{j+1} are randomized by r_f^j and r_g^j, respectively, for all rounds $j = 0, \ldots, 2\ell - 1$, no information is leaked. In Line 28, $\mathcal{A}$ reconstructs only the search result (the length and start position of LMEM). ◀

Making Sense of a Cophylogeny Output: Efficient Listing of Representative Reconciliations

Yishu Wang
Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, F-69622 Villeurbanne, France
Inria Grenoble Rhône-Alpes, Villeurbanne, France

Arnaud Mary
Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, F-69622 Villeurbanne, France
Inria Grenoble Rhône-Alpes, Villeurbanne, France

Marie-France Sagot
Inria Grenoble Rhône-Alpes, Villeurbanne, France
Université de Lyon, Université Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive UMR 5558, F-69622 Villeurbanne, France

Blerina Sinaimeri
Luiss University, Rome, Italy
ERABLE team, Inria Grenoble Rhône-Alpes, Villeurbanne, France

Abstract

Cophylogeny reconciliation is a powerful method for analyzing host-parasite (or host-symbiont) co-evolution. It models co-evolution as an optimization problem where the set of all optimal solutions may represent different biological scenarios which thus need to be analyzed separately. Despite the significant research done in the area, few approaches have addressed the problem of helping the biologist deal with the often huge space of optimal solutions. In this paper, we propose a new approach to tackle this problem. We introduce three different criteria under which two solutions may be considered biologically equivalent, and then we propose polynomial-delay algorithms that enumerate *only* one representative per equivalence class (without listing all the solutions). Our results are of both theoretical and practical importance. Indeed, as shown by the experiments, we are able to significantly reduce the space of optimal solutions while still maintaining important biological information about the whole space.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Mathematics of computing → Graph enumeration; Theory of computation → Backtracking

Keywords and phrases Cophylogeny, Enumeration, Equivalence relation, Dynamic programming

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.3

1 Introduction

Reconstructing the evolutionary history of parasites (or symbionts) and their hosts has many applications such as for example identifying and tracing the origins of emerging infectious diseases [8, 16, 26]. These studies have become increasingly more important with the large amount of publicly available sequence data. A powerful framework for modeling host-parasite co-evolution is provided by *cophylogeny* models which derive evolutionary scenarios for both hosts and parasites (usually evolutionary trees are computed from DNA sequence data). Co-evolution is usually modeled as a problem of mapping the phylogenetic tree of the parasites to the one of the hosts (see e.g. [6, 19, 7, 32]). Such mapping, called a reconciliation, allows the identification of some biologically motivated events: (a) cospeciation, when the parasite diverges in correspondence to the divergence of a host species; (b) duplication, when the parasite diverges but not the host; (c) host-switching, when a parasite switches from one host

species to another independent of any host divergence; and (d) loss, which can describe for instance speciation of the host species independently of the parasite, which then follows just one of the new host species. Finding the "best" reconciliation is modeled as an optimization problem by assigning a cost to each of the different types of events and then seeking the reconciliations that minimize the total cost (computed in an additive way). In practice, there may often be many optimal solutions which, although having the same total cost, can be quite different among them and correspond to different biological scenarios. Most of the software proposed in the literature therefore do not rely only on one optimal solution but enumerate *all* of them (e.g. [20, 32, 7, 15, 28]). A crucial issue is that often the number of optimal solutions is unrealistically large (exponential in the size of the trees) [7, 14, 13, 18, 11], making it practically impossible to analyze each one of them separately.

To tackle this problem, we observe that although many of the solutions can be indeed very different, a large number of them are quite similar and can be considered biologically equivalent. We thus first propose various equivalence relations for grouping the reconciliations that may be considered biologically equivalent, then we provide algorithms which efficiently enumerate *only* the equivalence classes or one representative reconciliation per class.

State of the art

Many methods have been proposed in the literature to deal with the large number of optimal reconciliations. Some early approaches propose sampling the space of optimal reconciliations uniformly at random [2, 29]. However, as the optimal reconciliation space can be both large and heterogeneous [12], this does not guarantee that important information is not lost.

Other methods try to understand the structure of the space of solutions by computing some global properties such as the frequency of the events across the space [29], the diameter of the space [12], the pairwise distance among the optimal reconciliations [27]. In a similar direction, other methods propose a single reconciliation (e.g. a "median" reconciliation) to represent the whole space of optimal ones [22, 14, 11]. However, the results presented in [13, 11, 12, 27] show that the space can be very diverse and making inferences from a single reconciliation might lead to conclusions that can be contradicted by other optimal reconciliations. The method in [22] has been generalized in [24] in order to find a set of k medoids, or k centers that represent the space. However, these algorithms have a running time of $O(n^{k+3} \log k)$ (where k is the number of clusters and n is the size of the trees) and are thus not applicable in practice. Finally, in [18, 28] the solutions are clustered using a similarity distance among the reconciliations. However, in some cases the results of the clustering can be hard to interpret (see Section 3.3).

Our contribution

In this paper, we propose an approach that is entirely different from the ones discussed in the state of the art section. We first formally define under what conditions two solutions can be considered biologically equivalent. To this end, we introduce three different relations of equivalence. We then propose an algorithm that efficiently enumerates the set of "equivalence classes" or that enumerates one representative per class *without* having to first generate all of them. The algorithms that we present are *polynomial-delay*, meaning that the time between the output of any solution and the next one is bounded by a polynomial function of the input size. Our results are of both practical and theoretical importance. Indeed, the problem of enumerating equivalence classes, and particularly the generation of representative solutions is a challenge in the context of enumeration algorithm. It has been identified as a need in different areas, such as genome rearrangements [4], artificial intelligence [1], pattern matching [3, 21], or the study of RNA shapes [9].

The method has already been implemented in a software presented as an application note in [33]. The algorithms, the proofs, and the experiments are presented here for the first time. It is worth mentioning that the theoretical results in this paper have inspired the introduction of a general framework to enumerate equivalence classes for a whole class of problems which can be addressed by dynamic programming algorithms [34].

2 Model description and definitions of the equivalence relations

2.1 Definitions

In this section, we formally present the phylogenetic tree reconciliation problem that was originally introduced by Goodman et al. in 1979 [10]. We start by providing some definitions that will be used in the paper.

For a directed graph G, we denote by $V(G)$ and $A(G)$ respectively the set of nodes and the set of arcs of G. The out-neighbors of a node v are called its children. We consider ordered rooted trees in which arcs are directed away from the root. For a tree T, we denote by $L(T)$ the set of leaf nodes, i.e. those nodes without children, and denote by $r(T)$ the root of T; the non-leaf nodes are called the internal nodes of T. A full rooted binary tree is a rooted tree in which every internal node has two children.

We denote by $p(w)$ the parent of a node w. The children of a node w are denoted by a couple (i.e. an ordered pair) $\mathsf{ch}(w)$. If there exists a directed path from a node v to a node w, the node w is called a *descendant* of v, and v is called an *ancestor* of w; if moreover $v \neq w$, we say that w is a *proper descendant* of v, and that v is a *proper ancestor* of w. If neither w is an ancestor of v nor w is an ancestor of v, we say that the two nodes are *incomparable*, and denote this as $v \not\sim w$. We denote by $\mathsf{LCA}(v, w)$ the lowest common ancestor of two nodes v and w. The subtree of T rooted at a node v containing all descendants of v is denoted by $T|_v$. Finally, we denote by $d_T(v, w)$ the distance, i.e. the number of arcs on a directed path, between two comparable nodes v and w in T.

We define next the Phylogenetic tree reconciliation problem (shortly, the Reconciliation problem). Let H and P be respectively the rooted phylogenetic trees of the host and parasite species, both binary and full. Let σ be a function from $L(P)$ to $L(H)$, representing the parasite/host associations between extant species. A reconciliation is a function ϕ that assigns, for each parasite node $p \in V(P)$, a host node $\phi(p) \in V(H)$, and satisfies the conditions stated in Definition 1. A reconciliation must induce an event function E_ϕ on $V(P)$ which associates each parasite node p to an event $E_\phi(p)$. The set of events is denoted by $\mathcal{E} := \{\mathbb{C}, \mathbb{D}, \mathbb{S}, \mathbb{T}\}$; the leaf parasite node has a special event $\mathbb{T}$; for internal parasite nodes, the event $E_\phi(p)$ is one among three options: *cospeciation* $\mathbb{C}$, *duplication* $\mathbb{D}$, and *host-switch* $\mathbb{S}$. The event for an internal node p will depend on the hosts that are assigned by ϕ to p and to the two children p_1 and p_2 of p. In Definition 1, this dependency is expressed by $E_\phi(p) := E(\phi(p), \phi(p_1), \phi(p_2))$.

▶ **Definition 1** (Reconciliation, Event of a node). *Given two phylogenetic trees H and P, and a function $\sigma : L(P) \to L(H)$, a reconciliation of (H, P, σ) is a function $\phi : V(P) \to V(H)$ satisfying the following:*

1. *For every leaf node $p \in L(P)$, $\phi(p)$ is equal to $\sigma(p)$, and $E_\phi(p) = \mathbb{T}$.*
2. *For every internal node $p \in V(P) \setminus L(P)$ with children (p_1, p_2), exactly one of the following applies:*
   a. *$E(\phi(p), \phi(p_1), \phi(p_2)) = \mathbb{S}$, that is, either $\phi(p_1) \not\sim \phi(p)$ and $\phi(p_2)$ is a descendant of $\phi(p)$, or $\phi(p_2) \not\sim \phi(p)$ and $\phi(p_1)$ is a descendant of $\phi(p)$,*

b. $E(\phi(p), \phi(p_1), \phi(p_2)) = \mathbb{C}$, *that is,* $\mathsf{LCA}(\phi(p_1), \phi(p_2)) = \phi(p)$, *and* $\phi(p_1) \not\sim \phi(p_2)$,
c. $E(\phi(p), \phi(p_1), \phi(p_2)) = \mathbb{D}$, *that is,* $\phi(p_1)$ *and* $\phi(p_2)$ *are descendants of* $\phi(p)$, *and the previous two cases do not apply.*

In a reconciliation, an internal parasite node can be additionally associated to a number of *loss events*. The loss event is denoted by $\mathbb{L}$. A loss can only occur in conjunction with another event ($\mathbb{S}$, $\mathbb{C}$, or $\mathbb{D}$), and the definition of the number of losses splits into several cases according to the accompanying event. We give in Definition 2 the number of loss events associated to an internal node p, called the *loss contribution* $\xi_\phi(p)$. Since the loss contribution is also determined by the hosts that are assigned to p and to the children of p, we will also write $\xi_\phi(p) := \xi(\phi(p), \phi(p_1), \phi(p_2))$.

▶ **Definition 2** (Loss contribution). *Let* $\phi : V(P) \to V(H)$ *be a reconciliation. Let* p *be an internal node of the parasite tree with children* p_1, p_2. *Its loss contribution* $\xi_\phi(p)$ *is defined by:*

$$\xi_\phi(p) := \begin{cases} d_H(\phi(p), \phi(p_1)) & \text{if } E_\phi(p) = \mathbb{S} \text{ and } \phi(p) \not\sim \phi(p_2), \\ d_H(\phi(p), \phi(p_2)) & \text{if } E_\phi(p) = \mathbb{S} \text{ and } \phi(p) \not\sim \phi(p_1), \\ d_H(\phi(p), \phi(p_1)) + d_H(\phi(p), \phi(p_2)) - 2 & \text{if } E_\phi(p) = \mathbb{C}, \\ d_H(\phi(p), \phi(p_1)) + d_H(\phi(p), \phi(p_2)) & \text{otherwise, } E_\phi(p) = \mathbb{D}. \end{cases}$$

The function E_ϕ partitions the set of internal parasite nodes into three disjoint subsets according to their event; these subsets are denoted by $V^{\mathbb{C}}(P)$, $V^{\mathbb{D}}(P)$, $V^{\mathbb{S}}(P)$. The number of occurrences of each of the three events together with the number of losses make up the *event vector* of the reconciliation ϕ:

▶ **Definition 3** (Event vector). *The* event vector *of a reconciliation* ϕ *is a vector of four integers consisting of the total number of each type of events* $\mathbb{C}$, $\mathbb{D}$, $\mathbb{S}$, *and* $\mathbb{L}$, *i.e.*

$$\vec{e}(\phi) := \left(\left|V^{\mathbb{C}}(P)\right|, \left|V^{\mathbb{D}}(P)\right|, \left|V^{\mathbb{S}}(P)\right|, \sum_{p \in V(P) \setminus L(P)} \xi_\phi(p) \right). \tag{1}$$

Given a *cost vector* $\vec{c} := (c(\mathbb{C}), c(\mathbb{D}), c(\mathbb{S}), c(\mathbb{L}))$ assigning a real number to each type of event, the *cost of a reconciliation* ϕ is equal to the dot product between the cost vector and the event vector $\text{cost}(\phi) := \vec{c} \cdot \vec{e}(\phi)$. We are now ready to formulate the optimization version of the Reconciliation problem: Given two phylogenetic trees H and P, a function $\sigma : L(P) \to L(H)$, and a cost vector $\vec{c}$, find a reconciliation ϕ of (H, P, σ) of minimum cost.

In Figure 1, we show two different reconciliations on the same input (H, P, σ). Depending on the cost vector, these reconciliations may or may not be optimal. Notice that if the cost vector is $(0, 0, 0, 0)$, any valid reconciliation will be optimal.

2.2 Dynamic programming algorithm

The Reconciliation problem can be solved by dynamic programming. One of the first methods which took into account all the events described in the previous section was introduced by Michael Charleston in 1998 [5] and has been improved since by different authors. These methods have different ways of dealing with time feasibility which makes the problem hard on undated trees. We will not discuss this further in the present paper, except for mentioning that in the dynamic programming approach presented in this section, the trees are considered undated, and the time feasibility issue can be dealt with in a subsequent step as described in [7]. On the other hand, we show in this section a formulation of the dynamic programming algorithm in terms of a certain directed graph which we will define.

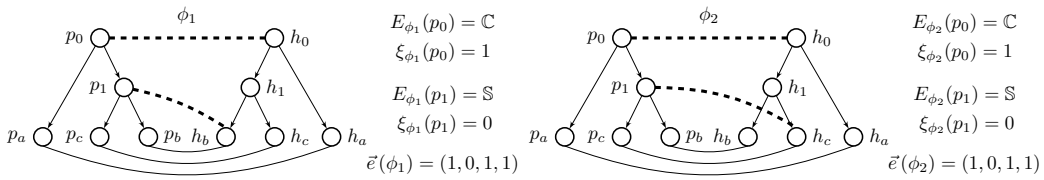


Figure 1 Example of two reconciliations ϕ_1 and ϕ_2 on the same input. For each reconciliation, we draw the parasite tree on the left, the host tree on the right; the solid edges represent the associations for the leaf parasite nodes; the dashed edges represent the associations for the internal parasite nodes.

The graph structure can be seen as a means for efficiently enumerating all optimal solutions of the optimization problem, and more importantly, we will use it later in Section 3 for enumerating equivalence classes of optimal reconciliations.

2.2.1 Recurrence relations

Given an instance $(H, P, \sigma, \vec{c})$, the minimum cost of a reconciliation can be found by dynamic programming. Recall that $\mathcal{E} := \{\mathbb{C}, \mathbb{D}, \mathbb{S}, \mathbb{T}\}$ is the set of possible events for a node. Let $U := V(P) \times V(H) \times \mathcal{E}$. We call a triple $(p, h, e) \in U$ a *cell* of the dynamic programming table. Consider a function $f : U \to \mathbb{R} \cup \{\infty\}$, where the *value* of a cell $f(p, h, e)$ is defined to be the minimum cost of a reconciliation between the subtree $P|_p$ (i.e., the subtree of P rooted at the node p) and the host tree H mapping p to h, such that the event of p is e. Then f can be computed as follows:

1. If p is a leaf,

$$f(p, h, e) = \begin{cases} 0 & \text{if } h = \sigma(p) \text{ and } e = \mathbb{T}, \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

2. Otherwise, p is an internal node with children (p_1, p_2). In this case,

$$f(p, h, e) = \min_{\substack{E(h,h_1,h_2)=e \\ h_1,h_2 \in V(H) \\ e_1,e_2 \in \mathcal{E}}} f(p_1, h_1, e_1) + f(p_2, h_2, e_2) + c(e) + c(\mathbb{L})\, \xi(h, h_1, h_2)\,. \tag{3}$$

The minimum cost of a reconciliation is then given by $\min_{h \in V(H), e \in \mathcal{E}} f(r(P), h, e)$.

2.2.2 ad-AND/OR graphs and solution subtrees

In order to find one optimal reconciliation or to efficiently enumerate all optimal reconciliations, a directed graph can be constructed from the recurrence relations Equations (2) and (3): it is a compact representation of all series of computations performed by dynamic programming which result in the optimal cost value. To do this, we rely on a well-known structure in Computer Science, that is the *AND/OR graph* [23]. More specifically, we consider a particular flavor of AND/OR graphs that we call *acyclic decomposable AND-OR graphs*. This structure is known for having an intimate relationship with dynamic programming on a tree.

▶ **Definition 4** (ad-AND/OR graph). *A directed graph G is an* acyclic decomposable AND/OR graph *(an ad-AND/OR graph) if it satisfies the following:*

- *G is a DAG.*
- *G is bipartite: its node set $V(G)$ can be partitioned into $(\mathcal{A}, \mathcal{O})$ so that all arcs of G are between these two sets. Nodes in $\mathcal{A}$ are called* AND nodes*; nodes in $\mathcal{O}$ are called* OR$^+$ nodes.

- *Every AND node has in-degree at least one and out-degree at least one. The set of nodes with out-degree zero is then a subset of $\mathcal{O}$ and is called the set of* goal nodes*; the remaining OR^+ nodes are simply the* OR nodes*. The subset of OR nodes of in-degree zero is the set of* start nodes*.*
- *G is decomposable: for any AND node, the sets of nodes that are reachable from each one of its child nodes are pairwise disjoint.*

▶ **Definition 5** (Solution subtree)**.** *A* solution subtree *T of an ad-AND/OR graph G is a subgraph of G which: (1) contains exactly one start node; (2) for any OR node in T it contains exactly one of its child nodes in G, and for any AND node in T it contains all its children in G.*

The set of solution subtrees of G is denoted by $\mathcal{T}(G)$. It is immediate to see that a solution subtree is indeed a subtree of G: it is a rooted tree, the root of which is a start node. If we would drop the requirement of G being decomposable, the object defined in Definition 5 would not be guaranteed to be a tree.

▶ **Definition 6** (Subgraph starting from a set of nodes)**.** *Let G be an ad-AND/OR graph. Let $\mathcal{O}$ be a set of OR^+ nodes of G. The* subgraph of G starting from $\mathcal{O}$*, denoted by $G/\mathcal{O}$, is the subgraph obtained from G by setting $\mathcal{O}$ as the new set of start nodes (i.e. by removing all nodes that are not reachable from $\mathcal{O}$ through directed paths).*

2.2.3 The reconciliation graph

The reconciliation graph is a concept already present in the literature [29, 7, 17]. Since, depending on the application, slightly different definitions of this structure exist, to avoid ambiguity, we describe how to construct the *reconciliation graph* of a given instance of the Reconciliation problem from the recurrence Equations (2)–(3).

The construction is done in two steps. In the first step, we build a graph in which every node retains an additional attribute, its *value*, and every OR^+ node is uniquely labeled by a dynamic programming cell $(p, h, e) \in U$. In the second step, we *prune* the graph by removing nodes that do not yield optimal values.

1. For each $(p, h, e) \in U$ such that p is a leaf, create a goal node labeled by (p, h, e); its value is equal to 0 if $h = \sigma(p)$ and ∞ otherwise. Then, for each $(p, h, e) \in U$ in the post-order of $V(P)$, let p_1, p_2 be the two children of p,
   i. For each (p_1, h_1, e_1) and each (p_2, h_2, e_2) such that $E(h, h_1, h_2) = e$, create an AND node, connect it to the two OR^+ nodes respectively labeled by (p_1, h_1, e_1) and (p_2, h_2, e_2). Its value is equal to the sum of the values of its two children, plus $c(e) + c(\mathbb{L})\, \xi(h, h_1, h_2)$.
   ii. Create a single OR node, connect it to every AND node created in the previous step. Its label is (p, h, e), and its value is the minimum of the values of its children.
2. For each $(r(P), h, e) \in U$, remove the OR node labeled by that cell unless its value is equal to the optimal cost. For each OR node s, remove the arc to its child AND node s_i if the value of s_i is not equal to the value of s. Finally, remove recursively all AND nodes without incoming arcs.

It can be checked that the reconciliation graph is indeed an ad-AND/OR graph as defined in Definition 4. An OR^+ node labeled by (p, h, e) is a start node if and only if $p = r(P)$, and is a goal node if and only if $p \in L(P)$. It is also immediate to see that each AND node in the reconciliation graph has exactly one in-neighbor and exactly two children. We will consider

the two children as a couple: for an AND node s, if its in-neighbor is labeled by (p, h, e) and its two children s_1 and s_2 are respectively labeled by (p_1, h_1, e_1) and (p_2, h_2, e_2), we will say that s_1 is the first child and s_2 is the second child of s if p_1 and p_2 are respectively the first and second child of p; otherwise, we say that s_1 is the second child and s_2 is the first child. Keeping the correct order of the children, we can extend the notation "ch" to the set of nodes of the reconciliation graph: if s is an AND node, $\mathsf{ch}(s)$ is the couple (ordered pair) of the two child OR$^+$ nodes of s; if s is an OR node, $\mathsf{ch}(s)$ is simply the set of its AND child nodes. For an OR node, we will typically be interested not in its children but in its set of "grandchildren", hence we introduce here a new notation. If s is an OR node, we call the *grandchild couples*, denoted by $\mathsf{gch}(s)$, the union of the children of its child AND nodes (it is a set of couples of OR$^+$ nodes): $\mathsf{gch}(s) := \bigcup_{s_i \in \mathsf{ch}(s)} \mathsf{ch}(s_i)$. Notice that an OR$^+$ node can appear as grandchild of two different nodes, and can also appear in two different grandchild couples of a same node (see Figure 2).

The dynamic programming algorithms for the Reconciliation problem which enable the efficient enumeration of all optimal reconciliations are based on the following observation:

▷ Claim. Let $(H, P, \sigma, \vec{c})$ be a given instance of the Reconciliation problem. The reconciliation graph G, constructed as described in the previous paragraph is an ad-AND/OR graph, and the set $\mathcal{T}(G)$ of solution subtrees of G correspond bijectively to the set of optimal reconciliations.

To see this, consider an OR$^+$ node s labeled by a cell $(p, h, e) \in U$ of the dynamic programming table. For the subgraph $G/\{s\}$ (see Definition 6), the following can be proven by induction: the set of solution subtrees $\mathcal{T}(G/\{s\})$ corresponds bijectively to the set of optimal reconciliations of the dynamic programming subproblem at (p, h, e), i.e. the optimal reconciliations between the subtree $P|p$ and H such that p is mapped to h and the event of p is e. In practice, to convert a solution subtree $T_1 \in \mathcal{T}(G)$ into a reconciliation ϕ, we only need to look at the labels (p, h, e) of the OR$^+$ nodes in T_1 (a reconciliation can simply be viewed as a collection of triples of the form (p, h, e)). We will henceforth use interchangeably the terms *solution subtrees* of the reconciliation graph and *optimal reconciliations* of the problem instance.

The reconciliation graph can be constructed using $O(|V(P)||V(H)|^3)$ time and space complexity [7]. After the construction, the total number of optimal reconciliations can also be computed. It is a well-known folklore result that the set of solution subtrees of an ad-AND/OR graph can be enumerated efficiently: the delay between outputting two consecutive solutions is linear in the size of the solution. Therefore, there is an algorithm with a $O(|V(P)||V(H)|^3)$ time pre-processing step and $O(|V(P)|)$ time delay for enumerating the optimal reconciliations.

Figure 2 shows a reconciliation graph based on the same input (H, P, σ) as in Figure 1 with nine solution subtrees. Among these nine reconciliations, four have event vector $(0, 0, 2, 0)$, two have $(1, 0, 1, 0)$, two have $(1, 0, 1, 1)$ (ϕ_1 and ϕ_2 of Figure 1), and one has $(2, 0, 0, 0)$. The event vector of the reconciliation shown in bold is $(1, 0, 1, 1)$.

2.3 Equivalent optimal reconciliations

In this section, we first introduce four definitions of equivalence between reconciliations and study the relationship between them, then we explain the motivation for defining such equivalence relations and state the problems of enumerating the equivalence classes and counting the size of each class. The algorithmic contribution solving these problems and the experimental results will be presented in the subsequent sections.

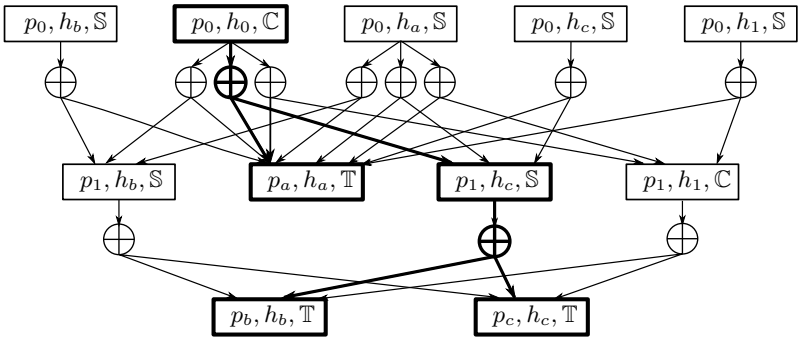


Figure 2 Example of a reconciliation graph for the input (H, P, σ) in Figure 1. Crossed circles are AND nodes. Rectangles are OR^+ nodes. The cells with which the OR^+ nodes are labeled are written inside. One solution subtree is shown in bold.

2.3.1 Definitions

In Definitions 7–9, we give three equivalence relations on the set of optimal reconciliations. One is based on a global property, the event vector, which is already defined in Definition 3. The other two equivalence relations are based on "local properties", i.e. on the event $E_\phi(p)$ and the host $\phi(p)$ that are assigned by ϕ for each parasite node p.

▶ **Definition 7** (V-equivalence). *Two reconciliations ϕ_1 and ϕ_2 are* Vector-equivalent*, or shortly* V-equivalent*, if their event vectors are equal: $\vec{e}(\phi_1) = \vec{e}(\phi_2)$.*

▶ **Definition 8** (E-equivalence). *Two reconciliations ϕ_1 and ϕ_2 are* Event-equivalent*, or shortly* E-equivalent*, if $E_{\phi_1}(p) = E_{\phi_2}(p)$ for all $p \in V(P)$.*

▶ **Definition 9** (CD-equivalence). *Two reconciliations ϕ_1 and ϕ_2 are* Cospeciation-Duplication-equivalent*, or shortly* CD-equivalent*, if $E_{\phi_1}(p) = E_{\phi_2}(p)$ for all $p \in V(P)$ (i.e. they are E-equivalent), and the hosts of non-host-switch parasite nodes are the same: $E_{\phi_1}(p) \neq \mathbb{S} \implies \phi_1(p) = \phi_2(p)$.*

Each one of these equivalence relation splits the set of optimal reconciliations of a given instance into *equivalence classes*, i.e. subsets of pairwise equivalent reconciliations. One *representative* of an equivalence class is simply a reconciliation in the corresponding subset. We will abuse the terminology and call equivalence classes the objects that best represent the common property of the reconciliations in that subset. A reconciliation in a particular equivalence class will then be a reconciliation satisfying that property.

▶ **Definition 10** (Equivalence classes). *In this paper, the term* equivalence class *has the following meanings, depending on the equivalence relation:*

- *For the V-equivalence relation, a* V-equivalence class *is an event vector $\vec{e}$, i.e. a vector of four integers.*
- *For the E-equivalence relation, an* E-equivalence class *is a function $E : V(P) \to \mathcal{E}$ that associates each node of the parasite tree with an event.*
- *For the CD-equivalence relation, a* CD-equivalence class *is a function $E^{CD} : V(P) \to \mathcal{E} \times (V(H) \cup \{?\})$ that associates each node of the parasite tree with an ordered pair (e, h), where either*
  - *e is an event between $\mathbb{T}$, $\mathbb{C}$ and $\mathbb{D}$ and h is a node of the host tree, or*
  - *e is the host-switch event $\mathbb{S}$ and h is a special symbol $?$.*

We can make the following remarks about the relationships between these equivalence relations. CD-equivalent reconciliations are also E-equivalent. Being E-equivalent implies that the first three elements of their event vectors are equal. As we only consider reconciliations having the same minimum cost, if the cost of a loss event $c(\mathbb{L})$ is nonzero, E-equivalent reconciliations necessarily have the same number of losses, hence are also V-equivalent. On the other hand, if $c(\mathbb{L}) = 0$, E-equivalent reconciliations are not necessarily V-equivalent.

In Figure 1, the pair ϕ_1 and ϕ_2 are equivalent under all three equivalence relations. In Figure 2, the nine reconciliations split into four V-equivalence classes (the four event vectors).

2.3.2 Motivation and challenges

The first and foremost motivation of defining equivalence relations is the need of capturing useful biological information from the set of optimal reconciliations, when this set is too large for manual analyses or for exhaustive enumeration. The V-equivalence classes already conveys some information about the co-evolutionary history of the hosts and their parasites. Indeed, a high number of cospeciations may indicate that hosts and parasites evolved together, while a high number of host-switches may indicate that the parasites are able to infect different host species. Under the scope of the E-equivalence relation, we are also interested in which parasites are associated to each type of event (disregarding losses).

The CD-equivalence relation is motivated by the idea that when a host-switch happens, there may be various hosts that can be selected as the parasite's "landing site". In this case, we choose to consider as equivalent those reconciliations for which, while the hosts that receive the switching parasites may differ, all the other parasite-host associations (not corresponding to a host-switch) are the same. These reconciliations are similar and often indistinguishable without additional biological information. Indeed, take the two reconciliations ϕ_1 and ϕ_2 in Figure 1: they are identical except for one switching parasite p_1, which is mapped to h_b by ϕ_1 and to h_c by ϕ_2. Since h_b and h_c are two sibling nodes sharing the same parent in the host tree, without further information, there is no good way to tell apart the two reconciliations ϕ_1 and ϕ_2, hence we consider them as equivalent.

Equipped with our definitions of equivalence classes, we aim at studying the features of the set of optimal reconciliations by enumerating the equivalence classes. Naively, one would iterate through every reconciliation and record their properties, then report the equivalence classes, and, only at the end, report the statistics of the reconciliations in each equivalence class. However, when the number of reconciliations is too large, for example, $> 10^{42}$ (see Section 3.3 and [33]), the naive method is not feasible.

The challenge is then to enumerate directly the equivalence classes of optimal reconciliations without enumerating the latter explicitly. Concretely, the set of optimal reconciliations will be represented implicitly as $\mathcal{T}(G)$, the set of solution subtrees of a reconciliation graph G. Given a reconciliation graph as input, we will tackle the following problems:

- Count the number of equivalence classes.
- Enumerate the equivalence classes.
- Study a particular equivalence class. That is, given an equivalence class,
  - Count the number of reconciliations in that class,
  - Find one representative (i.e. one optimal reconciliation) of that class,
  - Enumerate all reconciliations of that class.

3 Equivalence classes enumeration: algorithms and results

3.1 V-equivalence class enumeration

The enumeration of V-equivalence classes (i.e. all event vectors among the optimal reconciliations) can be achieved by a simple modification of the dynamic programming algorithm.

First, we can notice that the number of different event vectors is bounded by a polynomial. Let $n = |V(H)|$ and $m = |V(P)|$. The first three elements of any event vector necessarily sum up to $\frac{m-1}{2}$, the number of internal parasite nodes, hence there are only $O(m^2)$ possible combinations. The loss contribution $\xi_\phi(p)$ for each parasite node p for any ϕ is at most twice the diameter of the host tree (i.e. twice the maximum distance between two nodes), so the fourth element of any event vector is bounded by $O(nm)$. Therefore, the number of event vectors is bounded by $O(nm^3)$.

We are interested in the following two problems: listing all event vectors, and, given a particular event vector, listing one (or all) optimal reconciliations of that event vector. Both can be done without much difficulty by doing some additional book-keeping in the dynamic programming algorithm, i.e. during the construction of the reconciliation graph. The idea is to remember the set of event vectors in every step, corresponding to the event vectors of the optimal solutions of the current dynamic programming subproblem. Then, for each event vector, one reconciliation (or all reconciliations) of the V-equivalence class can be found by backtracking. Since the technique is quite standard, the details are omitted.

3.2 E-equivalence class enumeration

By Definition 10, an E-equivalence class is a function from the set of nodes $V(P)$ of the parasite tree to the set $\mathcal{E} := \{\mathbb{C}, \mathbb{D}, \mathbb{S}, \mathbb{T}\}$ of events. In this section, we will represent an E-equivalence class as a set T of ordered pairs of the form (p, e) where $p \in V(P)$ and $e \in \mathcal{E}$. In the same manner, a reconciliation ϕ, i.e. a solution subtree in $\mathcal{T}(G)$, can be written as a set of ordered triples of the form (p, h, e). We say that a reconciliation ϕ *belongs to the E-equivalence class* T, and denote it as $\pi(\phi) = T$, if for each $(p, h, e) \in \phi$, there exists a unique couple $(p, e) \in T$. Using this notation, a set of couples of the form (p, e) is an E-equivalence class if and only if there exists $\phi \in \mathcal{T}(G)$ such that $\pi(\phi) = T$; the set of all E-equivalence classes is denoted by $\pi(\mathcal{T}(G))$.

The problem of studying a particular E-equivalence class is easy: given an E-equivalence class T, the reconciliation graph G can be pruned in such a way that its set of solution subtrees corresponds to the reconciliations that belong to the class T (we simply need to remove all OR nodes unless its label (p, h, e) corroborates the given class: $(p, e) \in T$). Counting and enumerating the E-equivalence classes are, however, more challenging problems. We will at present concentrate on the problem of enumerating all E-equivalence classes.

The algorithm is based on the simple idea of traversing the reconciliation graph in a top-down fashion (a similar approach can be used in the algorithm that finds all the solution subtrees). In order to obtain a polynomial time delay algorithm, during the traversal, we can no longer consider the nodes one by one; the sets of nodes that are in the solution subtrees of the same E-equivalence class must be traversed together. To make this clear, it is convenient to define the *color* of the OR^+ nodes; an E-equivalence class will then simply be a set of colors.

▶ **Definition 11** (Color of a node, Color couple)**.**

- *If an* OR^+ *node* s *in the reconciliation graph is labeled by* $(p, h, e) \in U$*, we say that* s *is* colored by *the ordered pair* $(p, e) \in V(P) \times \mathcal{E}$*.*
- *Let* s_1 *and* s_2 *be two* OR^+ *nodes colored respectively by* (p_1, e_1) *and by* (p_2, e_2)*. The* color couple *of the couple of nodes* (s_1, s_2) *is the couple of colors* $((p_1, e_1), (p_2, e_2))$*.*

To enumerate the E-equivalence classes by a top-down recursive traversal of the reconciliation graph, our algorithm should achieve the following goal: given a set $\mathcal{O}$ of OR$^+$ nodes of the same color (p, e), enumerate $\pi(\mathcal{T}(G/\mathcal{O}))$, i.e. all E-equivalence classes of the subgraph $G/\mathcal{O}$. Any such a class will include the color (p, e). If p is not a leaf, the events of the two children of the node p are given by the color couples of the grandchild couples $\mathsf{gch}(\mathcal{O})$ (by extension, gch of a set of nodes is the union of gch of every node in the set). A naive algorithm can be described as follows: for each color couple $((p_1, e_1), (p_2, e_2))$ of $\mathsf{gch}(\mathcal{O})$, first take the union $\mathcal{O}_1$ of the first grandchildren of color (p_1, e_1) and the union $\mathcal{O}_2$ of the second grandchildren of color (p_2, e_2), then call the algorithm on $\mathcal{O}_1$ and independently on $\mathcal{O}_2$, and finally combine the results together, that is, perform a Cartesian product between $\pi(\mathcal{T}(G/\mathcal{O}_1))$ and $\pi(\mathcal{T}(G/\mathcal{O}_2))$.

The pitfall of the naive approach is that not every combination between the E-equivalence classes of the reconciliations of the two child subtrees is valid. Our algorithm, shown in Algorithm 1, can be viewed as an improved version of the naive algorithm in which particular care has been taken to ensure that only valid combinations are outputted. Along with each E-equivalence class T, it also outputs a set $\widetilde{\mathcal{O}}$ which is a subset of the input set $\mathcal{O}$: it is equal the union of the root OR$^+$ nodes of all solution subtrees $\phi \in \mathcal{T}(G/\mathcal{O})$ such that $\pi(\phi) = T$. Notice that in Algorithm 1 we employ both the **return** and the **yield** statements for the output, the difference being that the latter does not halt the algorithm.

Algorithm 1 Enumerating E-equivalence classes.

Input: *a node p of the parasite tree, an event $e \in \mathcal{E}$, a set $\mathcal{O}$ of OR$^+$ nodes*
Require: *The nodes in $\mathcal{O}$ are all colored with (p, e).*
Output: *all E-equivalence classes of $G/\mathcal{O}$, and for each class, a subset of $\mathcal{O}$*
Function `Enumerate`*(p, e, $\mathcal{O}$)*:
  if *p is a leaf* **then** // necessarily $e = \mathbb{T}$ and $\mathcal{O}$ only contains goal nodes
    return $\{(p, e)\}$, $\mathcal{O}$
  end
  /* otherwise, necessarily $e \in \{\mathbb{C}, \mathbb{D}, \mathbb{S}\}$ and $\mathcal{O}$ only contains OR nodes */
  Let (p_1, p_2) be the children of p
  Partition the set of grandchild couples $\mathsf{gch}(\mathcal{O})$ according to their color couples
  for *each subset $\{(s_1^i, s_2^i)\}_{1 \le i \le k}$ of $\mathsf{gch}(\mathcal{O})$ of color couple $((p_1, e_1), (p_2, e_2))$* **do**
    Let $\mathcal{O}_1 := \bigcup_{1 \le i \le k} \{s_1^i\}$ // $\mathcal{O}_1$ is the set of the first grandchildren
    for *each pair of T_1 and $\widetilde{\mathcal{O}_1}$ outputted by* `Enumerate`*(p_1, e_1, $\mathcal{O}_1$)* **do**
      Let $\mathcal{O}_2 := \bigcup_{1 \le i \le k} \left\{ s_2^i \mid \text{it exists } s_1 \in \widetilde{\mathcal{O}_1} \text{ such that } (s_1, s_2^i) \in \mathsf{gch}(\mathcal{O}) \right\}$
      /* $\mathcal{O}_2$ is the set of the second grandchildren compatible with $\widetilde{\mathcal{O}_1}$ */
      for *each pair of T_2 and $\widetilde{\mathcal{O}_2}$ outputted by* `Enumerate`*(p_2, e_2, $\mathcal{O}_2$)* **do**
        Let $\widetilde{\mathcal{O}} := \left\{ s \in \mathcal{O} \mid \exists s_1 \in \widetilde{\mathcal{O}_1}, \exists s_2 \in \widetilde{\mathcal{O}_2}, \text{ s.t. } (s_1, s_2) \in \mathsf{gch}(s) \right\}$
        yield $T_1 \cup T_2 \cup \{(p, e)\}$, $\widetilde{\mathcal{O}}$
      end
    end
  end

Before the proof of correctness, let us recall some important notations. For a subgraph $G/\mathcal{O}$ of the reconciliation graph G, a solution subtree is denoted by $\phi \in \mathcal{T}(G/\mathcal{O})$. The root OR$^+$ node of a solution subtree ϕ is denoted by $r(\phi)$. If the root node $r(\phi)$ is labeled by (p, h, e), the solution subtree ϕ is interpreted as an optimal reconciliation between the

parasite subtree $P|_p$ and the host tree H such that p is mapped to h and the event of p is e (shortly, we say that ϕ is a reconciliation of $P|p$). We will use interchangeably the terms *solution subtree* and *reconciliation*, and we will represent a reconciliation ϕ as a set of triples.

▶ **Lemma 12.** *In Algorithm 1,* `Enumerate`*$(p, e, \mathcal{O})$ outputs all E-equivalence classes in $\pi(\mathcal{T}(G/\mathcal{O}))$ exactly once, and for each outputted pair of T and $\widetilde{\mathcal{O}}$, we have $\widetilde{\mathcal{O}} = \bigcup_\phi \{r(\phi) \mid \pi(\phi) = T, \phi \in \mathcal{T}(G/\mathcal{O})\}$.*

Proof. The proof is by induction on the height h_p of the $P|_p$. We use the fact that the pre-condition in the **Require** statement in Algorithm 1 is true for all recursive calls of `Enumerate` (easy induction). When $h_p = 0$, p is a leaf and $\{(p, \sigma(p), \mathbb{T})\}$ is the only reconciliation in $\mathcal{T}(G/\mathcal{O})$, therefore, $\{(p, e)\}$ is the only E-equivalence class. The outputted set $\mathcal{O}$ contains in this case the unique goal node of G labeled by $(p, \sigma(p), \mathbb{T})$. Now we assume $h_p > 0$.

First direction. Consider a fixed pair of $T := T_1 \cup T_2 \cup \{(p, e)\}$ and $\widetilde{\mathcal{O}}$ outputted at Line 16, and take a node s in $\widetilde{\mathcal{O}}$. We show that there exists a reconciliation $\phi \in \mathcal{T}(G/\mathcal{O})$ such that $s = r(\phi)$ and $\pi(\phi) = T$ (i.e. T is a valid E-equivalence class). By the induction hypotheses, T_1 is an E-equivalence class so there exists a reconciliation ϕ_1 of $P|_{p_1}$ such that $\pi(\phi_1) = T_1$. Let $s_1 := r(\phi_1)$. Take a node $s_2 \in \mathcal{O}_2$ such that $(s_1, s_2) \in \mathsf{gch}(s)$. By the induction hypotheses, there exists a reconciliation ϕ_2 of $P|_{p_2}$ such that $r(\phi_2) = s_2$ and $\pi(\phi_2) = T_2$. Define $\phi := \phi_1 \cup \phi_2 \cup \{(p, h, e)\}$, where (p, h, e) is the label of s. Then ϕ is a valid reconciliation in $\mathcal{T}(G/\mathcal{O})$ (notice that ϕ is a solution subtree of $G/\mathcal{O}$ if and only if $(s_1, s_2) \in \mathsf{gch}(s)$), and satisfies $\pi(\phi) = T$.

Second direction. Consider an E-equivalence class $T \in \pi(\mathcal{T}(G/\mathcal{O}))$, and take a reconciliation $\phi \in \mathcal{T}(G/\mathcal{O})$ such that $\pi(\phi) = T$. We show that T is outputted exactly once at Line 16 together with a set $\widetilde{\mathcal{O}}$ containing the root node of ϕ. Assume that the root node $s := r(\phi)$ is labeled with the triple (p, h, e), then ϕ can be uniquely written as the union $\phi_1 \cup \phi_2 \cup \{(p, h, e)\}$ where ϕ_1 and ϕ_2 are respectively reconciliations of $P|_{p_1}$ and $P|_{p_2}$. Furthermore, T can be uniquely written as the union $T_1 \cup T_2 \cup \{(p, e)\}$ where $T_1 = \pi(\phi_1)$ and $T_2 = \pi(\phi_2)$. Notice that T_1 and T_2 do not depend on the choice of ϕ; for T to be outputted exactly once, it suffices to show that each of T_1 and T_2 is outputted exactly once. For $i = 1, 2$, let $s_i := r(\phi_i)$ and let (p_i, e_i) be the color of s_i. At Line 10, we only need to consider the iteration corresponding to the color couple $((p_1, e_1), (p_2, e_2))$, as no other iteration can output T_1 or T_2 from a recursive call. Since $s_1 \in \mathcal{O}_1$ and $\phi_1 \in \mathcal{T}(G/\mathcal{O}_1)$, by the induction hypotheses, T_1 is outputted exactly once in Line 12 together with a set $\widetilde{\mathcal{O}_1}$ containing s_1. For this pair of T_1 and $\widetilde{\mathcal{O}_1}$, the set $\mathcal{O}_2$ computed at Line 13 contains the node s_2. Hence, by applying again the induction hypotheses to $\phi_2 \in \mathcal{T}(G/\mathcal{O}_2)$, T_2 is outputted exactly once in Line 14 together with $\widetilde{\mathcal{O}_2}$ containing s_2. It remains to check that the set $\mathcal{O}$ outputted together with T does contain the node s. As $s_i \in \widetilde{\mathcal{O}_i}$ for $i = 1, 2$, this is straightforward from the computation of $\mathcal{O}$. ◀

▶ **Theorem 13.** *Using Algorithm 1, the E-equivalence classes of a reconciliation graph can be enumerated in $O(mn^2)$ time delay, where $m = |V(P)|$ and $n = |V(H)|$.*

Proof. To obtain all E-equivalence classes $\pi(\mathcal{T}(G))$, it suffices to first partition the set of start nodes of the reconciliation graph according to their colors, then, for each subset $\mathcal{O}_i$ of start nodes of color (p, e), make one call of `Enumerate`$(p, e, \mathcal{O})$. By Lemma 12, we output every E-equivalence class of $\mathcal{T}(G/\mathcal{O})$ exactly once. Since any E-equivalence class of $\mathcal{T}(G)$ is an E-equivalence class of $\mathcal{T}(G/\mathcal{O}_k)$ for a unique k, we output every E-equivalence class of $\mathcal{T}(G)$ exactly once.

For the complexity, consider the recursion tree formed by the recursive calls of `Enumerate`. Notice that each node p of the parasite tree corresponds to exactly one recursive call, the size of the recursion tree is thus $O(m)$. In each recursive call, the partitioning of $\mathsf{gch}(\mathcal{O})$ and the computation of the sets $\mathcal{O}_1$, $\mathcal{O}_2$, and $\widetilde{\mathcal{O}}$ can all be done in time linear in the size of $\mathsf{gch}(\mathcal{O})$, which is $O(n^2)$. Therefore, $O(mn^2)$ time is needed in the worst case between outputting two E-equivalence classes. ◀

CD-equivalence class enumeration

For the CD-equivalence relation, the problems of enumerating the equivalence classes and studying one particular equivalence class can be solved using the exact same method as for the E-equivalence relation. One only needs to adapt the Definition 11 of the color of an OR^+ node. Instead of the couple (p, e), the color of an OR^+ node labeled by $(p, h, e) \in U$ is now a triple: the triple (p, h, e) for $e \neq \mathbb{S}$, or, when $e = \mathbb{S}$, the triple $(p, ?, \mathbb{S})$ (see Definition 10).

3.3 Experimental results

To evaluate the usefulness of the equivalence classes in practice, we obtained 20 real datasets from the literature. The choice of the datasets was motivated by the goal of covering many different situations (such as different sizes of the trees), different contexts (such as the genes/species one that has been shown to be very closely related to the hosts/parasites context, see for instance [25, 35]), different topologies, etc. We also chose five cost vectors $\vec{c} := (c(\mathbb{C}), c(\mathbb{D}), c(\mathbb{S}), c(\mathbb{L}))$ from the literature, namely $(-1, 1, 1, 1)$ (maximizing the cospeciation), $(0, 1, 1, 1)$ (minimizing the events that lead to incongruencies between the tree topologies), $(0, 1, 2, 1)$, $(0, 2, 3, 1)$ (host-switches are more penalized), and $(0, 1, 1, 0)$ which is a vector chosen only for theoretical purposes and does not penalize cospeciations and losses.

The goal of the first set of experiments is to check that when the number of all optimal reconciliations is large, the number of equivalence classes is significantly smaller. To this purpose, we ran the algorithm on all the datasets with all the five cost vectors, and computed the number of optimal solutions and the number of equivalence classes. For each instance (i.e. dataset and cost vector) having at least 50 optimal reconciliations, we computed for each equivalence relation a value that we called *Reduction* and which is equal to the number of equivalence classes over the number of optimal reconciliations. In Figure 3, each x coordinate corresponds to an instance; for each instance we plotted three points that correspond to the Reduction values for the three equivalence relations. One can observe that the Reduction values of the V- and the E-equivalence relations (blue circles and red triangles) are almost all below the value of 0.1. In other words, for these two definitions of equivalence, one can strongly hope for at least a ten-fold decrease, and in some cases for a thousand-fold decrease in the number of reconciliations that need to be analyzed. As expected, the V- and the E-equivalence relations are the ones that usually lead to a small number of equivalence classes, while the CD-equivalence relation may lead to a larger number of classes, sometimes close to the optimal reconciliations (Reduction close to 1).

We show now that the equivalence classes not only allow us to reduce the number of reconciliations to consider, but also provide useful information about the set of optimal reconciliations. In Table 1, we present the detailed results obtained for the dataset of *Wolbachia* and their arthropod hosts [31, 30] and the five cost vectors. All the cost vectors lead to a number of optimal reconciliations that is at least 10^{42}, a number too large for any exhaustive enumeration method. However, in all cases there are only a small number of optimal event vectors (except for the least biologically meaningful cost vector $(0, 1, 1, 0)$). For the

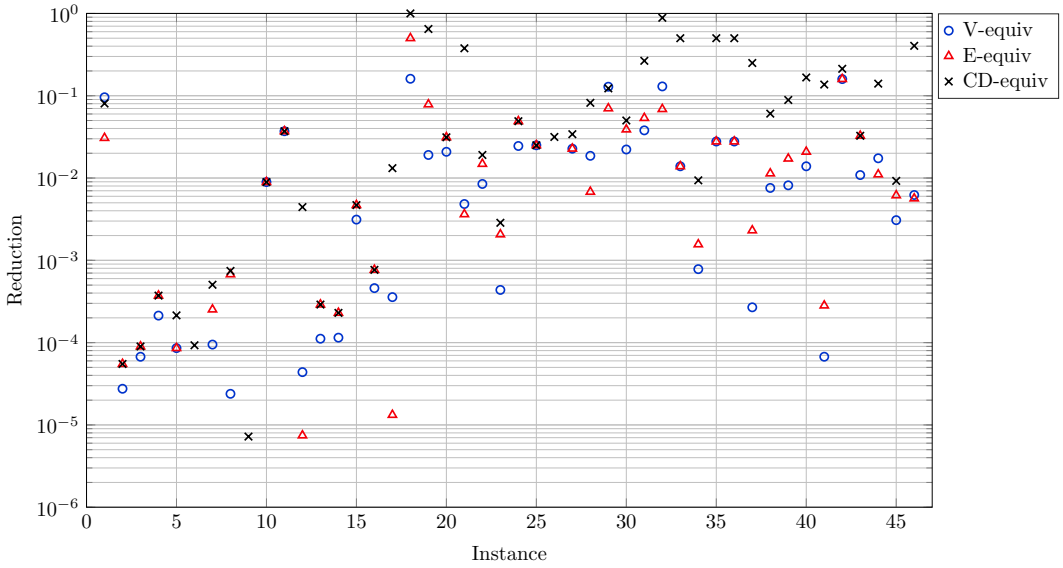


Figure 3 X-axis: All 46 instances (i.e. the pairs of datasets and cost vectors). Y-axis: In logarithmic scale, the Reduction value that is equal to the number of equivalence classes over the total number of reconciliations. For each instance, three points are plotted: the blue circle, the red triangle, and the black X, corresponding respectively to the V-, E-, and CD-equivalence relations. Four points of Reduction values less than 10^{-6} are omitted.

cost vector $(0, 2, 3, 1)$, the seven optimal event vectors are: $(102, 0, 284, 36)$, $(103, 0, 283, 39)$, $(104, 0, 282, 42)$, $(105, 0, 281, 45)$, $(106, 0, 280, 48)$, $(107, 0, 279, 51)$, and $(108, 0, 278, 54)$. From the list of event vectors, one can see that the dataset can be explained by a large number of host-switches and cospeciations, and that there have probably been no duplication. Therefore, by simply considering the equivalence classes one already has an idea of the diversity of the optimal reconciliations. Our approach is thus helpful for drawing conclusions about the co-evolutionary history of this pair of host/parasite association for which few prior analysis methods apply.

Table 1 Experimental results for the *Wolbachia* dataset and for each cost vector. $|L(H)|$ and $|L(S)|$ are the number of leaves of the host tree and the parasite tree; $|\mathcal{R}|$ is the number of optimal reconciliations; $|\mathrm{V_{eq}}|$, $|\mathrm{P_{eq}}|$, and $|\mathrm{CD_{eq}}|$ are respectively the number of V-, E-, and CD-equivalent classes. The dash indicates that the counting of the equivalence classes did not finish.

Dataset	$\lvert L(H)\rvert$	$\lvert L(S)\rvert$	Cost vector	$\lvert\mathcal{R}\rvert$	$\lvert\mathrm{V_{eq}}\rvert$	$\lvert\mathrm{E_{eq}}\rvert$	$\lvert\mathrm{CD_{eq}}\rvert$
Wolbachia [31, 30]	387	387	$(-1, 1, 1, 1)$	$\approx 10^{47}$	10	4080	24192
			$(0, 1, 1, 1)$	$\approx 10^{48}$	11	40960	76800
			$(0, 1, 2, 1)$	$\approx 10^{47}$	10	4080	24192
			$(0, 2, 3, 1)$	$\approx 10^{42}$	7	96	1152
			$(0, 1, 1, 0)$	$\approx 10^{136}$	—	$\approx 10^{27}$	—

Finally, the algorithm is quite efficient in practice, as for example for the cost vector $(-1, 1, 1, 1)$, to enumerate all the optimal event vectors, it took around 8 minutes for the dataset of *Wolbachia* and their arthropod hosts on a single thread of the Intel Core i5-3380M CPU. The enumeration of equivalence classes, together with other features such as the visualization of the E- and the CD-equivalence classes, is freely available in the software Capybara; more information can be found in [33].

3.4 Comparison with eMPRess

eMPRess [18, 28] is a tool that includes the possibility for the user to cluster the space of optimal solutions using agglomerative hierarchical clustering. The user can define the desired final number of clusters and a lower bound for the initial number of clusters (the actual initial number depends on the structure of the reconciliation graph, and can be much larger than the chosen lower bound). Then, pairs of clusters are merged using a linkage criterion until the desired number of clusters is obtained. The authors consider two different linkage criteria: (i) minimizing the average distance between the solutions within each cluster with respect to a given distance metric (the symmetric distance or the path distance), (ii) maximizing the average event support in each cluster.

As already mentioned in the introduction, the approach of eMPRess is fundamentally different from the one we propose. We believe that it is interesting to remark some of the differences between the two methods that the user should keep in mind when applying one method or the other.

It is important to notice that the results obtained with our algorithm and with eMPRess can be very different. Two solutions that may be considered equivalent may have a large symmetric or path distance. Indeed, the symmetric distance between two reconciliations is defined as the number of associations that are found in one reconciliation or the other but not in both. Inside an E-equivalence class, even though the type of the events is consistent among the reconciliations, all the associations can potentially be different, so the symmetric distance can take the largest possible value. Moreover, when using the event support criterion, it is important to keep in mind that within a cluster, by construction, the more ancestral events are more supported than the more recent events. While this may be biologically motivated, it is a bias that we may not want in some datasets.

These differences are also seen in practice as we applied eMPRess to some of the datasets used in the previous section, requiring that the number of final clusters is the same (or slightly larger) than the number of equivalence classes that we have found for that dataset. By analyzing the median reconciliations of the final clusters, we saw that, even for the V-equivalence relation (which is among those most analyzed in practical studies), some classes are not represented.

Finally, the worst case running time of the clustering method of eMPRess depends quadratically on the initial number of clusters and the time can be a limitation in practice. When we applied it to the *Wolbachia* dataset with the default cost vector $(0, 2, 3, 1)$ and the symmetric distance criterion, by starting with 336 initial clusters (level $L = 6$ in [18]) and choosing 10 as the final number of clusters, the software did not finish within 24 hours.

4 Conclusion

In this paper, we proposed a method that lists representative reconciliations from the (often huge) space of optimal solutions. To this purpose, we first defined when two reconciliations can be considered equivalent and then we provided efficient algorithms that output in polynomial delay only one reconciliation from each equivalence class. We proposed three different biologically motivated equivalence relations. We applied our algorithms to real datasets and showed that we were able to analyze the space of optimal reconciliations even in cases when the latter has a huge size (e.g. 10^{42}). As a future direction, we plan to extend our algorithms to other definitions of equivalence for reconciliations.

References

1 Steen A. Andersson, David Madigan, and Michael D. Perlman. A characterization of markov equivalence classes for acyclic digraphs. *Annals of Statistics*, 25(2):505–541, April 1997.

2 Mukul S. Bansal, Eric J. Alm, and Manolis Kellis. Reconciliation revisited: handling multiple optima when reconciling with duplication, transfer, and loss. *Journal of computational biology : a journal of computational molecular cell biology*, 20(10):738–754, October 2013. doi:10.1089/cmb.2013.0073.

3 Anselm Blumer, Janet A. Blumer, David H. Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, July 1987. doi:10.1145/28869.28873.

4 Marília D.V. Braga, Marie-France Sagot, Céline Scornavacca, and Eric Tannier. Exploring the solution space of sorting by reversals, with experiments and an application to evolution. *IEEE/ACM transactions on computational biology and bioinformatics*, 5(3):348–356, 2008. doi:10.1109/TCBB.2008.16.

5 Michael A. Charleston. Jungles: a new solution to the host/parasite phylogeny reconciliation problem. *Mathematical biosciences*, 149:191–223, 1998. doi:10.1016/s0025-5564(97)10012-8.

6 Michael A. Charleston. Recent results in cophylogeny mapping. *Advances in Parasitology*, 54:303–330, December 2003. doi:10.1016/s0065-308x(03)54007-6.

7 Beatrice Donati, Christian Baudet, Blerina Sinaimeri, Pierluigi Crescenzi, and Marie-France Sagot. Eucalypt: efficient tree reconciliation enumerator. *Algorithms for Molecular Biology*, 10(1):3, 2015. doi:10.1186/s13015-014-0031-3.

8 Graham J. Etherington, Susan M. Ring, Michael A. Charleston, Jo Dicks, Vic J. Rayward-Smith, and Ian N. Roberts. Tracing the origin and co-phylogeny of the caliciviruses. *Journal of General Virology*, 87(5):1229–1235, 2006. doi:10.1099/vir.0.81635-0.

9 Robert Giegerich, Björn Voß, and Marc Rehmsmeier. Abstract shapes of RNA. *Nucleic Acids Research*, 32(16):4843–4851, January 2004. doi:10.1093/nar/gkh779.

10 Morris Goodman, John Czelusniak, G. William Moore, A. E. Romero-Herrera, and Genji Matsuda. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Zoology*, 28(2):132–163, 1979. doi:10.2307/2412519.

11 Melissa Grueter, Kalani Duran, Ramya Ramalingam, and Ran Libeskind-Hadas. Reconciliation reconsidered: In search of a most representative reconciliation in the duplication-transfer-loss model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2019. doi:10.1109/TCBB.2019.2942015.

12 Jordan Haack, Eli Zupke, Andrew Ramirez, Yi-Chieh Wu, and Ran Libeskind-Hadas. Computing the diameter of the space of maximum parsimony reconciliations in the duplication-transfer-loss model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(1):14–22, 2019. doi:10.1109/TCBB.2018.2849732.

13 Katharina T Huber, Vincent Moulton, Marie-France Sagot, and Blerina Sinaimeri. Exploring and Visualizing Spaces of Tree Reconciliations. *Systematic Biology*, 68(4):607–618, December 2018. doi:10.1093/sysbio/syy075.

14 Katharina T. Huber, Vincent Moulton, Marie-France Sagot, and Blerina Sinaimeri. Geometric medians in reconciliation spaces of phylogenetic trees. *Information Processing Letters*, 136:96–101, 2018. doi:10.1016/j.ipl.2018.04.001.

15 Edwin Jacox, Cedric Chauve, Gergely J. Szöllősi, Yann Ponty, and Celine Scornavacca. ecceTERA: Comprehensive gene tree-species tree reconciliation using parsimony. *Bioinformatics*, 2016. doi:10.1093/bioinformatics/btw105.

16 Bonnie R. Lei and Kevin J. Olival. Contrasting patterns in mammal–bacteria coevolution: Bartonella and leptospira in bats and rodents. *PLOS Neglected Tropical Diseases*, 8(3):1–11, March 2014. doi:10.1371/journal.pntd.0002738.

17 Weiyun Ma, Dmitriy Smirnov, Juliet Forman, Annalise Schweickart, Carter Slocum, Srinidhi Srinivasan, and Ran Libeskind-Hadas. Dtl-rnb: Algorithms and tools for summarizing the space of dtl reconciliations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(2):411–421, 2018. doi:10.1109/TCBB.2016.2537319.

18 Ross Mawhorter and Ran Libeskind-Hadas. Hierarchical clustering of maximum parsimony reconciliations. *BMC Bioinformatics*, 20:612, 2019. doi:10.1186/s12859-019-3223-5.

19 Daniel Merkle and Martin Middendorf. Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information. *Theory in Biosciences*, 123:277–299, 2005. doi:10.1016/j.thbio.2005.01.003.

20 Daniel Merkle, Martin Middendorf, and Nicolas Wieseke. A parameter-adaptive dynamic programming approach for inferring cophylogenies. *BMC Bioinformatics*, 11(Supplementary 1):10 pages, January 2010. doi:10.1186/1471-2105-11-S1-S60.

21 Kazuyuki Narisawa, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Efficient computation of substring equivalence classes with suffix arrays. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching*, pages 340–351, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/s00453-016-0178-z.

22 Thi-Hau Nguyen, Vincent Ranwez, Vincent Berry, and Celine Scornavacca. Support measures to estimate the reliability of evolutionary events predicted by reconciliation methods. *PLOS ONE*, 8(10):1–14, 2013. doi:10.1371/journal.pone.0073667.

23 Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag Berlin Heidelberg, Tioga, Palo Alto, CA, 1982.

24 Alex Ozdemir, Michael Sheely, Daniel Bork, Ricson Cheng, Reyna Hulett, Jean Sung, Jincheng Wang, and Ran Libeskind-Hadas. Clustering the space of maximum parsimony reconciliations in the duplication-transfer-loss model. In *Algorithms for Computational Biology - 4th International Conference, AlCoB 2017, Aveiro, Portugal, June 5-6, 2017, Proceedings*, pages 127–139, 2017. doi:10.1007/978-3-319-58163-7_9.

25 Roderic D. M. Page. Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Systematic Biology*, 43(1):58–77, 1994. doi:10.1093/sysbio/43.1.58.

26 Pamela M. Pennington, Louisa Alexandra Messenger, Jeffrey Reina, José G. Juárez, Gena G. Lawrence, Ellen M. Dotson, Martin S. Llewellyn, and Celia Cordón-Rosales. The chagas disease domestic transmission cycle in guatemala: Parasite-vector switches and lack of mitochondrial co-diversification between triatoma dimidiata and trypanosoma cruzi subpopulations suggest non-vectorial parasite dispersal across the motagua valley. *Acta Tropica*, 151:80–87, 2015. Ecology and diversity of Trypanosoma cruzi. doi:10.1016/j.actatropica.2015.07.014.

27 Santi Santichaivekin, Ross Mawhorter, and Ran Libeskind-Hadas. An efficient exact algorithm for computing all pairwise distances between reconciliations in the duplication-transfer-loss model. *BMC Bioinformatics*, 20(20):636, 2019. doi:10.1186/s12859-019-3203-9.

28 Santi Santichaivekin, Qing Yang, Jingyi Liu, Ross Mawhorter, Justin Jiang, Trenton Wesley, Yi-Chieh Wu, and Ran Libeskind-Hadas. eMPRess: a systematic cophylogeny reconciliation tool. *Bioinformatics*, November 2020. doi:10.1093/bioinformatics/btaa978.

29 Celine Scornavacca, Wojciech Paprotny, Vincent Berry, and Vincent Ranwez. Representing a set of reconciliations in a compact way. *Journal of Bioinformatics and Computational Biology*, 11(02):1250025, 2013. doi:10.1142/S0219720012500254.

30 Patrícia M. Simões. *Diversity and dynamics of Wolbachia-host associations in arthropods from the Society archipelago, French Polynesia*. PhD thesis, University of Lyon 1, France, 2012. URL: https://tel.archives-ouvertes.fr/tel-00850707/file/SimoesP2012.pdf.

31 Patrícia M. Simões, Gladys Mialdea, Daphné Reiss, Marie-France Sagot, and Sylvain Charlat. *Wolbachia* detection: an assessment of standard PCR protocols. *Molecular Ecology Resources*, 11(3):567–572, 2011. doi:10.1111/j.1755-0998.2010.02955.x.

32 Maureen Stolzer, Han Lai, Minli Xu, Deepa Sathaye, Benjamin Vernot, and Dannie Durand. Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. *Bioinformatics*, 28(18):i409–i415, September 2012. doi:10.1093/bioinformatics/bts386.

33 Yishu Wang, Arnaud Mary, Marie-France Sagot, and Blerina Sinaimeri. Capybara: equivalence class enumeration of cophylogeny event-based reconciliations. *Bioinformatics*, 36(14):4197–4199, 2020. doi:10.1093/bioinformatics/btaa498.

34 Yishu Wang, Arnaud Mary, Marie-France Sagot, and Blerina Sinaimeri. A general framework for enumerating equivalence classes of solutions, 2021. (submitted). arXiv:2004.12143.

35 Nicolas Wieseke, Matthias Bernt, and Martin Middendorf. Unifying parsimonious tree reconciliation. In *Algorithms in Bioinformatics – 13th International Workshop, WABI 2013, Proceedings*, volume 8126 of *Lecture Notes in Computer Science*, pages 200–214, Berlin, Heidelberg, 2013. Springer. doi:10.1007/978-3-642-40453-5_16.

Perplexity: Evaluating Transcript Abundance Estimation in the Absence of Ground Truth

Jason Fan
University of Maryland, College Park, MD, USA

Skylar Chan
University of Maryland, College Park, MD, USA

Rob Patro
University of Maryland, College Park, MD, USA

Abstract

There has been rapid development of probabilistic models and inference methods for transcript abundance estimation from RNA-seq data. These models aim to accurately estimate transcript-level abundances, to account for different biases in the measurement process, and even to assess uncertainty in resulting estimates that can be propagated to subsequent analyses. The assumed accuracy of the estimates inferred by such methods underpin gene expression based analysis routinely carried out in the lab. Although hyperparameter selection is known to affect the distributions of inferred abundances (e.g. producing smooth versus sparse estimates), strategies for performing model selection in experimental data have been addressed informally at best.

Thus, we derive *perplexity* for evaluating abundance estimates on fragment sets directly. We adapt perplexity from the analogous metric used to evaluate language and topic models and extend the metric to carefully account for corner cases unique to RNA-seq. In experimental data, estimates with the best perplexity also best correlate with qPCR measurements. In simulated data, perplexity is well behaved and concordant with genome-wide measurements against ground truth and differential expression analysis.

To our knowledge, our study is the first to make possible model selection for transcript abundance estimation on experimental data in the absence of ground truth.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases RNA-seq, transcript abundance estimation, model selection

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.4

Supplementary Material *Software (Source Code)*: `https://github.com/COMBINE-lab/perplexity`
Software (Source Code): `https://github.com/COMBINE-lab/perplexity-paper`

Funding This work is supported by NIH R01 HG009937, and by NSF CCF-1750472, and CNS-1763680. Additionally, JF is supported by the NSF GRFP award no. DGE-1840340.
Conflicts of interest. RP is a co-founder of Ocean Genomics, Inc.

1 Introduction

Due to its accuracy, reproducibility, simplicity and low cost, RNA-seq has become one of the most popular high-throughput sequencing assays in contemporary use, and it has become the *de facto* method for the profiling of gene and transcript expression in many different biological systems. While there are many uses for RNA-seq that span the gamut from *de novo* transcriptome assembly [5, 10] through meta-transcriptome profiling [30], one of the most common uses is to interrogate the gene or isoform-level expression of known (or newly-assembled) transcripts, often with the subsequent goal of performing a differential analysis between conditions of interest.

Because of the popularity of gene and transcript expression profiling using RNA-seq, considerable effort has been expended in developing accurate, robust and efficient computational methods for inferring transcript abundance estimates from RNA-seq data. Some popular approaches focus on counting the aligned RNA-seq reads that overlap genes in different ways [1, 19]. However, these approaches have no principled way to deal with reads that align well to multiple loci (e.g. to different isoforms of a gene, or between sequence-similar regions of related genes), and this restricts their use primarily to gene-level analysis, where they may still under-perform more sophisticated approaches that attempt to resolve fragments of ambiguous origin [33].

Alternatively, many approaches offer the ability to estimate transcript-level expression using RNA-seq data (which can, if later desired by a user, be aggregated to the gene-level). The majority of these approaches perform statistical inference over a probabilistic generative model of the experiment based either on sufficient statistics of counts [13, 36] or the set of fragment alignments themselves [17]. Moreover, in addition to methods focused on deriving point estimates for transcript abundances, there has been considerable development of probabilistic Bayesian approaches for this inference problem [9, 11, 22, 23, 24, 26], as well as recent attempts at multi-sample probabilistic models for simultaneous experiment-wide transcript abundance estimation [14, 15]. Bayesian approaches can sometimes offer more accurate or robust inference than methods based strictly on maximum likelihood estimation, but these Bayesian models invariably expose prior distributions, with associated hyperparameters, upon which the resulting inferences depend.

Interestingly, the recommended best practices suggested by the different Bayesian (or variational Bayesian) approaches for selecting hyperparameters differ. Specifically, Nariai et al. [22] evaluate performance varying the prior used in their variational Bayesian expectation maximization (VBEM)-based method, and they conclude that a small prior (i.e. $\alpha < 1$) leads to a sparse solution, which, in turn, results in improved accuracy. On the other hand, Hensman et al. [11] perform inference using a prior of $\alpha = 1$ read per transcript. They find that, doing so, their method produces the most *robust* estimates (i.e. with the highest concordance between related replicates) that are also more accurate under different metrics that they measure. Their conclusion is that methods adopting a maximum likelihood model inferred using an expectation maximization procedure tend to produce sparse estimates close to the boundary of the parameter space which leads to less robust estimation among related samples. Unfortunately, regardless of how prior studies have argued for a "better" prior, none provide an empirical or practical procedure for model selection. Rather, they show that a value works well across a range of data under some evaluation metric, and set this as the default value for all inference tasks. Given the number of existing methods that can make use of prior information (including methods like those by Srivastava et al. [34] for single-cell data, or those by Liu et al. [20] that use orthogonal modalities of data to set priors), it becomes increasingly important to develop methods that lets one robustly and automatically select an appropriate prior (hyperparameter) for these algorithms.

To perform model (or hyperparameter) selection for transcript abundance estimators, one must be able to evaluate estimated abundances. However, evaluation of abundance estimates remains a challenge for current methods on experimental data where ground truth is completely absent. Notably, evaluation of transcript abundance estimators on experimental data have relied on careful experiment design that enables comparisons to complementary assays (e.g. correlation with qPCR) or measurements (e.g. concordance with known mixing proportions or spike-ins) [35]. Such evaluation procedures vary from study-to-study, and are simply not possible when complementary experiments are not designed or available. Thus, the natural question is then: *can the quality of transcript abundance estimates be meaningfully evaluated on the set of given fragments directly?*

It may initially be unintuitive to think that the "goodness" of a transcript abundance estimate can be evaluated in the absence of ground truth. However, in a related line of research, likelihood-based metrics for assessing the quality of *de novo* assemblies, where ground truth is unavailable, have been explored. For example, Rahman and Pachter [27] developed a method to compute the likelihoods of assembled genomes; Li et al. [18] developed a likelihood-based score to evaluate transcriptome assemblies; Smith-Unna et al. [32] developed a method to assess the quality of assembled contigs in transcriptomes; and Clark et al. [6] developed a method that is applicable to both genome and metagenomic assemblies. Furthermore, if we look to other unsupervised problem settings where ground truth annotations are absent, metrics for measuring the "goodness" of estimated models with latent parameters not only exist, but are regularly used. For example, metrics such as the silhouette score used to evaluate clustering algorithms come to mind [29]. In fact, evaluation of unsupervised probabilistic models, especially language and topic models in natural language processing, is commonplace [4, 12]. Specifically, *perplexity*, the inverse geometric mean per-*word* likelihood of a held-out test set, has been ubiquitously used to compare models [4].

In this work, we derive perplexity for transcript abundance estimation with respect to held-out per-*read* likelihoods. As we shall see, the perplexity of a held-out fragment set given an abundance estimate, computed via a quantify-then-validate approach, is a theoretically and experimentally motivated measure of the quality of the given estimate. Notably, perplexity quantifies an important biologically motivated intuition – that a good abundance estimate ought to generalize and generate the validation set, which is, in a sense, a form of a technical replicate, with high probability.

Perplexity can be used wherever the assessment of the quality of abundance estimates is desired. For example, perplexity can be used to compare different transcript abundance estimation algorithms or, as suggested above, to perform model selection to obtain the most accurate estimates from a given algorithm. In this work, we focus on experimentally assessing perplexity with respect to the latter, model selection for the prior used to estimate abundances with `salmon` [26]. In `salmon`, the reads-per-transcript prior size is a hyperparameter that controls its preference for inferring sparse or smooth abundance estimates. Notably, the problem of model selection offers a succinct assessment and immediately useful application of how perplexity can be computed to evaluate and compare the quality of candidate transcript abundance estimates.

1.1 Contributions

Theoretically, we derive and motivate a notion of *perplexity* for transcript abundance estimation – a metric for evaluating inferred estimates in the absence of ground truth. Experimentally, we demonstrate that perplexity for transcript abundance estimates is well behaved, and establish empirical correspondence between perplexity and other metrics that are more commonly used to demonstrate the "goodness" of transcript abundance estimates.

We summarize our experimental contributions as follows:

1. In experimental data from the Sequencing Quality Control (SEQC) consortium [35], we show that transcript abundance estimates with the lowest perplexity (lower is better) achieve the highest correlation with complementary qPCR measurements of biological replicates.
2. In simulated data, perplexity is concordant with respect to three measurements against ground truth: Spearman correlation with respect to expressed transcripts, AUROC with respect to unexpressed transcripts, and downstream differential transcript expression analysis.

Evidenced by these results, we propose perplexity as the first and, to our knowledge, only theoretically and experimentally justified metric for model selection for transcript abundance estimation in *experimental* data where ground truth is entirely absent.

2 Preliminaries: (Approximate) Likelihood for transcript abundance estimation

Before deriving *perplexity* for transcript abundance estimation, we shall briefly recall and define the necessary objects that pertain to the *likelihood* of the probabilistic model that underpins transcript abundance estimation (as in [17, 26]).

The transcript abundance estimation problem, or quantification, from short RNA-seq *fragments* (a term used to refer, generically, to either single reads or read pairs), is the problem of assigning each fragment f_j of an input fragment-set $\mathcal{F} = \{f_1, ...f_N\}$ to its transcript of origin. For this work, we shall only consider quantification with respect to a given reference transcriptome whereby a quantifier maps each input fragment f_j to a transcript in an input set of reference transcripts $\mathcal{T} = \{t_1, .., t_M\}$.

Given the sequence of an input fragment, said fragment may align to more than one transcript, t_i, in the reference transcriptome $\mathcal{T}$. Here, the *de facto* method for determining transcript of origin for fragments that multi-map to more than one transcript is to view the true fragment to transcript assignment as a latent variable, and to infer the latent variable's expected value by performing inference in the underlying probabilistic model.

Assuming an appropriate normalization of alignment scores, we write the probability of observing a fragment, f_j, given that it originates from (or aligns to) transcript t_i to be $P(f_j|t_i)$. The probability that a molecule in a sample that is selected for sequencing is the transcript t_i is then $P(t_i|\theta)$, a multinomial over $\mathcal{T}$. Marginalizing over all possible alignments, the *likelihood* of observing the fragment set $\mathcal{F}$ given model parameters θ is,

$$\mathcal{P}(\mathcal{F} \mid \theta) = \prod_{j}^{N} \sum_{i=1}^{M} P(t_i \mid \theta) \cdot \mathcal{P}(f_j \mid t_i). \tag{1}$$

In this work, we shall work with the *range-factorized* equivalence class approximation of the likelihood that has proven to be effective and is efficient to compute [38]. Here, sets of fragments in $\mathcal{F}$ that map to the same set of transcripts, and have similar conditional probabilities of arising from these transcripts, are said to belong to the equivalence class $\mathcal{F}^q$ (indexed by q). Instead of working with alignment probabilities $\mathcal{P}(f_j|t_i)$ of each fragment, fragments in an equivalence class $\mathcal{F}^q$ are approximated to have the same conditional probability $\mathcal{P}(f_j|\mathcal{F}^q, t_i)$ for mapping to each transcript t_i. Let $\mathcal{C}$ be the set of equivalence classes induced by $\mathcal{F}$ and $\Omega(\mathcal{F}^q)$ be the set of transcripts to which $f \in \mathcal{F}_q$ map. The range-factorized equivalence class approximation of the likelihood $\mathcal{P}(\mathcal{F} \mid \theta)$ is,

$$\mathcal{P}(\mathcal{F} \mid \theta) \approx \prod_{\mathcal{F}^q \in \mathcal{C}} \left(\sum_{t_i \in \Omega(\mathcal{F}^q)} P(t_i \mid \theta) \cdot \mathcal{P}(f_j \mid \mathcal{F}^q, t_i) \right)^{N^q}. \tag{2}$$

Here, the approximate likelihood can be computed over the number of unique equivalence classes, which is considerably smaller than the number of all possible alignments for all fragments.

3 Methods

We propose a subtle but instructive change in the usual computational protocol for evaluating transcript abundance estimates. We propose a *quantify-then-validate* approach which evaluates the quality of transcript abundance estimates directly on read-sets, analogous to *train-then-test* approaches for evaluating probabilistic predictors common in natural language processing (NLP) and other fields [3, Ch. 1.3]. Instead of quantifying all available fragments and then performing evaluation with respect to complementary measurements downstream, the quantify-then-validate approach validates and evaluates the quality of a given abundance estimate directly on a set of held-out *validation* fragments withheld from inference.

We derive and adapt from NLP, the notion of *perplexity* for transcript abundance estimation for this quantify-then-validate approach [4, 12]. Perplexity is computed given only an abundance estimate, and a held-out validation set of fragments as input. Thus, perplexity evaluates the quality of abundance estimates on fragments directly and can evaluate estimates from experimental data in the absence of ground truth. Most importantly, evaluating perplexity with the quantify-then-validate approach enables quantitative, evidence-based, cross-validated selection of hyperparameters for transcript abundance estimation methods that use them.

Perplexity for transcript abundance estimation quantifies the intuition that an abundance estimate for a given sample ought, with high probability, explain and generate the set of fragments of a technical replicate. The key observation is that the likelihood $\mathcal{P}(\mathcal{F}|\theta)$ is simply a value that can be computed for any fragment set $\mathcal{F}$ and any abundance estimate θ (model parameters), irrespective of whether θ is inferred from $\mathcal{F}$. It is the context and application of the likelihood, $\mathcal{P}(\mathcal{F}|\theta)$, that yields semantic meaning.

Given a fragment set, $\mathbf{F}$, over which one seeks to infer and evaluate abundance estimates, the quantify-then-validate procedure is as follows. First, partition the input set into a *quantified* set, $\mathcal{F}$, and a *validation* set, $\widehat{\mathcal{F}}$. Second, "quantify" and infer abundance estimates (model parameters) θ given the quantified set $\mathcal{F}$. Third, validate and compute the perplexity, $PP(\widehat{\mathcal{F}}, \theta)$ – the inverse geometric mean held-out per-read likelihood of observing the validation set, $\widehat{\mathcal{F}}$ – given model parameters θ and the validation set $\widehat{\mathcal{F}}$. The lower the perplexity, the better the parameters θ describe the held-out fragments $\widehat{\mathcal{F}}$, and the better the abundance estimate parameterized by θ ought to be. In fact, if we believe that the generative model is truly descriptive of the distributions that arise from the underlying biological and technical phenomena, perplexity is, in expectation, minimized when the "true" latent parameters are inferred.

Formally, given an abundance estimate θ, and a validation fragment-set $\widehat{\mathcal{F}} = \{\hat{f}_1, \ldots, \hat{f}_{\widehat{N}}\}$, the perplexity for transcript abundance estimation is:

$$\begin{aligned} PP(\widehat{\mathcal{F}}, \theta) &= \exp\left\{-\frac{1}{\widehat{N}} \log \mathcal{P}(\widehat{\mathcal{F}} \mid \theta)\right\} = \exp\left\{-\frac{1}{\widehat{N}} \sum_{j=1}^{\widehat{N}} \log \mathcal{P}(\hat{f}_j \mid \theta)\right\} \\ &= \exp\left\{-\frac{1}{\widehat{N}} \sum_{j=1}^{\widehat{N}} \log \sum_{i=1}^{M} \mathcal{P}(t_i \mid \theta) \cdot \mathcal{P}(\hat{f}_j \mid t_i)\right\}. \end{aligned} \tag{3}$$

Crucially, the probability $\mathcal{P}(\hat{f}_j \mid \theta)$ of observing each held out fragment given θ is computed and marginalized over two terms, $\mathcal{P}(\hat{f}_j \mid t_i)$ that depends only on the validation set of held-out fragments, and $P(t_i \mid \theta)$ that depends only on the given abundance estimate.

One particular application of the perplexity metric, which we explore here, is to select the best abundance estimate out of many candidate estimates arising from different hyperparameter settings for quantifiers. Thus, in this work, we use the range-factorized equivalence

class approximation for perplexity (as in Eq. 2) throughout [38]. Given the range-factorized equivalence classes, $\widehat{\mathcal{C}}$, induced by the *validation* set, $\widehat{\mathcal{F}}$, (where $\widehat{N}^q$ is the number of fragments in an equivalence class $\widehat{\mathcal{F}}^q \in \widehat{\mathcal{C}}$) the approximation is:

$$PP(\widehat{\mathcal{F}}, \theta) \approx \exp\left\{ -\frac{1}{\widehat{N}} \sum_{\widehat{\mathcal{F}}^q \in \widehat{\mathcal{C}}} \widehat{N}^q \log\left(\sum_{t_i \in \Omega(\widehat{\mathcal{F}}^q)} P(t_i \mid \theta) \cdot \mathcal{P}(\hat{f}_j \mid \widehat{\mathcal{F}}^q, t_i) \right) \right\}. \tag{4}$$

We use `salmon`'s selective-alignment based probabilistic model for conditional probabilities $\mathcal{P}(\hat{f}_j|\widehat{\mathcal{F}}^q, t_i)$ and effective lengths of transcripts, since the model and equivalence class approximation `salmon` uses has proven to be a fast and effective way to approximate the full likelihood [15, 38]. For the scope of this work, `salmon`'s format for storing range-factorized equivalence classes conveniently contains all relevant information and values to compute perplexity with vastly smaller space requirements than would be required to store per-fragment alignment probabilities $P(f_j|t_i)$.

3.1 "*Impossible*" fragments under parameter estimates θ

We now address a perplexity-related issue that is unique to evaluating transcript abundance estimates – that an observed event in the validation set may be deemed "impossible" given model parameters θ. The marginal probability, $\mathcal{P}(\hat{f}_j|\theta)$, for observing a fragment $\hat{f}_j$ in the validation set given some abundance estimate, θ, may actually be zero, even if said validation fragment aligns to the reference transcriptome. This occurs exactly when all transcripts, t_i, to which the validation fragment $\hat{f}_j$ map are deemed unexpressed by θ (i.e. $P(t_i|\theta) = 0$ for all such transcripts). Here, we say that $\hat{f}_j$ is an *impossible fragment* given θ, and that θ *calls* $\hat{f}_j$ impossible. When impossible fragments are observed in the validation set, perplexity is not a meaningful measurement.

To illustrate how impossible fragments come to be, consider the toy example in which all fragments in a quantified set that align to transcripts A, B, or C only ambiguously map to $\{A, B\}$, or to $\{A, C\}$. That is, no such fragments uniquely map – a phenomenon observed rather frequently for groups of similar isoforms expressed at low to moderate levels. Now, suppose that an abundance estimation model assigns all such fragments to transcript A and produces an estimate θ. The quantifier may be satisfying a prior that prefers sparsity; or prefers to do so because transcript A is considerably shorter than transcripts B and C, which gives it a higher conditional probability under a length normalized model. In this case, the marginal probability, $\mathcal{P}(\hat{f}_j|\theta)$, of observing a validation fragment $\hat{f}_j$ that maps to $\{B, C\}$ is exactly zero given the parameters θ.

As an example, we randomly withhold varying percentages of fragments from one sample (`SRR1265495`) as validation sets and use all remaining fragments to estimate transcript abundances with `salmon`'s default model (i.e. the VBEM model using prior size of 0.01 reads-per-transcript). Figure 1 shows that at all partitioned percentages, impossible fragments in the validation set are prevalent with respect to estimated abundances. In fact, due to the prevalence of impossible reads, perplexity as written in Eq. 4 is undefined (or infinite) for all estimates and all validation sets in the experiments below. An important observation in both the toy and experimental examples is that there likely exist better abundance estimates that would call fewer fragments impossible, while still assigning high likelihood to the rest of the (possible) fragments. For example, an abundance estimate that reserves even some small probability mass to transcript B in the toy example would not call the validation fragments in question impossible.

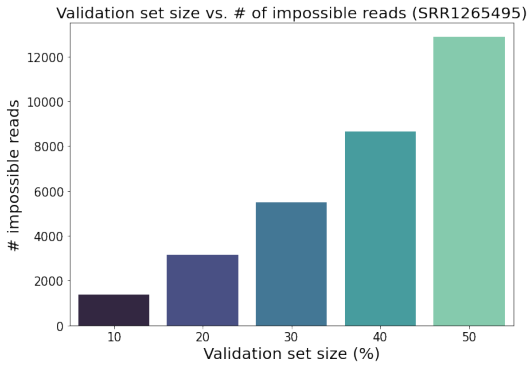


Figure 1 Number of fragments called impossible versus withheld validation fragment set size for sample `SRR1265495`. All remaining fragments are used to estimate abundances using `salmon`'s VBEM model using default parameters (i.e. using a prior size of 0.01 reads-per-transcript).

3.2 Smoothed perplexity: accounting for "*impossible*" validation fragments

The problem with impossible fragments is not only that they exist. It is that, for a fixed validation fragment set, perplexity deems an abundance estimate that calls even one fragment impossible equally as bad as an abundance estimate that calls all fragments impossible. However, the former is clearly preferable to the latter. Furthermore, as we shall see in the experiments that follow, the number of fragments called impossible by an abundance estimate can actually be indicative of inaccuracies with respect to estimated abundances of transcripts called expressed by θ. Thus, one must quantitatively account for impossible fragments to enable the comparison of estimates that call some validation fragments impossible.

Other fields that have adopted and used perplexity (e.g. NLP) usually sidestep the issue of impossible events entirely both by construction and pre-processing, working only with smoothed probabilistic models in which no-event has probability zero, or removing rare words from input language corpora. However, neither strategy is available nor appropriate for evaluating transcript abundance estimates. It is neither reasonable nor useful to amend and modify each of the many modern quantifiers to produce smooth outputs (outputs in which no transcript has truly zero abundance), and fragments and transcripts cannot be pre-processed away since the set of expressed transcripts cannot be identified *a priori*. One may also be tempted to simply remove impossible fragments from a validation set, $\widehat{\mathcal{F}}$, before computing a perplexity or hold out fragments – but this also is not a valid strategy. This is because two different abundance estimates θ and θ' may call different validation fragments in $\widehat{\mathcal{F}}$ impossible, and comparisons of likelihoods $P(\widehat{\mathcal{F}}'|\theta')$ and $P(\widehat{\mathcal{F}}|\theta)$ are only meaningful if the validation sets are the same (i.e. $\widehat{\mathcal{F}} = \widehat{\mathcal{F}}'$). Furthermore, there is no straightforward strategy to sample and hold-out validation fragments so that no fragments are impossible. This is because most validation fragments cannot be determined to be impossible prior to abundance estimation, and any non-uniform sampling strategy would alter the underlying distributions that estimators aim to infer.

Thus, we propose a *smoothed perplexity* measure to evaluate the quality of abundance estimates in which a consistent smoothing scheme can be fairly applied to any given abundance estimate. By smoothing an input abundance estimate, impossible fragments result in a penalty instead of immediately shrinking $P(\widehat{\mathcal{F}}|\theta)$ to zero. More concretely, we define smoothed perplexity given abundance estimate θ to be the perplexity evaluated with respect to the

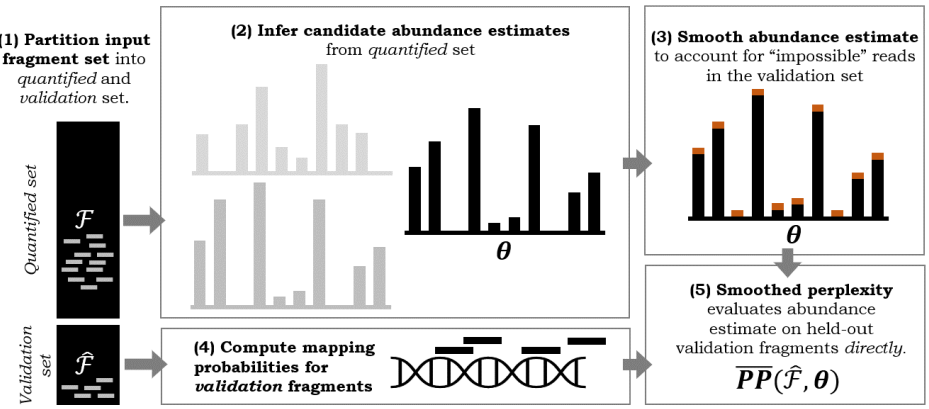


Figure 2 Overview of the *quantify-then-validate* approach using *smoothed perplexity* to evaluate the quality of abundance estimates directly on fragment sets in the absence of ground truth. (1) An input fragment set is first partitioned into a *quantified* and a *validation* set. (2) Abundance estimates for different candidate models (e.g. for explored hyperparameters as part of model selection) are inferred from the *quantified* fragment set only. (3) To account for "*impossible*" fragments and avoid shrinkage to unbounded perplexities, given abundance estimates are smoothed (see Sections 3.1 and 3.2). (4) Mapping probabilities to the reference transcriptome are computed for fragments in the validation set. (5) *Smoothed perplexity* computed given each input abundance estimate and the held-out validation fragment set can be used to evaluate and perform model selection – the lower the perplexity, the better an abundance estimate describes the held-out set of validation fragments.

smoothed distribution $\mathcal{P}(t_i|s_\beta(\theta))$. The Laplacian smoothing scheme $s_\beta(\theta)$ smooths input abundance estimate θ by redistributing a small constant probability mass. Let $\mathcal{P}(t_i|\theta) = \eta_i$, and M be the number of transcripts in the reference, the smoothed distribution $\mathcal{P}(t_i|s_\beta(\theta))$, parameterized by β, is defined:

$$\mathcal{P}(t_i|s_\beta(\theta)) = \frac{\eta_i + \beta}{1 + M\beta}. \tag{5}$$

This is equivalent to adding, for each transcript t_i in the reference, $\beta \cdot \sum_j^M c_j/\tilde{l}_j$ reads-per-nucleotide to the expected fragment counts c_i then re-normalizing to obtain TPMs, given the model parameters θ and effective transcript lengths $\tilde{l}_i$ (as defined in `salmon` [26]).

We are now ready to define *smoothed perplexity* in full. Given an abundance estimate θ and a validation set of fragments $\hat{\mathcal{F}}$, the smoothed perplexity measure $\overline{PP}(\hat{\mathcal{F}}, \theta)$ is,

$$\overline{PP}(\hat{\mathcal{F}}, \theta) = \exp\left\{ -\frac{1}{\hat{N}} \sum_{\hat{\mathcal{F}}^q \in \hat{\mathcal{C}}} \hat{N}^q \log\left(\sum_{t_i \in \Omega(\hat{\mathcal{F}}^q)} \mathcal{P}(t_i \mid s_\beta(\theta)) \cdot \mathcal{P}(\hat{f}_j \mid \hat{\mathcal{F}}^q, t_i) \right) \right\}. \tag{6}$$

We schematically illustrate how smoothed perplexity using the proposed quantify-then-validate protocol is computed to evaluate the quality of transcript abundance estimates in Figure 2.

For all following sections, for brevity, we shall use *perplexity* to mean *smoothed perplexity* unless stated otherwise.

3.3 Model selection using perplexity in practice

Arguably, one of the most useful outcomes of being able to evaluate the quality of abundance estimates in the absence of ground truth is the ability to perform model selection for transcript abundance estimation in experimental data. For those familiar with train-then-test

experimental protocols for model selection in machine learning or NLP, model selection for transcript abundance estimation *vis-a-vis* our proposed quantify-then-validate approach is analogous and identical in abstraction. However, since, to our knowledge, this work is the first to propose a quantify-then-validate approach for transcript abundance estimation, we shall briefly detail how perplexity ought to be used in practice.

Let us consider model selection via 5-fold cross-validation using perplexity given some fragment set $\mathbf{F}$. First, $\mathbf{F}$ is randomly partitioned into five equal sized, mutually exclusive validation sets, $\{\widehat{\mathcal{F}}_1, \ldots, \widehat{\mathcal{F}}_5\}$ – and quantified sets are subsequently defined, $\mathcal{F}_i = \mathbf{F} - \widehat{\mathcal{F}}_i$. Now, suppose we desire to choose between L model configurations (e.g. from L hyperparameter settings). Then for each ℓ-th candidate model, we produce a transcript abundance estimate from each i-th quantified set, $\theta_i^{(\ell)}$. To select the best out of the L candidate models, one simply selects the model that minimizes the average perplexity over the five folds, $\frac{1}{5}\sum_i \overline{PP}(\widehat{\mathcal{F}}_i, \theta_i^{(\ell)})$.

One additional practical consideration should also be noted. Given *any* pair of quantification and validation sets $\mathcal{F}$ and $\widehat{\mathcal{F}}$, a validation fragment, $\hat{f}_j \in \widehat{\mathcal{F}}$, can be *necessarily impossible*. A necessarily impossible validation fragment is one that maps to a set of transcripts to which no fragments in the quantified set $\mathcal{F}$ also map. Such a fragment will always be called impossible given any abundance estimate deriving from the quantified set $\mathcal{F}$, since no fragments in $\mathcal{F}$ provide any evidence that transcripts to which $\hat{f}_j$ map are expressed.

It is of limited meaning to evaluate estimates with respect to necessarily impossible fragments. For the purposes of this work, we shall consider the penalization of an abundance estimate only with respect to impossible fragments that are recoverable – in other words, fragments that could be assigned non-zero probability given a better abundance estimate inferable from $\mathcal{F}$. As such, we remove necessarily impossible validation fragments from $\widehat{\mathcal{F}}$, given $\mathcal{F}$, prior to computing perplexity.

3.4 Data

3.4.1 Sequencing Quality Control (SEQC) project data

We downloaded Illumina HiSeq 2000 sequenced data consisting of 100+100 nucleotide paired-end reads from the Sequencing Quality Control (SEQC) project [35]. SEQC samples are labeled by four different conditions $\{A, B, C, D\}$, with condition A being Universal Human Reference RNA and B being Human Brain Reference RNA from the MAQC consortium [31], with additional spike-ins of synthetic RNA from the External RNA Control Consortium (ERCC) [2]. Conditions C and D are generated by mixing A and B in 3:1 and 1:3 ratios, respectively.

In this work, we analyze the first four replicates from each condition sequenced at the Beijing Genomics Institute (BGI) – one of three official SEQC sequencing centers. For each sample, we aggregate fragments sequenced by all lanes from the flowcell with the lexicographically smallest identifier.[1] Quantitative PCR (qPCR) data of technical replicates for each sample in each condition are downloaded via the `seqc` BioConductor package.

3.4.2 Simulated lung transcript expression data

We simulated read-sets based on 10 sequenced healthy lung samples, with Sequence Read Archive accession number `SRR1265{495-504}` [16]. Transcript abundance estimates inferred by Salmon using the `--useEM` flag for each sample are used as ground truth abundances for

[1] Scripts to download and aggregate SEQC data are available at `github.com/thejasonfan/SEQC-data`.

read simulation (expressed in transcripts per million (TPM) and expected read-per-transcript counts). Then, transcript abundances in samples `SRR1265{495-499}`, for 10% of transcripts expressed in at least one of the five samples, are artificially up or down regulated by a constant factor (2.0×) to simulate differential transcript expression. We treat the resulting read-per-transcript counts as ground truth, and generate for each sample a fragments set of 100+100 nucleotide paired-end reads using Polyester at a uniform error rate of 0.001 with no sequence specific bias [8].

3.5 Evaluation and experiments

The purpose of the experiments in this work are twofold. First, to establish the relationship and correspondence between perplexity and commonly used measures of goodness or accuracy in transcript abundance estimation. And second, to demonstrate how model and hyperparameter selection can be performed using perplexity. In particular, we perform and evaluate hyperparameter selection for `salmon` with respect to the prior size in the variational Bayesian expectation maximization (VBEM) model used for inference [26]. The user-selected prior size for the VBEM model in `salmon` encodes the prior belief in the number of reads-per-transcript expected for any inferred abundance estimate. This hyperparameter controls `salmon`'s preference for inferring sparse or smooth estimates – the smaller the prior size, the sparser an estimate `salmon` will prefer. As discussed above, prior studies on Bayesian models have not necessarily agreed on how sparse or smooth a good estimate ought to be [11, 22] – the experiments in this work aim to provide a quantitative framework to settle this disagreement.

We perform all experiments according to the proposed quantify-then-validate procedure and report results with respect to various metrics over a 5-fold cross-validation protocol. We set the smoothing parameter for perplexity to $\beta = 10^{-8}$ for all experiments. We use the Ensembl human reference transcriptome GRCh37 (release 100) for all abundance estimation and analysis [37].

3.5.1 Evaluation versus parallel SEQC qPCR measurements

We analyze the relationship between perplexity and accurate abundance estimation in experimental data from the SEQC consortium. In SEQC data, we evaluate accuracy of abundances estimated by `salmon` by comparing estimates to qPCR gene expression data on biological replicates, a coarse proxy to ground truth. We evaluate the Spearman correlation between gene expressions of qPCR probed genes in SEQC replicates versus the corresponding abundance estimates. Gene expression from estimated transcript expression is aggregated via `txImport` [33] with transcript-to-gene annotations from `EnsDb.Hsapiens.v86` [28]. From gene expression data, Ensembl genes are mapped to corresponding Entrez IDs via `biomaRt` [7], and 897 genes are found to have a corresponding qPCR measurement in downloaded SEQC data. Expressions for genes with repeated entries in SEQC qPCR data are averaged.

3.5.2 Evaluation versus ground truth on simulated data

In simulated data, since ground truth abundances are available, we compare estimated TPMs (computed by `salmon`) against ground truth TPMs under two metrics.

First, we consider the Spearman correlation with respect to known expressed transcripts (i.e. transcripts with non-zero expression in ground truth abundances). We choose to evaluate Spearman correlation with respect to ground truth non-zero TPMs because of the presence

of many unexpressed transcripts in the ground truth, meaning a high number of values tied at rank zero. Here, small deviations from zeros can lead to large changes in rank, leading to non-trivial differences in the resulting Spearman correlation metric. We demonstrate this phenomenon with respect to the ground truth abundance of a simulated sample (`SRR1265495`) with a mean TPM of 5.98, in which 49% of transcripts are unexpressed (82,358 / 167,268). We report the change in Pearson correlation, R^2 score, and Spearman correlation of ground truth TPMs versus ground truth TPMs perturbed with normally distributed noise at varying standard deviations. As we can see from Figure 3, even small perturbations cause non-trivial changes in Spearman rank correlation, while changes in Pearson correlation are entirely imperceptible. The Pearson correlation, however, suffers from the well known problem that, in long-tailed distributions spanning a large dynamic range, like those commonly observed for transcript abundances, the Pearson correlation is largely dominated by the most abundant transcripts.

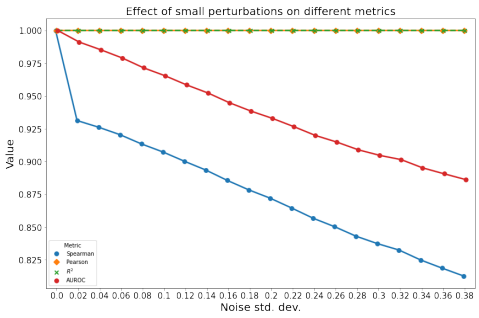


Figure 3 Spearman correlation, Pearson correlation and R^2 with respect to all transcripts in the reference, and AUROC for recalling ground truth unexpressed transcripts, with respect to added normally distributed noise with varying standard deviations. Plotted lines for Pearson correlation and R^2 overlap.

Second, we complement measuring Spearman correlation of non-zero ground truth TPMs with reporting the area under receiver operating characteristic (AUROC) for recalling ground truth zeros based on estimated abundances. While the measurement of Spearman correlation on the truly expressed transcripts is robust to small changes in predicted abundance near zero, it fails to account for false postive predictions even if they are of non-trivial abundance. The complementary metric of the AUROC for recalling ground truth zeros complements that metric, since it is affected by false positive predictions.

3.5.3 Differential expression analysis on simulated data

We perform transcript level differential expression analysis and analyze the recall of known differentially expressed transcripts in simulated lung tissue data (See 3.4.2). We perform differential expression analysis at the trancript level using `swish` [39] using 20 inferential replicates from `salmon`. We modified `salmon` to ensure that prior sizes supplied via the `--vbPrior` flag are propagated to the Gibbs sampling algorithm. We plot receiver operating characteristic (ROC) curves and report the mean AUROC for predicting differentially expressed transcripts over multiple folds. We assign $P = 1$ to transcripts for which `swish` does not assign adjusted P-values.

3.6 Implementation

We implement smoothed perplexity in Rust and provide `snakemake` [21] workflows to (1) set up quantified-validate splits of read-sets for K-fold cross-validation, and (2) compute perplexities of `salmon` abundance estimates with respect to validation fragment sets at: `https://github.com/COMBINE-lab/perplexity`. Code to reproduce the experiments and figures for this work is available at `https://github.com/COMBINE-lab/perplexity-paper`.

4 Results

4.1 Lower perplexity implies more accurate abundance estimates in experimental SEQC data

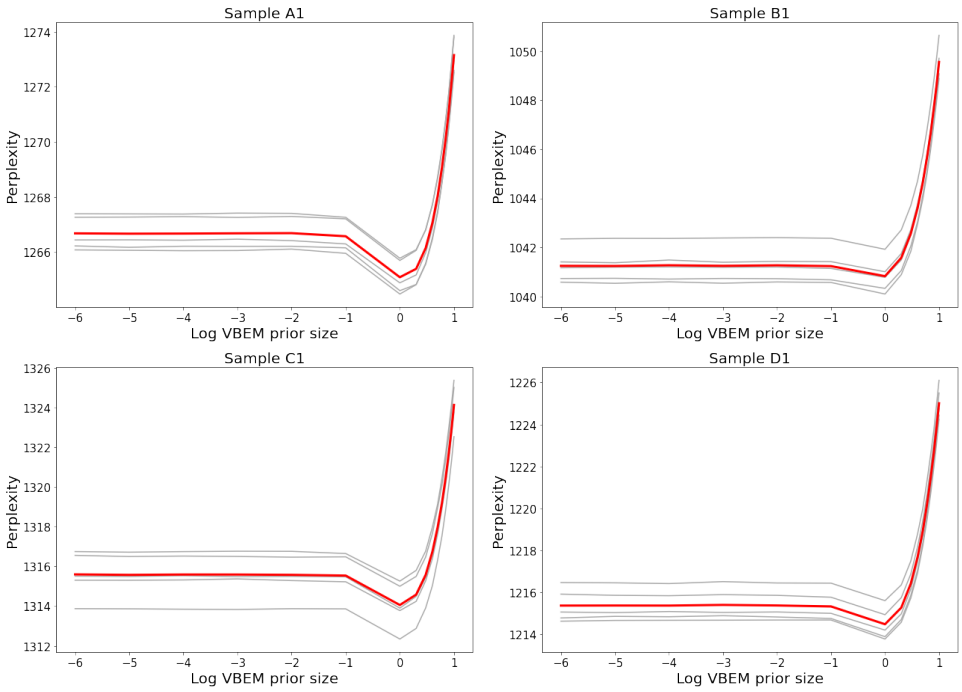


Figure 4 Perplexity plots for SEQC samples. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples – plots only for the first replicate of samples from conditions *A*-*D* are shown. Perplexity plots for other replicates are consistent within condition and are included in the Appendix. Mean perplexities across five folds are plotted in red, and perplexities for each fold are plotted in gray.

In experimental data from the Sequencing Quality Control (SEQC) project [35], we demonstrate that perplexity can be used to perform parameter selection and select the `salmon` VBEM prior size that leads to the most accurate transcript abundance estimates. We note that perplexity plots for replicates are similar within conditions *A*-*D*, and thus include only plots for the first replicate in each condition in the main text – plots for other samples are presented in the Appendix, Figure A1, for completeness.

Empirically, perplexity is well-behaved over all samples in the experimental data. As shown in Figure 4 and 5, plots of perplexity against VBEM prior size and Spearman correlation against VBEM prior size both display an empirically convex shape minimized at the same VBEM prior size. This suggests that minimizing perplexity is, at least, locally optimal with respect to the set of explored hyperparameters.

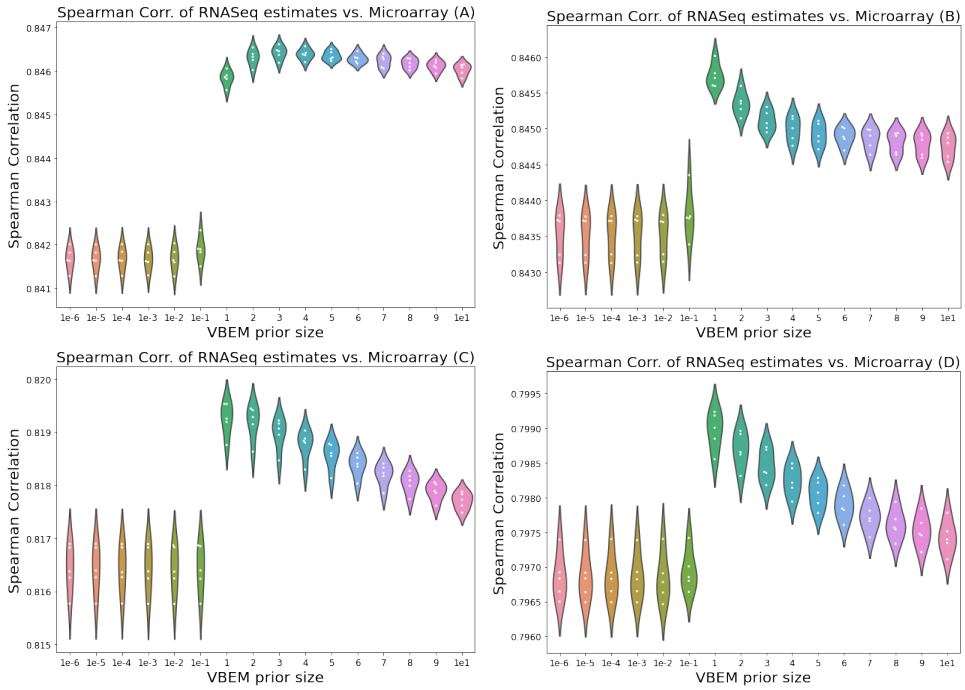


Figure 5 Spearman correlation of abundance estimates at various VBEM reads-per-transcript prior sizes, versus parallel qPCR microarray gene-expression measurements conditions A-D. Each point in above plots indicate the mean correlation across replicates for a given fold.

Furthermore, for almost all samples, perplexity is minimized where correlation with qPCR measurements is maximized. For all replicates in conditions $\{B, C, D\}$, estimates that minimize perplexity with respect to held-out validation fragments achieve the best correlation with qPCR measured gene expression. For replicates in these conditions, abundances inferred using a prior size of 1 read-per-transcript resulted in estimates with the lowest perplexity. In replicates from condition A, estimates with lowest perplexity are significantly better than estimates at default hyperparameter settings (0.01 reads-per-transcript).

Perhaps surprisingly, both perplexity and correlation against qPCR measurements prefer a reads-per-transcript prior size that is larger than the 0.01 reads-per-transcript that is the current default for the `salmon` VBEM model. Selecting a larger per-transcript prior for transcript abundance estimation with `salmon` results in estimates that are more smooth and less sparse. For smoother abundance estimates, fewer validation time fragments are likely called impossible (compared to sparser estimates). In these cases, the number of impossible reads called by an estimate not only indicates inferential errors with regard to transcripts incorrectly called unexpressed, but likely suggests less accurate inferred abundances with respect to transcripts that are called expressed.

To the best of our knowledge, this experiment is the first to carry out both an effective and ubiquitously applicable quantitative strategy to perform model selection in the context of transcript abundance estimation on experimental data in the absence of ground truth.

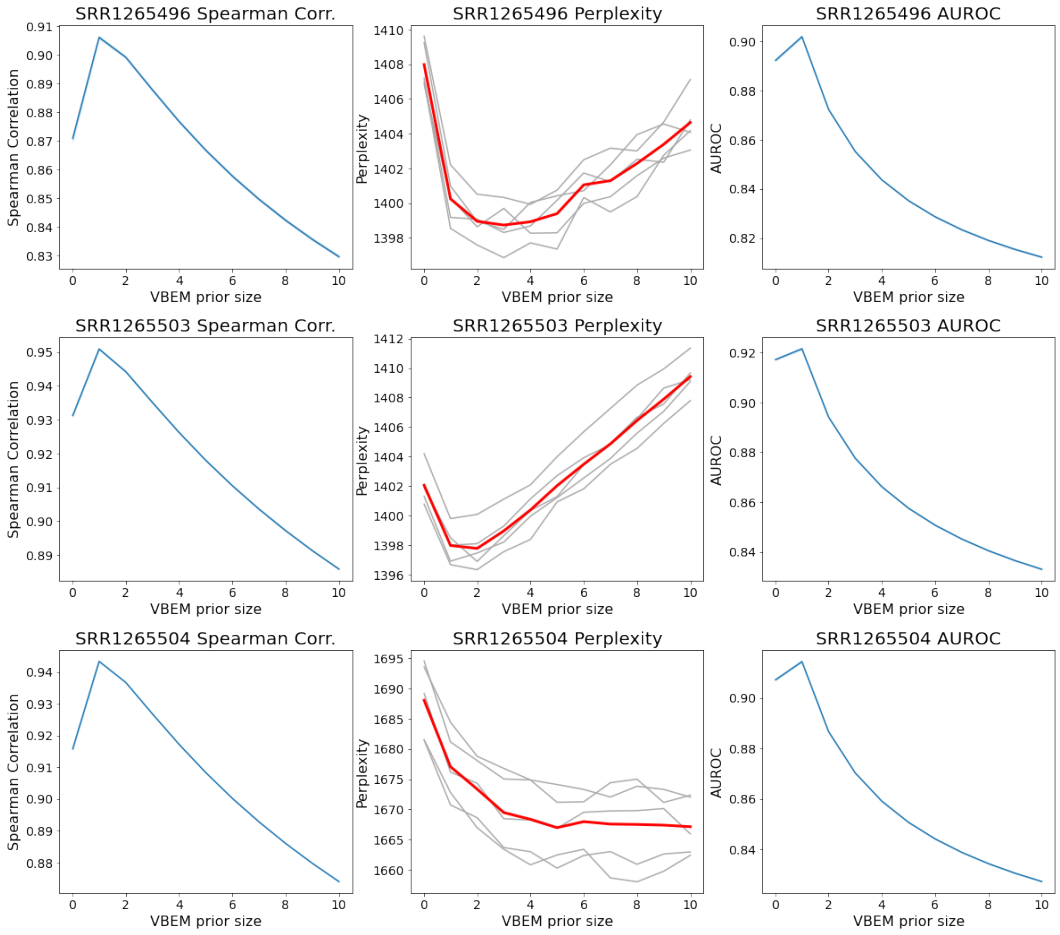


Figure 6 Quality of transcript abundance estimates as a function of VBEM per-nucleotide prior size for samples `SRR1265{496,503,504}`. (Left column) Spearman Correlation with respect ground truth expressed transcripts. (Middle column) Perplexity of abundance estimates; perplexities per-fold indicated in gray and mean perplexities in red. (Right column) AUROC for retrieving ground truth unexpressed transcripts. Leftmost plotted points for all plots use default `salmon` VBEM prior size of 0.01 reads-per-transcript.

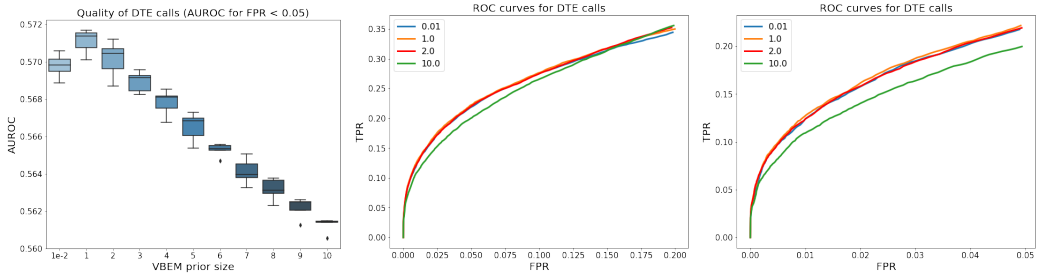


Figure 7 Accuracy of differential expression analysis with respect to experiment-wide selection of VBEM per-nucleotide prior size. (Left) AUROC with respect to DTE calls at real FPRs up to 0.05. (Middle) ROC curve up to FPR = 0.20. (Right) ROC curve up to FPR = 0.05. To reduce visual clutter, only the ROC curves some representative VBEM prior size settings are plotted.

4.2 Perplexity versus ground truth, and differential expression analysis in simulated data

In simulated data, the relationship between perplexity and measurements against ground truth, though well-behaved, is admittedly less direct. In short, under the experimental framework we have chosen, minimizing perplexity does not always find the best performing estimates. Across all 10 samples, perplexity prefers abundance estimates that are smoother than estimates that are most accurate when compared to ground truth. For brevity, we include in the main text perplexity plots of three samples (`SRR1265{496,503,504}`) that are representative of three main modalities of perplexity plot behaviors (Figure. 6). For completeness, and refer the reader to the appendix for analagous plots for the seven remaining samples (Figures A2 and A3).

In all but one sample (`SRR1265504`), perplexity plots display a empirically convex shape with a local minima close to the optimal VBEM prior size (1 read-per-transcript). For example, for sample `SRR1265503`, perplexity is minimized at a VBEM prior setting of 2 reads-per-transcript, the second best performing hyperparameter setting with respect to Spearman correlation (Figure. 6; middle). And for sample `SRR1265496`, we can clearly see that perplexity prefers VBEM prior setting in a wide local minima ranging from 2 to 4 reads-per-transcript (Figure. 6; top). Sample `SRR1265504` is the only sample for which a local minimal perplexity cannot be identified with respect to the range of hyperparameters scanned (Figure. 6; bottom). However, the perplexity plot for `SRR1265504` displays a knee-like behavior which suggests that after a certain VBEM prior size, larger VBEM prior sizes are no longer preferred – which is consistent across all perplexity plots and comparisons to ground truth.

These observations in the simulated data could suggest that perplexity may be an imperfect tool, or perhaps that different characteristics and read depths between the experimental and simulated data signal the need for a data-dependent selection mechanism for the smoothing function used to evaluate perplexity. Nonetheless, these observations do offer several insights as to how perplexity ought to be used in practice, especially when careful (albeit qualitative) inspection of perplexity plots reveal inconsistent preferences for hyperparameters across similar samples experiment-wide. First, perplexities may prefer abundance estimations smoother than ideal. In particular, when perplexities between two VBEM prior settings are close, or when perplexities are roughly minimized for a range of values, one ought to select the model that provides the sparsest estimates. Second, our experiments suggest that an optimal hyperparameter setting for a set of samples can be selected experiment-wide and perplexity plots can be used as a rough guide to select said hyperparamter setting. For example, visual inspection of perplexity plots (Figures A2 and A3) experiment-wide show a knee-like behavior and rough local minima for perplexity beginning at a VBEM prior size of 2 reads-per-transcript – the second best hyperparameter setting.

Thus, we note that perplexity can be used to quantitatively screen for bad abundance estimates (or the hyperparameters that generate them). The significance of this observation may be overlooked at first. However, to our knowledge, perplexity is the only metric that can differentiate between a satisfactory and a much more inaccurate abundance estimate when ground truth is absent.

Given the above, we also analyze the accuracy of differential transcript expression (DTE) analysis of estimates with the same VBEM prior size experiment-wide. We report AUROC of DTE calls up to a nominally useful maximum false discovery rate (FDR) of 0.05 (Figure 7). Not surprisingly, AUROC of DTE calls mirror the shape of Spearman correlations of estimates

infered from different VBEM prior sizes. Again, though minimizing perplexities does not exactly select the best estimates with regard to downstream DTE analysis, perplexity plots begin to exhibit plateaus or knee-like behaviors at VBEM prior size of 2 reads-per-transcript, the second best performing hyperparameter setting with regard to DTE (Figure 7).

5 Discussion

In this work, we derive the smoothed perplexity metric, which, to our knowledge, is the first metric that enables the evaluation of the quality of transcript abundance estimates in the absence of ground truth. Though we focus only on performing model selection with respect to one hyperparameter (the VBEM prior size) in `salmon`, model selection for other settings (e.g. choosing the number of bins for the range-factorized likelihood approximation, or selecting between VBEM and EM models and optimizations) are also certainly possible using perplexity.

In experimental data from the Sequencing Quality Control (SEQC) project [35], we show that the most accurate abundance estimates consistently have the lowest perplexity (lower is better) and demonstrate how quantitative model selection can be performed on input fragment sets directly and in the absence of ground truth. In simulated samples, we demonstrate a looser, but still useful, relationship between perplexity and measurements against ground truth. One possible explanation for the more erratic behavior and noisier perplexity plots for our simulated samples is due to these samples consisting of many fewer fragments than SEQC samples. On average, the simulated samples contain 17,410,732 fragments on average while the SEQC samples average 47,589,281 fragments.

Admittedly, the parameterization of the smoothing applied prior to input abundance estimates is somewhat unsatisfying. We do note, however, that at different settings of β, when a minima with regard to perplexity is observed in analyzed samples, the minima remains largely consistent – we demonstrate this for SEQC sample $A1$ in Figure A4. We plan to address the trade-offs and strategies for selecting smoothing strategies in future work.

Other directions for future work include utilizing perplexity or other metrics based on held-out likelihoods to not only select hyperparameters, but also to compare different abundance estimation models themselves. Furthermore, perplexity can also be adapted and applied to other problem settings in bioinformatics in which abundances are inferred from probabilistic models. For example, in metagenomics where model selection (i.e. choosing confidence cutoffs for taxa identification, or selecting candidate reference genomes) can have a large effect on abundance estimates [25].

In sum, this work demonstrates that evaluation of transcript abundance estimates in the absence of ground truth is possible, and presents a promising new direction in which estimated abundances are evaluated and validated directly on input fragment sets.

References

1 Simon Anders, Paul Theodor Pyl, and Wolfgang Huber. Htseq—a python framework to work with high-throughput sequencing data. *Bioinformatics*, 31(2):166–169, 2015.

2 Shawn C Baker, Steven R Bauer, Richard P Beyer, James D Brenton, Bud Bromley, John Burrill, et al. The External RNA Controls Consortium: a progress report. *Nature Methods*, 2(10):731–734, 2005. doi:10.1038/nmeth1005-731.

3 Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2016.

4 David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, 2003.

5 Elena Bushmanova, Dmitry Antipov, Alla Lapidus, and Andrey D Prjibelski. rnaSPAdes: a de novo transcriptome assembler and its application to RNA-Seq data. *GigaScience*, 8(9), September 2019. giz100. doi:10.1093/gigascience/giz100.

6 Scott C. Clark, Rob Egan, Peter I. Frazier, and Zhong Wang. ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, 29(4):435–443, 2013. doi:10.1093/bioinformatics/bts723.

7 Steffen Durinck, Paul T Spellman, Ewan Birney, and Wolfgang Huber. Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt. *Nature Protocols*, 4(8):1184–1191, 2009. doi:10.1038/nprot.2009.97.

8 Alyssa C. Frazee, Andrew E. Jaffe, Ben Langmead, and Jeffrey T. Leek. Polyester: simulating RNA-seq datasets with differential transcript expression. *Bioinformatics*, 31(17):2778–2784, April 2015. doi:10.1093/bioinformatics/btv272.

9 Peter Glaus, Antti Honkela, and Magnus Rattray. Identifying differentially expressed transcripts from RNA-seq data with biological variation. *Bioinformatics*, 28(13):1721–1728, 2012.

10 Manfred G Grabherr, Brian J Haas, Moran Yassour, Joshua Z Levin, Dawn A Thompson, Ido Amit, et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nature Biotechnology*, 29(7):644–652, 2011. doi:10.1038/nbt.1883.

11 James Hensman, Panagiotis Papastamoulis, Peter Glaus, Antti Honkela, and Magnus Rattray. Fast and accurate approximate inference of transcript expression from RNA-seq data. *Bioinformatics*, 31(24):3881–3889, August 2015. doi:10.1093/bioinformatics/btv483.

12 F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976. doi:10.1109/proc.1976.10159.

13 Hui Jiang and Wing Hung Wong. Statistical inferences for isoform expression in RNA-seq. *Bioinformatics*, 25(8):1026–1032, 2009.

14 Daniel C. Jones, Kavitha T. Kuppusamy, Nathan J. Palpant, Xinxia Peng, Charles E. Murry, Hannele Ruohola-Baker, and Walter L. Ruzzo. Isolator: accurate and stable analysis of isoform-level expression in rna-seq experiments. *bioRxiv*, 2016. doi:10.1101/088765.

15 Daniel C. Jones and Walter L. Ruzzo. Polee: RNA-Seq analysis using approximate likelihood. *bioRxiv*, 2020. doi:10.1101/2020.09.09.290411.

16 Woo Jin Kim, Jae Hyun Lim, Jae Seung Lee, Sang-Do Lee, Ju Han Kim, and Yeon-Mok Oh. Comprehensive analysis of transcriptome sequencing data in the lung tissues of copd subjects. *International Journal of Genomics*, 2015:206937, March 2015. doi:10.1155/2015/206937.

17 Bo Li and Colin N. Dewey. Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, August 2011. doi:10.1186/1471-2105-12-323.

18 Bo Li, Nathanael Fillmore, Yongsheng Bai, Mike Collins, James A Thomson, Ron Stewart, and Colin N Dewey. Evaluation of de novo transcriptome assemblies from RNA-Seq data. *Genome Biology*, 15(12):553, 2014. doi:10.1186/s13059-014-0553-5.

19 Yang Liao, Gordon K Smyth, and Wei Shi. featurecounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923–930, 2014.

20 Peng Liu, Rajendran Sanalkumar, Emery H Bresnick, Sündüz Keleş, and Colin N Dewey. Integrative analysis with chip-seq advances the limits of transcript quantification from rna-seq. *Genome research*, 26(8):1124–1133, 2016.

21 Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, et al. Sustainable data analysis with Snakemake. *F1000Research*, 10:33, 2021. doi:10.12688/f1000research.29032.1.

22 Naoki Nariai, Osamu Hirose, Kaname Kojima, and Masao Nagasaki. TIGAR: transcript isoform abundance estimation method with gapped alignment of RNA-Seq data by variational Bayesian inference. *Bioinformatics*, 29(18):2292–2299, July 2013. doi:10.1093/bioinformatics/btt381.

23 Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yosuke Kawai, and Masao Nagasaki. A bayesian approach for estimating allele-specific expression from RNA-seq data with diploid genomes. In *BMC genomics*, volume 17(1), pages 7–17. BioMed Central, 2016.

24 Naoki Nariai, Kaname Kojima, Takahiro Mimori, Yukuto Sato, Yosuke Kawai, Yumi Yamaguchi-Kabata, and Masao Nagasaki. Tigar2: sensitive and accurate estimation of transcript isoform expression with longer RNA-seq reads. *BMC genomics*, 15(10):1–9, 2014.

25 Daniel J. Nasko, Sergey Koren, Adam M. Phillippy, and Todd J. Treangen. RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology*, 2018. doi:10.1186/s13059-018-1554-6.

26 Rob Patro, Geet Duggal, Michael I. Love, Rafael A. Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, April 2017. doi:10.1038/nmeth.4197.

27 Atif Rahman and Lior Pachter. CGAL: computing genome assembly likelihoods. *Genome Biology*, 14(1):R8, 2013. doi:10.1186/gb-2013-14-1-r8.

28 Johannes Rainer. *EnsDb.Hsapiens.v86: Ensembl based annotation package*, 2017. R package version 2.99.0.

29 Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. doi:10.1016/0377-0427(87)90125-7.

30 Migun Shakya, Chien-Chi Lo, and Patrick S. G. Chain. Advances and challenges in metatranscriptomic analysis. *Frontiers in Genetics*, 10:904, 2019. doi:10.3389/fgene.2019.00904.

31 Leming Shi, Laura H Reid, Wendell D Jones, Richard Shippy, Janet A Warrington, et al. The MicroArray Quality Control (MAQC) project shows inter- and intraplatform reproducibility of gene expression measurements. *Nature Biotechnology*, 24(9):1151–1161, 2006. doi:10.1038/nbt1239.

32 Richard Smith-Unna, Chris Boursnell, Rob Patro, Julian M. Hibberd, and Steven Kelly. TransRate: reference-free quality assessment of de novo transcriptome assemblies. *Genome Research*, 26(8):1134–1144, 2016. doi:10.1101/gr.196469.115.

33 Charlotte Soneson, Michael I. Love, and Mark D. Robinson. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Research*, 4, 2015. doi:10.12688/f1000research.7563.1.

34 Avi Srivastava, Laraib Malik, Hirak Sarkar, and Rob Patro. A Bayesian framework for inter-cellular information sharing improves dscRNA-seq quantification. *Bioinformatics*, 36(Supplement_1):i292–i299, 2020.

35 Zhenqiang Su, Paweł P Łabaj, Sheng Li, Jean Thierry-Mieg, Danielle Thierry-Mieg, Wei Shi, et al. A comprehensive assessment of RNA-seq accuracy, reproducibility and information content by the Sequencing Quality Control Consortium. *Nature Biotechnology*, 32(9):903–914, 2014. doi:10.1038/nbt.2957.

36 Ernest Turro, Shu-Yi Su, Ângela Gonçalves, Lachlan JM Coin, Sylvia Richardson, and Alex Lewin. Haplotype and isoform specific expression estimation using multi-mapping RNA-seq reads. *Genome biology*, 12(2):1–15, 2011.

37 Andrew D Yates, Premanand Achuthan, Wasiu Akanni, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, et al. Ensembl 2020. *Nucleic Acids Research*, 48(D1):D682–D688, November 2019. doi:10.1093/nar/gkz966.

38 Mohsen Zakeri, Avi Srivastava, Fatemeh Almodaresi, and Rob Patro. Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics*, 33(14):i142–i151, July 2017. doi:10.1093/bioinformatics/btx262.

39 Anqi Zhu, Avi Srivastava, Joseph G Ibrahim, Rob Patro, and Michael I Love. Nonparametric expression analysis using inferential replicate counts. *Nucleic Acids Research*, 47(18):e105–e105, 2019. doi:10.1093/nar/gkz622.

A Appendix

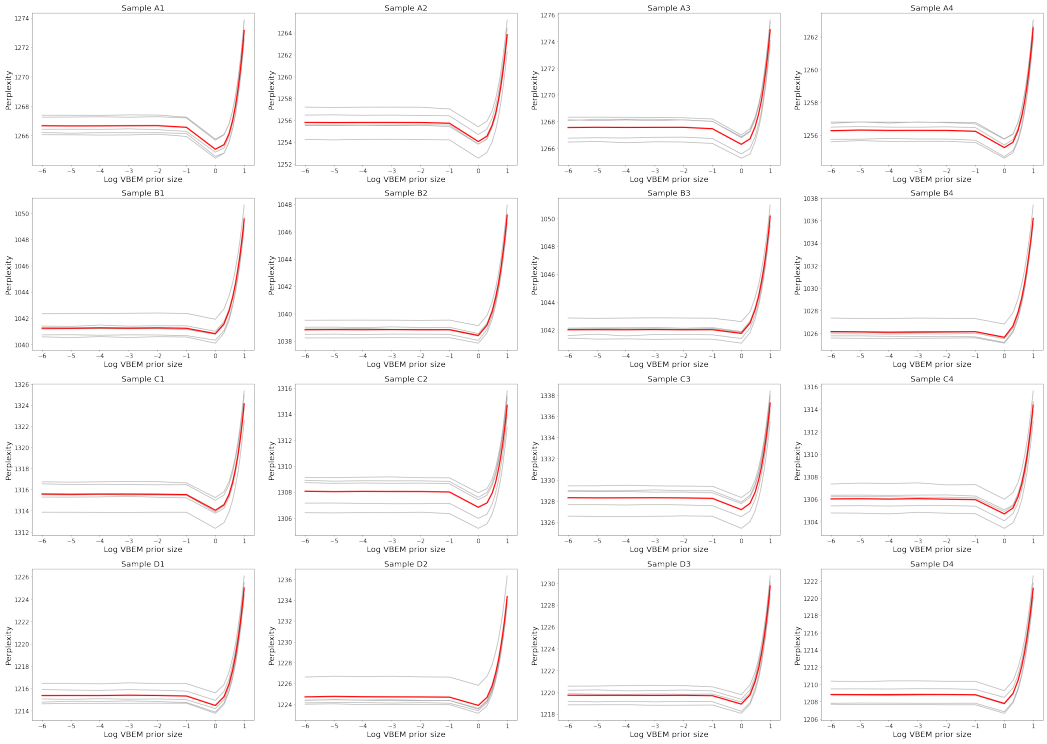


Figure A1 Perplexity plots. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

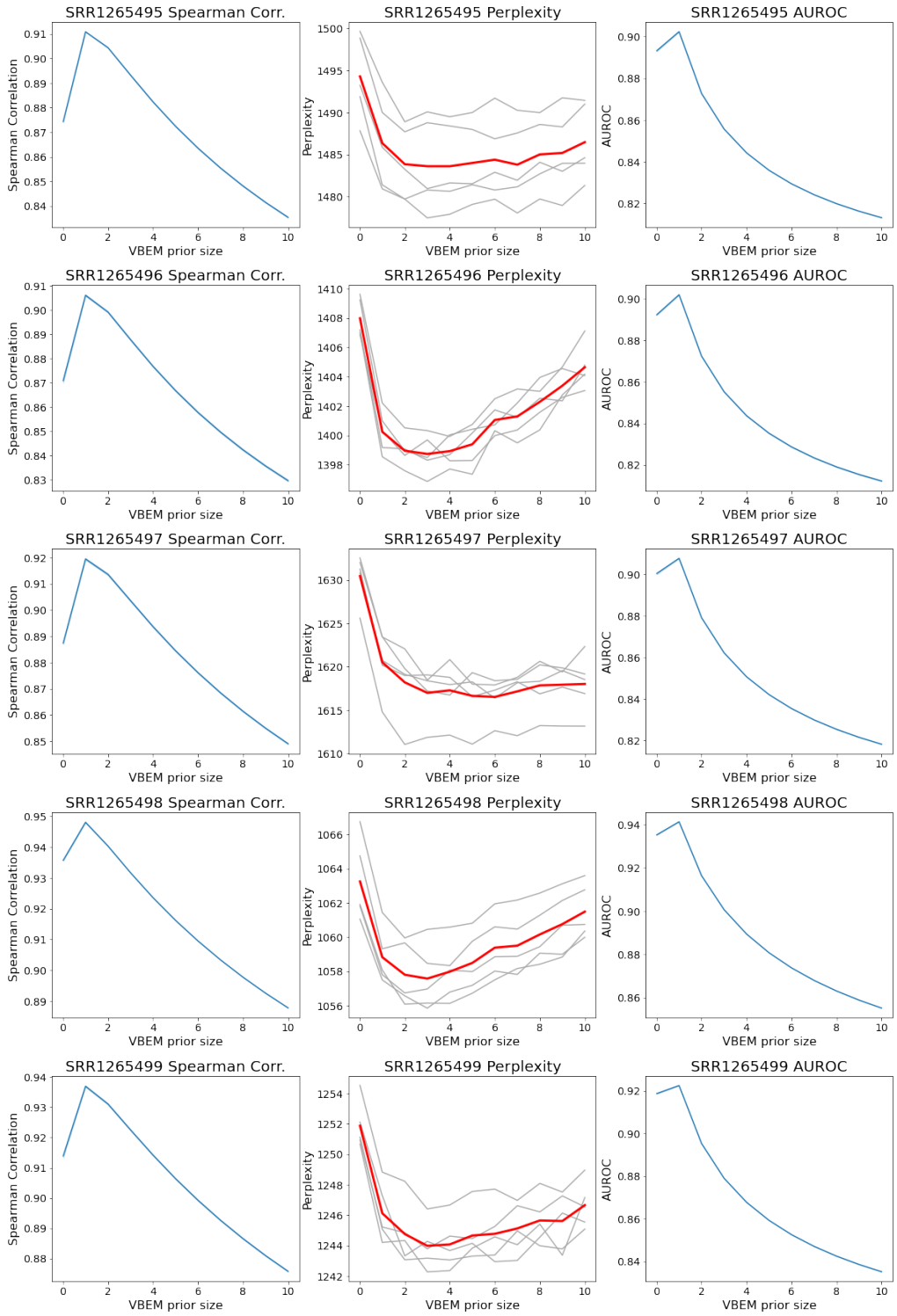


Figure A2 Quality of transcript abundance estimates as a function of VBEM per-nucleotide prior size for samples `SRR1265{495-499}`. (Left column) Spearman Correlation with respect ground truth expressed transcripts. (Middle column) Perplexity of abundance estimates; perplexities per-fold indicated in gray and mean perplexities in red. (Right column) AUROC for retrieving ground truth unexpressed transcripts. Leftmost plotted points for all plots use default `salmon` VBEM prior size of 0.01 reads-per-transcript.

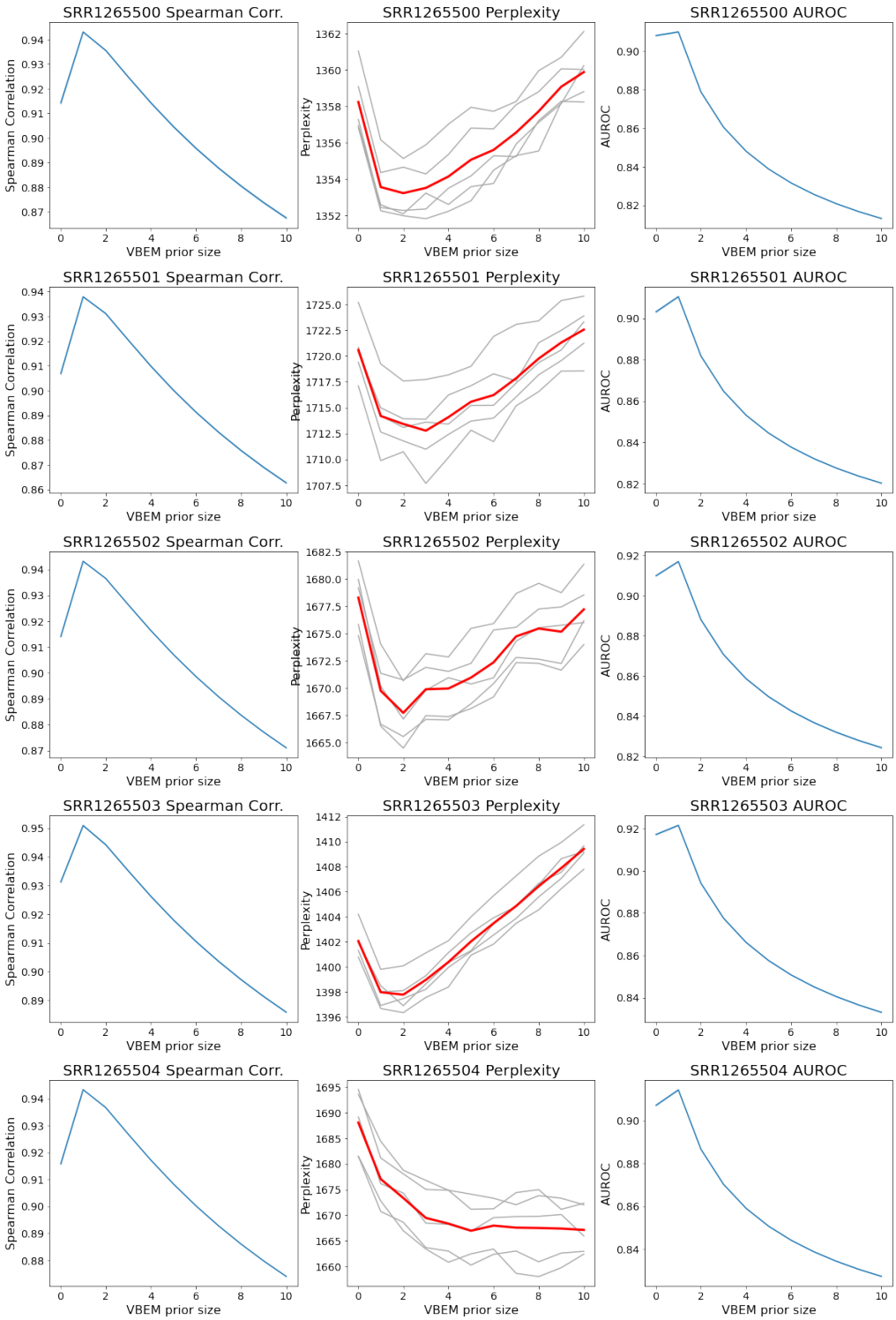


Figure A3 Quality of transcript abundance estimates as a function of VBEM per-nucleotide prior size for samples `SRR1265{500-504}`.

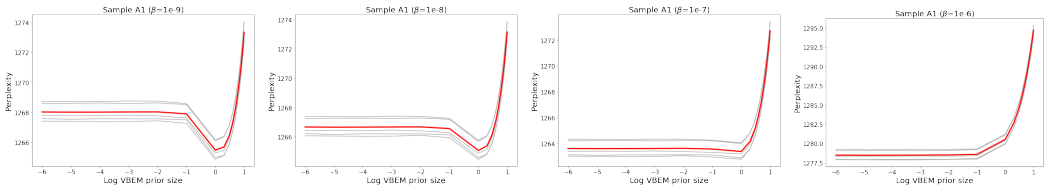


Figure A4 Perplexity plots for SEQC sample $A1$ at different smoothing parameter settings. Plots show perplexity versus VBEM reads-per-transcript prior size for SEQC samples. Mean perplexities across five folds are plotted in red, and gray perplexities for each fold are plotted are plotted in gray.

The Maximum Duo-Preservation String Mapping Problem with Bounded Alphabet*

Nicolas Boria
Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Laurent Gourvès
Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Vangelis Th. Paschos
Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Jérôme Monnot
Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Abstract

Given two strings A and B such that B is a permutation of A, the MAX DUO-PRESERVATION STRING MAPPING (MPSM) problem asks to find a mapping π between them so as to preserve a maximum number of duos. A duo is any pair of consecutive characters in a string and it is preserved by π if its two consecutive characters in A are mapped to same two consecutive characters in B. This problem has received a growing attention in recent years, partly as an alternative way to produce approximation algorithms for its minimization counterpart, MIN COMMON STRING PARTITION, a widely studied problem due its applications in comparative genomics. Considering this favored field of application with short alphabet, it is surprising that MPSM$^\ell$, the variant of MPSM with bounded alphabet, has received so little attention, with a single yet impressive work that provides a 2.67-approximation achieved in $O(n)$ [5], where $n = |A| = |B|$. Our work focuses on MPSM$^\ell$, and our main contribution is the demonstration that this problem admits a Polynomial Time Approximation Scheme (PTAS) when $\ell = O(1)$. We also provide an alternate, somewhat simpler, proof of NP-hardness for this problem compared with the NP-hardness proof presented in [16].

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Dynamic programming; Theory of computation → Pattern matching; Theory of computation → Complexity classes

Keywords and phrases Maximum-Duo Preservation String Mapping, Bounded alphabet, Polynomial Time Approximation Scheme

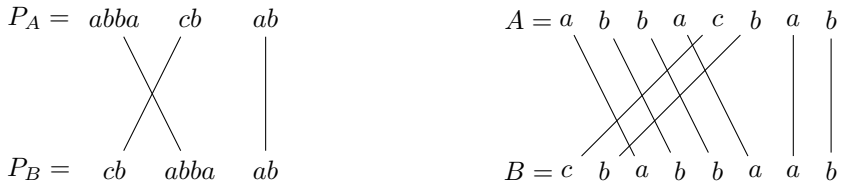
Digital Object Identifier 10.4230/LIPIcs.WABI.2021.5

Funding *Nicolas Boria*: Supported by Agence Nationale de la Recherche (ANR), project STAP ANR-17-CE23-0021.
Laurent Gourvès: Supported by Agence Nationale de la Recherche (ANR), project THEMIS ANR-20-CE23-0018

1 Introduction

Evaluating the similarity between strings is a core problem of stringology, with various applications ranging from data compression to bioinformatics. Various distance measures on string have been proposed such as the Hamming distance (see [23] for a full survey), the Jaro-Winkler distance [28], the overlap coefficient [27], etc. However, in real life applications, many of these simple distance measures fail to provide significant information, and the

* This work is dedicated to the memory of our colleague and friend Jérôme Monnot who untimely passed away.



(a) A solution expressed as blocks. **(b)** The same solution expressed as mapping.

Figure 1 An instance of MCSP along with its solution.

computation of more involved distances is often required. One of the most commonly used distance measure between two strings is the so called *edit distance* [18]. Given a set of allowed operations on strings, and a cost function on the allowed operations, the edit distance measures the minimum overall cost of edit operations that need to be performed to transform the first string into the second. The edit distance is a versatile concept that applies to various complex objects apart from strings, and allows for the representation of many different problems by considering different sets of allowed edit operations and different cost functions. In data compression, the edit distance can be used to help store efficiently a couple or a set of similar yet different data (e.g., different versions of the same object), by encoding a base element only, and then representing each other element of the dataset as the series of edit operations that results in it starting from the base element [25]. In bioinformatics, the edit distance provides some measure of kinship between species by measuring the similarity of their DNA [17, 24].

When the only allowed edit operation is shifting a block of characters within the string, computing the edit distance between two strings amounts to solving MIN COMMON STRING PARTITION problem. The MIN COMMON STRING PARTITION (MCSP) is a fundamental problem in the field of string comparison [9, 15], and can be applied more specifically to genome rearrangement problems, as shown in [26], where each block of the partition can be seen as a gene. Consider two strings A and B, both of length n, such that B is a permutation of A. MCSP asks for a partition $\mathcal{P}_A$ of A and a partition $\mathcal{P}_B$ of B, both of minimum cardinality $|\mathcal{P}_A| = |\mathcal{P}_B| = p$, such that $\mathcal{P}_A$ is a permutation of $\mathcal{P}_B$. As B can be reconstructed by shifting $p-1$ blocks of $\mathcal{P}_A$, the edit distance between A and B is equal to $p-1$. Figure 1 provides a visual representation of the problem with $A = abbacbab$ and $B = cbabbaab$.

By focusing on how the blocks of $\mathcal{P}$ are mapped between A and B, the problem can be alternatively defined as follows: given two strings A and B, both of length n, such that B is a permutation of A, one needs to define a mapping π that maps each position i of A to a position $\pi(i)$ of B such that, for all $i \in \{1, \ldots, n\}$, the letter $A[i]$ is the same as the letter $B[\pi(i)]$. The number $p-1$ of cuts in a partition with p blocks is then equal to the number of consecutive positions, or *duos*, $(i, i+1)$ such that $\pi(i)+1 \neq \pi(i+1)$ (in Figure 1b, this happens for $i = 4$ and $i = 6$). Hence, maximizing the number of duos that are *preserved* by the mapping, i.e., such that $\pi(i)+1 = \pi(i+1)$ immediately yields a solution for MCSP. This gives rise to the maximization version of the MSCP problem, called MAX DUO-PRESERVATION STRING MAPPING (MPSM):

▶ **Definition 1.** *MAX DUO-PRESERVATION STRING MAPPING*
Given two strings A and B of length n such that A is a permutation of B, the MAX DUO-PRESERVATION STRING MAPPING problem (MPSM) asks for a bijective mapping π between positions of A and B such that:

- $\forall i \in \{1, \ldots, n\}$, $A[i] = B[\pi(i)]$
- *the number of duos preserved by π is maximum*

Related works

k-MCSP denotes the restricted version of MCSP where each character occurs at most k times. This problem has been shown NP-hard and APX-hard even with $k = 2$ [15]. Several approximations are known for low values of k [9, 10, 11, 15, 19, 20], and the problem was also studied extensively in terms of parameterized algorithms [6, 7, 12, 16]. Another variant of MCSP, denoted by MCSP$^\ell$, deals with the version where the alphabet used to form the strings consists of at most ℓ characters. This version of the problem has been proved to be NP-hard [16] for any $\ell \geq 2$. The best approximation ratio known so far for the general version is $O(\log n \log^* n)$ [11].

In order to tackle the approximation issue from a different angle, the maximization version of the problem (as described in Definition 1) was introduced in [8], with an $O(k^2)$ approximation for $k-$MPSM (where, similarly to $k-$MCSP, each character occurs at most k times). The problem has be shown APX-hard in [3] even with $k = 2$, and the article also provided the first constant approximation for the general problem with a simple 4-approximation algorithm. This ratio was later improved to 3.5 using a local search technique [2], 3.25 through a combinatorial triplet matching approach [4], and finally $2 + \varepsilon$ in $n^{O(1/\varepsilon)}$ with a combination of a greedy algorithm and local search [13]. Note that [13] also presented a 2.67-approximation algorithm with time complexity $O(n^2)$. Moreover a $1.4+\varepsilon$ approximation algorithm for $2-$MCSP appears in [29]. Regarding the version of MCSP where the alphabet has at most ℓ characters (MCSP$^\ell$), a recent work proposed a 2.67-approximation algorithm that requires time as low as $O(n + \ell^7)$ [5]. The method presented in [5] also guarantees a 2.67-approximation in $O(n^3)$ for the weighted version of the problem which was introduced in [22]. The weighted version takes into consideration the positions of the preserved duos (the closer the better). Finally, the problem was also studied through the prism of fixed-parameter tractability, and was shown to be FPT with respect to the number of preserved duos in [1], whereas [21] presented more efficient algorithms still parameterized with respect to the solution size.

Our Contribution

In this work, we tackle the MCSP$^\ell$ problem where the alphabet used to form strings A and B has at most $\ell = O(1)$ characters. The problem is NP-hard even with $\ell = 2$, as its minimization counterpart has be shown to be NP-hard in [16]. We do however provide a more direct reduction in Section 4. The main contribution of this article consists of the proof that MCSP$^\ell$ admits a Polynomial Time Approximation Scheme (PTAS), as we provide an algorithm based on dynamic programming that guarantees, for every fixed integer $k > 1$, a $(1 + \frac{1}{k-1})$-approximation within time complexity $O(kn^{1+c\ell^k})$, where c is some constant, which amounts to $O(n^{O(1)})$ provided that both k and ℓ are constant. The algorithm and its approximation analysis are presented in Section 3.

We remind that a PTAS is a family of approximation algorithms. For any fixed $\varepsilon > 0$, a PTAS for a maximization problem runs in time polynomial in the instance size and outputs a feasible solution of value SOL such that $(1 + \varepsilon)SOL \geq OPT$ where OPT denotes the optimal value achieved by a feasible solution. A *fully* polynomial time approximation scheme (FPTAS) also satisfies $(1 + \varepsilon)SOL \geq OPT$ but its running time is polynomial both in the instance size and $1/\varepsilon$.

Since the objective value of MCSP$^\ell$ is upper bounded by a polynomial of the instance size (indeed, at most $n - 1$ duos can be preserved), the existence of an FPTAS for MCSP$^\ell$ would lead to the unlikely fact that P=NP.

2 Preliminaries

Let A and B represent two strings formed on an alphabet $\mathcal{L}$, such that B is a permutation of A. Let $n = |A| = |B|$. We denote by ℓ the size of the alphabet $\mathcal{L}$.

A *word* w on alphabet $\mathcal{L}$ is a non empty tuple of letters from $\mathcal{L}$. The i-th letter of w is denoted by $w[i]$ (similar to the fact that the i-th letter of A is denoted by $A[i]$). $\mathcal{D}_k(\mathcal{L})$ is the complete dictionary of words on $\mathcal{L}$ with at most k letters (i.e., the word size is at most k). Note that $|\mathcal{D}_k(\mathcal{L})| = \sum_{i=1}^{k} \ell^i = O(\ell^k)$. When the context is clear, the dictionary is denoted by $\mathcal{D}$.

Cardinality assignments X and $[w]$

In what follows, X denotes a vector in $\mathbb{N}^{|\mathcal{D}_k(\mathcal{L})|}$ called *cardinality assignment*, that assigns a cardinality[1] to every word of $\mathcal{D}_k(\mathcal{L})$. $X(w)$ denotes the cardinality assigned to word w in X. $[w]$ denotes the cardinality assignment that assigns 1 to w and 0 to every other word.

Regular operations on vectors are allowed on cardinality assignments. In particular, the cardinality assignment $X + [w]$ (resp., $X - [w]$) is identical to X except that position w is increased (resp., decreased) by one unit.

Finally, we denote by X_0 the cardinality assignment that assigns 0 to every word.

String description and String-cuts

We call *string-cut* a partitioning S of a string A in sub-strings. Concretely, a string-cut is an ordered list of words whose concatenation results in A. We say that a cardinality assignment X *describes* a string-cut S if words in S appear with the exact frequencies described by X. Similarly, we say that a cardinality assignment X *describes* a string A if there exists a string-cut S of A such that X describes S.

A string-cut is said to be k-*bounded* if its words have length at most k. We denote by S_t the t-th word of a string-cut S, while the operation $S :: w$ consists of appending word w to the end of S.

Example

Consider the string $A = abbabbaab$ formed on the alphabet $\mathcal{L} = \{a, b\}$. The dictionary $\mathcal{D}_2(\mathcal{L})$ contains all possible words of length at most two using letters of alphabet $\mathcal{L}$. Namely, $\mathcal{D}_2(\mathcal{L}) = \{a, b, aa, bb, ab, ba\}$. The string A can be partitioned in the following 2-bounded string-cut $S = (ab, b, ab, ba, ab)$ described by the cardinality assignment $X = (0, 1, 0, 0, 3, 1)$ (because S has zero a, one b, zero aa, zero bb, three ab, and one ba). Hence, it holds that the cardinality assignment X describes A.

Cardinality assignment evaluation

For any cardinality assignment X, we introduce the function $\texttt{eval}(X)$, which returns the number of duos of characters *within the words of the cardinality assignment*, namely:

$$\texttt{eval}(X) = \sum_{w \in \mathcal{D}} (|w| - 1) \cdot X(w).$$

[1] How many times the word occurs.

When two string-cuts S_A and S_B are described by the same cardinality assignment X (i.e., S_A is a permutation of S_B), any mapping between words of S_A of S_B translates in a mapping between letters of A and letters of B that preserves *at least* $\texttt{eval}(X)$ duos. There are possibly more preserved duos if consecutive words in S_A are mapped to consecutive words in S_B as shown in Figure 2, where two strings and A and B are indeed represented by two string-cuts that are described by the same cardinality assignment. On the one hand, the mapping in Figure 2a preserves only duos that are within words of the string-cuts, and hence it saves 4 duos. On the other hand, the mapping in Figure 2b maps two consecutive words in S_A to two consecutive words in S_B so that an extra duo bb is preserved.

(a) A mapping that preserves $\texttt{eval}(X) = 4$ duos. **(b)** A mapping that preserves $\texttt{eval}(X)+1 = 5$ duos.

Figure 2 Two mappings between S_A and S_B with $A = abbabbaab$ and $B = ababbabab$.

Also, notice that given two string-cuts on A and B that are described by the same cardinality assignment X, a mapping between words of the string-cuts can be easily constructed: map the i-th occurrence of each word in S_A to its i-th occurrence in S_B. Since any mapping of words preserves at least $\texttt{eval}(X)$ duos, our method aims at computing two string cuts S_A and S_B that are both described by some cardinality assignment X, such that $\texttt{eval}(X)$ is a lower bound on the number of duos preserved by some mapping which may be reconstructed by arbitrarily mapping words of S_A to words of S_B. We now move on to the description of our algorithm.

3 A Polynomial Time Approximation Scheme for MPSM$^\ell$

In the following, we describe a method based on dynamic programming which results in a polynomial time approximation scheme (PTAS) for MPSM$^\ell$.

For the sake of clarity, the algorithm is presented in a two-fold fashion. In Section 3.1 we describe and analyze a procedure called $\texttt{GENERATE}(A, k)$ which, given a string A and an integer k, uses dynamic programming to generate an exhaustive collection of possible cardinality assignments that describe some k-bounded string-cut of A. One important feature that ensures the polynomial complexity of our method is the following: although the number of different k-bounded string-cuts of A and B grows exponentially with n, the number of different cardinality assignments describing these string-cuts is upper bounded by a polynomial in n provided that both k and ℓ are constant. In Section 3.2, we show how to use the data structure created by the procedure $\texttt{GENERATE}(A, k)$ to find the best possible cardinality assignment that describes both A and B, and to generate a matching between characters of A and B that saves the maximum number of duos within k-bounded string-cuts of A and B. Finally, in Section 3.3, we show how the method yields a PTAS for MSPM$^\ell$.

3.1 Algorithm GENERATE(A, k)

Algorithm 1 GENERATE(A, k).

Require: a string A and a positive integer k
Ensure: a couple $(\Omega, \mathcal{S})$ such that
- Ω is a set of cardinality assignments such that any k-bounded string-cut of A is described by some cardinality assignment $X \in \Omega$
- $\mathcal{S}$ is a dictionary that assigns a single string-cut $\mathcal{S}(X)$ to every cardinality assignment $X \in \Omega$

```
1:  Ω_0 ← {X_0}
2:  S(X_0) = ∅
3:  for i = 1 to n do
4:    Ω_i ← ∅
5:    for each j ∈ [1, min(i, k)] do
6:      w ← (A[i − j + 1], ..., A[i])
7:      for each X ∈ Ω_{i−j} do
8:        if (X + [w]) ∉ Ω_i then
9:          Ω_i ← Ω_i ∪ (X + [w])
10:         S(X + [w]) ← S(X) :: w
11:       end if
12:     end for
13:   end for
14: end for
15: Ω ← Ω_n
16: return (Ω, S)
```

Algorithm 1 is based on dynamic programming and generates a collection of cardinality assignments Ω, as well as a dictionary $\mathcal{S}$ of corresponding string-cuts, such that for each $X \in \Omega$, there exists a single string-cut $\mathcal{S}(X) \in \mathcal{S}$ that describes X. When building Ω_i, the algorithm considers words w of length $j \leq k$ which are sub-strings of A and whose last character is $A[i]$. Then, it appends w onto a string-cut $\mathcal{S}(X)$ such that $X \in \Omega_{i-j}$, and store the resulting string-cut in $\mathcal{S}$.

Let us describe the data structures that are created throughout Algorithm 1. GENERATE(A, k) aims at computing:

- a complete collection Ω containing all possible cardinality assignments describing some k-bounded string-cut of A,
- a corresponding dictionary of string-cuts $\mathcal{S}$, such that for all $X \in \Omega$, there exists a unique string -cut in $\mathcal{S}$ described by X. $\mathcal{S}(X)$ denotes the unique string-cut in $\mathcal{S}$ described by X.

The first property of $(\Omega, \mathcal{S})$ is proved in the following (Proposition 2). The second property is ensured by Lines 8-10: a single entry is added to the dictionary $\mathcal{S}$ (Line 10) only when a new cardinality assignment is added to some set Ω_i (Line 9), while Line 8 ensures that there is no duplicate entries within the Ω_i's.

Let us now prove that the collection Ω produced by Algorithm 1 contains all the cardinality assignments necessary to describe any k-bounded string-cut of A.

▶ **Proposition 2.** *For any k-bounded string-cut S of A, it holds that there exists $X \in \Omega$ that describes S.*

Proof. We demonstrate the following claim by induction on i.

▷ Claim 3. Any k-bounded string-cut S of the sub-string $A_i = (A[1], \ldots, A[i])$ is described by some cardinality assignment X of Ω_i.

The claim holds trivially for $i = 0$ and $i = 1$, as one can verify that after the first iteration, we have $\Omega_1 = \{A[1]\}$. In other words, Ω_1 contains a single cardinality assignment, the one that describes a single word with a single letter (the first letter of A).

Now, let us suppose that Claim 3 is verified for all $j < i$ and prove that it is verified also for i.

Let S be a k-bounded string-cut of A_i, let X' denote the cardinality assignment that describes S, and let $\tilde{w}$ with length $|\tilde{w}|$ denote the last word of S. We need to prove that X' belongs to Ω_i by the end of the algorithm.

By hypothesis, the claim holds for $j = i - |\tilde{w}|$. In other words, there is a cardinality assignment $\tilde{X} \in \Omega_{i-|\tilde{w}|}$ that describes the string-cut $S \setminus \tilde{w}$ of $A_{i-|\tilde{w}|}$. Moreover, $\tilde{X}$ is equal to $X' - [\tilde{w}]$ as the string cut $\tilde{X}$ corresponds to the string-cut X' with one less occurrence of word $\tilde{w}$.

Consider the i^{th} iteration of the outer loop starting at Line 3 of Algorithm 1 and the inner loop (Line 5) where $j = |\tilde{w}|$ (Recall that S is k-bounded so $|\tilde{w}| \leq k$). By induction hypothesis, it holds that $\tilde{X}$ belongs to $\Omega_{i-|\tilde{w}|}$, so there will be an iteration of the innermost loop (Line 7) with $X = \tilde{X}$: at this point of the algorithm, it will be checked if the cardinality assignment $\tilde{X} + [w] = X'$ already belongs to Ω_i, and will include it to Ω_i if it is not the case.

Hence, by the end of the algorithm, it holds that $X' \in \Omega_i$, and the proof is complete. ◀

Regarding the complexity of Algorithm 1, let us stress that no set Ω_i has any duplicate entries (ensured by the *if* condition at Line 8), so that its cardinality is bounded by the number of different possible cardinality assignments describing any k-bounded string-cut over a string of length i. Recall that such cardinality assignment is a vector with $|\mathcal{D}_k(\mathcal{L})|$ coordinates, each coordinate describing the frequency of a word in a string-cut. Roughly, each coordinate of a cardinality assignment of Ω_i is upper bounded by i, so that there are at most $i^{|\mathcal{D}_k(\mathcal{L})|}$ different cardinality assignments, i.e., $|\Omega_i| \leq i^{|\mathcal{D}_k(\mathcal{L})|}$. Moreover, let us remind that there exists a constant c such that $|\mathcal{D}_k(\mathcal{L})| \leq c \cdot |\mathcal{L}|^k$, and thus $|\Omega_i| \leq i^{c \cdot |\mathcal{L}|^k}$.

Hence, the total number of innermost loops that are made at the i^{th} iteration of the outer loop is at most $k|\Omega_{i-1}|$. The overall complexity is thus $\sum_{i=1}^{n} k|\Omega_i|$ which is bounded by $k \sum_{i=1}^{n} i^{c \cdot |\mathcal{L}|^k} = O(n \cdot kn^{c \cdot |\mathcal{L}|^k}) = O(kn^{c \cdot |\mathcal{L}|^k + 1})$.

3.2 Algorithm MATCH(A, B, k)

We now devise the following algorithm, called MATCH (see Algorithm 2), parameterized by k for MPSM, which first runs Algorithm 1 on A and B in order to compute the couples $(\Omega_A, \mathcal{S}_A)$ and $(\Omega_B, \mathcal{S}_B)$.

Algorithm 2 consists of identifying the cardinality assignment X^{SOL} that belongs to both Ω_A and Ω_B and which contains the largest number of duos. Along with the identification of X^{SOL} comes the identification of two string-cuts (namely, S_A^{SOL} and S_B^{SOL} for A and B, respectively) which are both described by X^{SOL}.

The complexity of the loop in MATCH-$\Omega(k)$ can be brought down to $O(|\Omega_A|)$ with a proper data structure, that is, a structure that ensures a fast (linear time) verification of the condition expressed at Line 5. Hence the overall complexity is given by the generation of couples $(\Omega_A, \mathcal{S}_A)$ and $(\Omega_B, \mathcal{S}_B)$, namely $O(kn^{1+c \cdot |\mathcal{L}|^k})$.

Algorithm 2 $\texttt{MATCH}(A, B, k)$.

Initialize $S_A^{SOL} = \emptyset, S_A^{SOL} = \emptyset, X^{SOL} = X_0$
$(\Omega_A, \mathcal{S}_A) \leftarrow \texttt{GENERATE}(A, k)$
$(\Omega_B, \mathcal{S}_B) \leftarrow \texttt{GENERATE}(B, k)$
for each $X \in \Omega_A$ **do**
  if $X \in \Omega_B$ **and** $\texttt{eval}(X) > \texttt{eval}(X^{SOL})$ **then**
    $X^{SOL} \leftarrow X$
  end if
end for
$S_A^{SOL} \leftarrow \mathcal{S}_A(X^{SOL})$
$S_B^{SOL} \leftarrow \mathcal{S}_B(X^{SOL})$
return (S_A^{SOL}, S_B^{SOL})

As mentioned at the end of Section 2, a mapping between letters of A of B that preserves at least $\texttt{eval}(X^{SOL})$ duos can be easily derived from the couple (S_A^{SOL}, S_B^{SOL}) computed by Algorithm 2. Now it remains to make the link between $\texttt{eval}(X^{SOL})$ and the number of duos preserved by an optimal solution.

3.3 Approximation Analysis

We now prove that $\texttt{MATCH}(A, B, k)$ yields a Polynomial Time Approximation Scheme for MPSM$^\ell$.

First, we need to provide some lower bound on the number of duos that are preserved by our solution.

▶ **Proposition 4.** *Let S_A^{SOL} and S_B^{SOL} be the string-cuts generated by $\texttt{MATCH}(A, B, k)$. Both are described by the same cardinality assignment X^{SOL}.*

Given any couple of k-bounded string-cuts $\tilde{S}_A$ on A and $\tilde{S}_B$ on B that are both described by the same cardinality assignment $\tilde{X}$, it holds that $\texttt{eval}(X^{SOL}) \geq \texttt{eval}(\tilde{X})$.

Proof. From Proposition 2, we can assert that $\tilde{X} \in \Omega_A$ and $\tilde{X} \in \Omega_B$. Hence, in the first part of the $\texttt{MATCH}(A, B, k)$, the cardinality assignment $\tilde{X}$ will be considered as a candidate. Eventually, the algorithm retains X^{SOL} that yields the best solution among all candidates considered, including $\tilde{X}$. ◀

Consider now an optimal solution π^* for MPSM$^\ell$ that preserves OPT duos, and an approximate solution π returned by Algorithm 2 that preserves $SOL \geq \texttt{eval}(X^{SOL})$ duos. As stated in Section 2, π^* induces string-cuts S_A^* and S_B^* of A and B, such that S_A^* is a permutation of S_B^*. These string-cuts may contain words that are strictly longer than k. Consider the string-cuts S'_A and S'_B that result from cutting all such words into slices of length at most k in both S_A^* and S_B^*, such that S'_A and S'_B are both described by the same cardinality assignment, say X'. No more than OPT/k additional cuts are added this way. It holds that:

- S'_A and S'_B are k-bounded, and they are described by the same cardinality assignment X'. Hence, by applying Proposition 2, it holds that $SOL \geq \texttt{eval}(X')$.
- The optimal mapping π^* preserves $\texttt{eval}(X')$ duos within words of X' and at most OPT/k duos between words of X', one for each added cut. Hence, it holds that $OPT \leq \texttt{eval}(X') + OPT/k$.

Combining these two facts, one immediately derives the following approximation between OPT and SOL:

$$\frac{OPT}{SOL} \leq 1 + \frac{1}{k-1}, \quad \forall k > 1.$$

By fixing $k = \lceil 1/\varepsilon + 1 \rceil$, we can then assert that our algorithm guarantees a $(1+\varepsilon)$-approximation within time complexity $O(n^{O(1)})$ provided that both k and ℓ are bounded by some constant. Hence, our final result holds:

▶ **Proposition 5.** *MPSM$^\ell$ with $\ell = O(1)$ admits a PTAS.*

4 Hardness result

The fact that MCSP^2 is NP-complete is known from [16, Theorem 1]. However, we give an alternate, somewhat simpler, proof.

▶ **Proposition 6** (Alternate Proof). *MPSM$^\ell$ and MCSP$^\ell$ are both NP-hard, even if $\ell = 2$.*

Proof. Consider an instance I of the NP-complete problem 3-PARTITION [14], with a set of integers $X = \{x_1, ..., x_n\}$ such that $n = 3m$, and let $m\Sigma = \sum_{x_i \in X} x_i$. As standard hypothesis for 3-PARTITION, suppose $\Sigma/4 < x_i < \Sigma/2$ holds for all x_i. The question is whether X can be partitioned in m triplets, each of total sum Σ.

We build an instance $J(I)$ of MPSM$^\ell$ with $\ell = 2$, consisting of two strings A and B built in the following way.

- For each integer x_i in X, we build a string A_i that consists of one b followed by $x_i + 1$ consecutive a's. We also build a sub-string A_{n+1} that contains a single letter b. String A is the concatenation of all strings A_i's.
- Consider the substring B_i that consists of one b followed by $\Sigma + 3$ letters a, followed by 2 letters b. String B results from the concatenation of m substrings B_i, with a single additional b at the very start. Note that all B_i's are identical, we merely index them by i to simplify the proof.

One can easily verify that both strings A and B contain exactly $m\Sigma + n$ occurrences of letter a and $n + 1$ occurrences of letter b, so $J(I)$ is a valid instance of MCSP^2.

We are going to show that instance I of 3-PARTITION admits a solution if and only if instance $J(I)$ of MPSM^2 admits a solution that preserves $m\Sigma + 2m$ duos.
($\Rightarrow$) First, note that in string A, there are exactly n ba duos, n ab duos, $m\Sigma$ aa duos, and no bb duo. On the other hand, string B has m ab duos, m ba duos, $m\Sigma + 2m$ aa duos, and $2m$ bb duos.

Thus, any solution will preserve at most $m\Sigma + 2m$ duos in total, that is, $\min\{m\Sigma, m\Sigma + 2m\} = m\Sigma$ duos of type aa, $\min\{m, n\} = m$ duos of type ab, $\min\{0, 2m\} = 0$ duos of type bb, and $\min\{m, n\} = m$ duos of type ba.

Suppose I is a yes-instance of 3-PARTITION. A solution is given by a set of triplets $\{T_1, ..., T_m\}$. Using this set, we can construct a solution to $J(I)$ which preserves exactly $m\Sigma + 2m$ duos.

For each $T_i = \{x_{i1}, x_{i2}, x_{i3}\}$, with $x_{i1} \leq x_{i2} \leq x_{i3}$, we map the whole sub-string A_{i1} to the first $x_{i1} + 2$ characters of sub-string B_i, we map all a's of A_{i2} to the following $x_{i2} + 1$ a's of B_i, and finally we map all a's of A_{i3} and the first letter of A_{i3+1} to the remaining part of B_i except for its final b, which remains unmatched for now.

The unmatched b's of A are arbitrarily matched with the remaining b's of B (their quantities are equal because B is a permutation of A).

The resulting overall mapping preserves every duo aa of A and all duos ab and ba of B, that is, a total of $m\Sigma + 2m$ duos. Thus, the first part of the reduction is complete.
($\Leftarrow$) Now suppose that there exists a mapping between A and B which preserves $m\Sigma + 2m$ duos. Since A and B have exactly $m\Sigma + 2m$ duos in common, the mapping must preserve all aa duos of A and all duos of type ab and ba of B.

Since all the aa duos of A are preserved, letters a of the word associated with any number of X cannot be mapped with letters a of more than one B_i. Since each B_i has exactly $m\Sigma+3$ letters a, each B_i receives words associated with numbers of X whose sum is at most $m\Sigma$. If one B_i receives words associated with numbers whose sum is strictly less than $m\Sigma$, then not all the aa duos of A are preserved, leading to a contradiction. Therefore, each B_i hosts a set of numbers whose sum is Σ. In other words, a 3-partition of the numbers is derived. ◀

5 Conclusion

When the alphabet used to form the instance is bounded, an exhaustive list of all possible cardinality assignments describing k-bounded string-cuts can be produced in polynomial time for any constant k. This fact helped us devise a Polynomial Time Approximation Scheme for MPSM$^\ell$, which is the best result one might expect as no FTPAS can exist for this problem unless P=NP. Future developments include inquiries as to how the techniques presented in this article can be adapted to tackle the weighted version of MPSM introduced in [22]. Though polynomial, the time complexity of our method may well prove to be intractable in practical cases with large instances. That is why we also intend to devise different approaches for MPSM$^\ell$ and evaluate them though experiments.

References

1 Stefano Beretta, Mauro Castelli, and Riccardo Dondi. Parameterized tractability of the maximum-duo preservation string mapping problem. *Theoretical Computer Science*, 646:16–25, 2016.

2 Nicolas Boria, Gianpiero Cabodi, Paolo Camurati, Marco Palena, Paolo Pasini, and Stefano Quer. A 7/2-Approximation Algorithm for the Maximum Duo-Preservation String Mapping Problem. In *27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, pages 11:1–11:8, 2016.

3 Nicolas Boria, Adam Kurpisz, Samuli Leppänen, and Monaldo Mastrolilli. Improved approximation for the maximum duo-preservation string mapping problem. In *International Workshop on Algorithms in Bioinformatics (WABI 2014)*, pages 14–25. Springer, 2014.

4 Brian Brubach. Further improvement in approximating the maximum duo-preservation string mapping problem. In *International Workshop on Algorithms in Bioinformatics (WABI 2016)*, pages 52–64. Springer, 2016.

5 Brian Brubach. Fast matching-based approximations for maximum duo-preservation string mapping and its weighted variant. In *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

6 Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz, and Irena Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In *Algorithms in Bioinformatics - 13th International Workshop, (WABI 2013)*, pages 244–258, 2013.

7 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA 2014)*, pages 102–121, 2014.

8 Wenbin Chen, Zhengzhang Chen, Nagiza F. Samatova, Lingxi Peng, Jianxiong Wang, and Maobin Tang. Solving the maximum duo-preservation string mapping problem with linear programming. *Theoretical Computer Science*, 530:1–11, 2014.

9 Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, Stefano Lonardi, and Tao Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 2(4):302–315, 2005.

10 Marek Chrobak, Petr Kolman, and Jirí Sgall. The greedy algorithm for the minimum common string partition problem. In *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, pages 84–95, 2004.

11 Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algorithms*, 3(1):2:1–2:19, 2007.

12 Peter Damaschke. Minimum common string partition parameterized. In *Algorithms in Bioinformatics, 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15-19, 2008. Proceedings*, pages 87–98, 2008.

13 Bartlomiej Dudek, Pawel Gawrychowski, and Piotr Ostropolski-Nalewaja. A family of approximation algorithms for the maximum duo-preservation string mapping problem. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

15 Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: Hardness and approximations. In *Algorithms and Computation, 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20-22, 2004, Proceedings*, pages 484–495, 2004.

16 Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *J. Comb. Optim.*, 23(4):519–527, 2012.

17 Bruce Johnson. A bioinformatics-inspired adaptation to Ukkonen's edit distance calculating algorithm and its applicability towards distributed data mining. *Journal of Software Engineering and Applications*, 1(1):8–12, 2008.

18 Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009.

19 Petr Kolman and Tomasz Walen. Approximating reversal distance for strings with bounded number of duplicates. *Discret. Appl. Math.*, 155(3):327–336, 2007.

20 Petr Kolman and Tomasz Walen. Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electron. J. Comb.*, 14(1), 2007.

21 Christian Komusiewicz, Mateus de Oliveira Oliveira, and Meirav Zehavi. Revisiting the parameterized complexity of maximum-duo preservation string mapping. *Theoretical Computer Science*, 847:27–38, 2020.

22 Saeed Mehrabi. Approximating weighted duo-preservation in comparative genomics. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics*, pages 396–406, Cham, 2017. Springer International Publishing.

23 Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

24 Patsaraporn Somboonsak and Mud-Armeen Munlin. A new edit distance method for finding similarity in Dna sequence. *World Academy of Science, Engineering and Technology, International Journal of Biological, Biomolecular, Agricultural, Food and Biotechnological Engineering*, 5:622–626, 2011.

25 Torsten Suel and Nasir Memon. Chapter 13 - algorithms for delta compression and remote file synchronization. In Khalid Sayood, editor, *Lossless Compression Handbook*, Communications, Networking and Multimedia, pages 269–289. Academic Press, San Diego, 2003.

26 Krister M. Swenson and Bernard M. E. Moret. Inversion-based genomic signatures. *BMC Bioinform.*, 10(S-1), 2009.

27 MK Vijaymeena and K Kavitha. A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal (MLAIJ)*, 3(1), 2016.

28 William E Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. *Proceedings of the Section on Survey Research Methods. American Statistical Association*, pages 354–359, 1990.

29 Yao Xu, Yong Chen, Guohui Lin, Tian Liu, Taibo Luo, and Peng Zhang. A (1.4+epsilon)-approximation algorithm for the 2-max-duo problem. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Treewidth-Based Algorithms for the Small Parsimony Problem on Networks

Celine Scornavacca
Institut des Sciences de l'Evolution, Université de Montpellier, CNRS, IRD, EPHE, France

Mathias Weller
LIGM, CNRS, Université Gustave Eiffel, Paris, France

Abstract

Phylogenetic reconstruction is one of the paramount challenges of contemporary bioinformatics. A subtask of existing tree reconstruction algorithms is modeled by the Small Parsimony problem: given a tree T and an assignment of character-states to its leaves, assign states to the internal nodes of T such as to minimize the *parsimony score*, that is, the number of edges of T connecting nodes with different states. While this problem is polynomial-time solvable on trees, the matter is more complicated if T contains reticulate events such as hybridizations or recombinations, i.e. when T is a network. Indeed, three different versions of the parsimony score on networks have been proposed and each of them is NP-hard to decide. Existing parameterized algorithms focus on combining the number of possible character-states with the number of reticulate events (per biconnected component). Here, we consider the treewidth of the undirected graph underlying the input network as parameter, presenting dynamic programming algorithms for (slight generalizations of) all three versions of the parsimony problem on networks. Our algorithms use a formulation of the treewidth that may facilitate formalizing treewidth-based dynamic programming algorithms on phylogenetic networks for other problems.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Applied computing → Molecular sequence analysis

Keywords and phrases Phylogenetics, parsimony, phylogenetic networks, parameterized complexity, dynamic programming, treewidth

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.6

Funding This work was supported by French Agence Nationale de la Recherche through the CoCoAlSeq project (ANR-19-CE45-0012).

Acknowledgements We thank Christophe Paul for sharing his expertise on treewidth formulations.

1 Introduction

Molecular phylogenetic reconstruction consists in inferring a well-founded evolutionary scenario of a set of species from molecular data [12]. An evolutionary scenario, also called a *phylogeny*, is usually represented by a directed tree with a unique source called *root*. In a phylogeny, the tips of the tree are associated to extant species for which we have data, and each internal node represents an extinct species giving rise to new species – a *speciation*. Therefore, each internal node represents the hypothetical ancestor of all species below it, and the root models the lowest common ancestor of all the species at the tips.

Parsimony on Trees

In this paper, molecular data consists of a set of molecular sequences (e.g. DNA or protein sequences) of the same length (one sequence per species). This kind of data can be seen as a matrix M of n sequences, each having m characters (exhibiting one of c possible states) where the state $M_{i,j}$ corresponds to the j^{th} character of the i^{th} species. There are several

methods to reconstruct well-founded phylogenies from matrices of characters [12]. They are all based on the idea of retrieving similarities among species by comparing the states taken by these species at the different characters of M. Here, we will focus on *parsimony methods*. The main hypothesis of these methods is that character changes are not frequent. Thus, the phylogenies that best explain the data are those requiring the fewest evolutionary changes, i.e. the ones having the optimal *parsimony score*, formally defined in Section 4. The problem of finding the optimal parsimony score for a given phylogeny T with respect to a matrix M is called the SMALL PARSIMONY problem and can be solved in $O(n \cdot m \cdot c)$ time [14] since each column in the matrix can be analyzed independently in linear time. When T is unknown, the problem of finding the phylogeny minimizing the parsimony score is called the BIG PARSIMONY problem. This latter is known to be NP-hard and numerous heuristic techniques for it are known [12].

Parsimony on Networks

When the evolution of the species of interest include, in additions to speciations, reticulate events such as *hybridizations* or *recombinations*, a single species may inherit from multiple direct ancestors. In this case, the phylogenies are no longer represented by rooted trees but by rooted DAGs [16] called *networks*. When scoring a given network, three very different definitions of the parsimony score have been proposed: the *hardwired* [20], the *softwired* [15, 26], and the *parental* parsimony score [32]. Roughly, the hardwired score takes into account all edges of the given network (characters are inherited from all parents), the softwired score takes only the edges of any "switching" (each character is inherited from one parent), and the parental score allows embedding lineages into the network (each allele of a character is inherited from one parent). See Section 4 for details and Figure 3 for an example. While these definitions coincide for trees, they give rise to three different small parsimony problems for networks.

When tracing mutually dependent characters (e.g. different genomic locations in a same non-recombinant region) on networks, we also have to make sure that dependent characters are inherited from the same parent (some columns of the matrix have to use the same "switching"/"embedding"). To avoid dealing with this problem, the small parsimony problems on networks have been studied predominantly under the assumption of independent genomic locations. This boils down to having $m = 1$ since each column of the matrix can be analyzed independently (as is the case for the small parsimony problem on trees). Another popular restriction is to consider *binary* networks, in which the root has outdegree 2, tips have indegree 1, and internal nodes have either indegree 1 and outdegree 2 (speciations) or indegree 2 and outdegree 1 (reticulations).

The hardwired small parsimony problem has been proven NP-hard and APX-hard whenever the number of states that a character can take, denoted c, is strictly greater than 2, and polynomial time solvable for binary characters [13]. A polynomial-time 1.35-approximation for all c and a $^{12}/_{11}$-approximation for $c = 3$ have been proposed [13]. Additionally, the problem has been shown fixed-parameter tractable (FPT) in the parsimony score [13], and with respect to $c + r$, where r is the number of reticulate events in the network [21].

The softwired small parsimony problem is also NP-hard and APX-hard [19, 13] for binary characters, and not FPT in the parsimony score (it is NP-hard to know if the softwired parsimony score is 1). Also, it has been shown that, for any constant $\epsilon > 0$, an approximation factor of $n^{1-\epsilon}$ is not possible in polynomial time, unless P = NP. On the positive side, the problem is FPT in $c + r$ [26, 13] and $c + \ell$, where ℓ is the *level* of the network [18, 13] (the maximum number of reticulations over all biconnected components of the network).

Unsurprisingly, the parental small parsimony problem has also been proven NP-hard, even for very restricted classes of networks [29], but is FPT both with respect to $c + r$ and with respect to $c + \ell$.

In this paper, we consider the case of independent characters, showing that the three variants of the small parsimony problem on networks are fixed-parameter tractable with respect to $c + t$, where t is the treewidth of the input network. Our proofs are constructive in the sense that a dynamic programming algorithm is provided for each version of the problem. Since the treewidth can be arbitrary small, even for growing values of ℓ, our algorithms can potentially be orders of magnitude faster than the state-of-the-art solutions. Moreover, our formulations are not limited to binary networks and they can take into account polymorphism as well as external information controlling the states that ancestral species may take.

Treewidth for Phylogenetic Networks

The treewidth of a graph can roughly be described as a measure of "tree-likeness" and it ranks among the smallest of such parameters [2] (in particular, the treewidth can be seen to be smaller than the level ℓ on any network). Together with the fact that it facilitates the design of dynamic programming algorithms, this explains the enormous popularity the treewidth received in the parameterized complexity community [5]. Starting with the groundbreaking work of Bryant and Lagergren [7] (using the celebrated result of Courcelle [9]), treewidth also gained traction with researchers studying algorithms for phylogenetics-related problems (surveyed in [8]). While this yielded some algorithms parameterized by the treewidth *of the display graph* of multiple trees (the result of "gluing" all trees at their leaves), we are not aware of any algorithms parameterized by the treewidth of the input network. In an attempt to facilitate the use of this parameter in future work, we dedicate Section 3 to presenting a "phylogenetics-friendly" formulation by representing tree-decompositions of the input network as a rooted tree Γ on the same vertex set as the network. In particular, this formulation generalizes our previously considered parameter "scanwidth" [3], which can be expected to yield easier dynamic programming formulations at the cost of being slightly larger than the treewidth.

Missing proofs are deferred to the appendix at the end of the paper.

2 Preliminaries

Mappings

For any x and y, we define $\delta(x, y)$ to be 0 if $x = y$ and 1, otherwise, and we abbreviate $1 - \delta(x, y) =: \overline{\delta}(x, y)$. We further abbreviate $\delta(\phi(x), \phi(y))$ as $\delta_\phi(x, y)$ for any function ϕ. We may denote a pair (x, y) as $x \to y$ if it is referring to an assignment of y to x by some function and as xy if it refers to an arc in a network. We sometimes use the name of a function $\phi : X \to Y$ to refer to its set of pairs $\{x \to y \mid \phi(x) = y\}$ and we let $\phi\,|_Z := \{(x \to y) \in \phi \mid x \in Z\}$ denote the *restriction* of ϕ to Z. We say $\phi(x) = \bot$ to indicate that ϕ is not defined for x. We denote the result of forcing $\phi(x) = y$ (whether or not x is mapped by ϕ) as

$$\phi\,[x \to y] := \begin{cases} \phi \cup \{x \to y\} & \text{if } \phi(x) = \bot \\ (\phi \setminus \{x \to \phi(x)\})\,[x \to y] & \text{otherwise} \end{cases}$$

Finally, for sets Z, X and $Y \subseteq X$ and functions ϕ and ψ, we write $\psi \trianglelefteq \phi$ (and say that ψ is a *subfunction* of ϕ) if

(a) $\phi : X \to Z$ and $\psi : Y \to Z$ and $\psi(x) \leq \phi(x)$ for all $x \in Y$, or
(b) $\phi : X \to 2^Z$ and $\psi : Y \to Z$ and $\psi(x) \in \phi(x)$ for all $x \in Y$, or
(c) $\phi : X \to 2^Z$ and $\psi : Y \to 2^Z$ and $\psi(x) \subseteq \phi(x)$ for all $x \in Y$.

Graphs and Phylogenetic Networks

In this work, we consider (weakly) connected directed acyclic graphs (DAGs) N that have a unique source ρ_N called *root*. If the sinks (aka *leaves*) of N are labeled, we call N a *phylogenetic network*. We denote the set of nodes of N with in-degree at least two by $R(N)$ and we call such nodes *reticulations*. If $R(N) = \varnothing$, then N is called a *tree*. The result of, for each $v \in R(N)$ removing all but one of its incoming arcs is called a *switching* of N and $\mathcal{S}(G)$ denotes the set of all switchings of N (observe that all switchings are spanning trees). Let $v \in V(N)$. We denote the successors (or "children") of v in G by $\mathrm{Succ}_G(v)$ and its predecessors (or "parents") by $\mathrm{Pred}_G(v)$. If N contains a directed u-w-path, then we say that w is a *descendant* of u and u is an *ancestor* of w (denoted as $w \leq_N u$ and $w <_N u$ if $u \neq w$). A set $Z \subseteq V(N)$ such that $u \not\leq_N w$ and $w \not\leq_N u$ for all $u, w \in Z$ is called an *anti-chain* in N. The *induced subgraph* $N[Z]$ of a set $Z \subseteq V(N)$ is the result of removing all nodes $x \in V(N) \setminus Z$ from N (together with their incident arcs) and, for any $v \in V(N)$, the network $N_v := N[\{w \mid w \leq_N v\}]$ is called the subnetwork *rooted at* v.

Large parts of this work are in context of a rooted tree Γ on $V(N)$ (see Figure 1). Specifically for the tree Γ, we permit ourselves to abbreviate $V(\Gamma_x)$ to Γ_x to increase readability. In such context, we additionally define the following sets for any nodes $y, z \in V(N)$: $\mathrm{Pred}_G^{\uparrow y}(z) := \mathrm{Pred}_G(z) \cap \Gamma_y$ and $\mathrm{Pred}_G^{\downarrow y}(z) := \mathrm{Pred}_G(z) \setminus \Gamma_y$ denote the respective *predecessors* of z in N that are or are not in Γ_y. Likewise, $\mathrm{Succ}_G^{\downarrow y}(z) := \mathrm{Succ}_G(z) \cap \Gamma_y$ and $\mathrm{Succ}_G^{\uparrow y}(z) := \mathrm{Succ}_G(z) \setminus \Gamma_y$ denote the respective *successors* of z in N that are or are not in Γ_y – notice that the arrow in the notation indicates the direction of the arc between z and the members of the set when drawing Γ top-to-bottom. If $z = y$, we drop y and simply write $\mathrm{Pred}_G^{\downarrow}(z)$, $\mathrm{Pred}_G^{\uparrow}(z)$, $\mathrm{Succ}_G^{\downarrow}(z)$, and $\mathrm{Succ}_G^{\uparrow}(z)$. We also abbreviate $\mathrm{Pred}_G^{\downarrow}(z) \cap R(G) =: \mathrm{Pred}_G^{\mathrm{R}\downarrow}(z)$ and $\mathrm{Succ}_G^{\uparrow}(z) \cap R(G) =: \mathrm{Succ}_G^{\mathrm{R}\uparrow}(z)$ as well as $\mathrm{Pred}_G^{\downarrow}(z) \setminus R(G) =: \mathrm{Pred}_G^{\mathrm{T}\downarrow}(z)$ and $\mathrm{Succ}_G^{\uparrow}(z) \setminus R(G) =: \mathrm{Succ}_G^{\mathrm{T}\uparrow}(z)$. All these functions generalize to sets $Z \subseteq V(N)$ (for example, $\mathrm{Pred}_G(Z) := \bigcup_{z \in Z} \mathrm{Pred}_G(z) \setminus Z$). Further, for any $X \subseteq V(N)$, we define the sets of arcs of N (a) from a node $u \in X$ to any ancestor of u in Γ as $A_X^{\uparrow}(N) := \{uw \in A(N) \mid u \in X \wedge u <_\Gamma w\}$ and (b) to a node $u \in X$ from any ancestor of u in Γ as $A_X^{\downarrow}(N) := \{uw \in A(N) \mid w \in X \wedge w <_\Gamma u\}$. For brevity, we abbreviate $A_X(N) := A_X^{\uparrow}(N) \cup A_X^{\downarrow}(N)$, $A_v^{\uparrow}(N) := A_{\Gamma_v}^{\uparrow}(N)$, $A_v^{\downarrow}(N) := A_{\Gamma_v}^{\downarrow}(N)$, and $A_v(N) := A_{\Gamma_v}(N)$.

3 An Alternative Formulation of Treewidth

In this section, we give an alternative definition of the *treewidth*, which allows to tackle the small parsimony problem for networks in a simpler and more intuitive way. Note that this alternative definition is known in the FPT community (Dendris et al. [11] call it the "support" of a vertex with respect to an ordering (when referring to Arnborg [1]) and Mescoff et al. [25], call it "tree vertex separation"). However, in these works its connection to treewidth is mostly touched in passing, so we felt the need to prove it explicitly here.

For a linear ordering σ of the nodes of an undirected graph G and a node x of G, let $\sigma[1..x]$ be the restriction of σ to the nodes preceeding x (that is, to $\{y \mid y \leq_\sigma x\}$). We write $x \rightsquigarrow_{G,\sigma} y$ if x and y are connected in $G[\sigma[1..x]]$. Note that $\rightsquigarrow_{G,\sigma}$ is a partial order on $V(G)$.

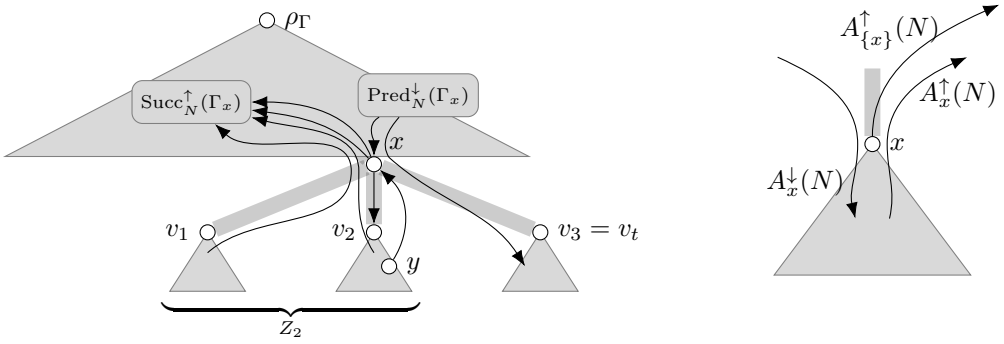


Figure 1 A tree Γ is depicted in gray and some arcs of N are depicted in black. Recall that t is the number of children of x and $Z_i := \bigcup_{1 \leq j \leq i} \Gamma_{v_j}$. Note that $x \in \mathrm{Succ}^{\uparrow}_N(Z_2) \setminus \mathrm{Succ}^{\uparrow}_N(\Gamma_x)$ since x is an ancestor of a node of Γ_{v_2} in N. Note that x is a reticulation of N with parents y (drawn) and z (not drawn) with $y <_\Gamma v_2 <_\Gamma x <_\Gamma z$. Thus, $z \in \mathrm{Pred}^{\downarrow}_N(x)$ but $y \in \mathrm{Pred}^{\uparrow_{v_2}}_N(x) \subseteq \mathrm{Pred}^{\uparrow}_N(x)$. Finally, note that $\mathrm{YW}^\Gamma_x = \mathrm{Pred}^{\downarrow}_N(\Gamma_x) \cup \mathrm{Succ}^{\uparrow}_N(\Gamma_x)$ and $\bigcup_{i \leq t} \mathrm{YW}^\Gamma_{v_i} \subseteq \mathrm{YW}^\Gamma_x \uplus \{x\}$.

▶ **Definition 1.** *Let σ be a linear order of the nodes of a graph G and let $v \in V(G)$. Then,*

$$\mathrm{ZW}^\sigma_v := \{u >_\sigma v \mid \exists_{w \in \sigma[1..v]} uw \in E(G) \wedge v \rightsquigarrow_{G,\sigma} w\} \qquad \text{and} \qquad \mathrm{zw}^\sigma_v := |\,\mathrm{ZW}^\sigma_v\,|.$$

Further, we abbreviate $\mathrm{zw}(\sigma) := \max_v \mathrm{zw}^\sigma_v$ and $\mathrm{zw}(G) := \min_\sigma \mathrm{zw}(\sigma)$. Further, we call the transitive reduction of the directed graph $(V(G), A^*)$ with $A^* := \{uv \in V(G)^2 \mid u \rightsquigarrow_{G,\sigma} v\}$ the canonical tree Γ^σ *of σ for G (as it turns out, Γ^σ is a rooted tree, see below).*

In the following, we say that a rooted tree Γ on $V(G)$ *agrees* with a directed or undirected graph G if, for all $uv \in E(G)$ either $u <_\Gamma v$ or $v <_\Gamma u$. We also extend the definition of $\rightsquigarrow_{G,\sigma}$ to such trees by writing $u \rightsquigarrow_{G,\Gamma} v$ if u and v are connected in $G[\Gamma_u]$.

▶ **Definition 2.** *Let G be a graph and let Γ agree with G. For each $v \in V(G)$, we define*

$$\mathrm{YW}^\Gamma_v := \{u >_\Gamma v \mid \exists_{w \leq_\Gamma v} uw \in E(G)\} \qquad \text{and} \qquad \mathrm{yw}^\Gamma_v := |\,\mathrm{YW}^\Gamma_v\,|$$

(see Figure 2). Then, we abbreviate $\mathrm{yw}(\Gamma) := \max_v \mathrm{yw}^\Gamma_v$ and $\mathrm{yw}(G) := \min_\Gamma \mathrm{yw}(\Gamma)$.

▶ **Lemma 3.** *Let Γ and Γ' be rooted trees agreeing with an undirected graph G and let $\leq_{\Gamma'}$ be a subset of $\leq_\Gamma$, that is, $x \leq_{\Gamma'} y \Rightarrow x \leq_\Gamma y$ for all $x, y \in V(G)$. Then, $\mathrm{yw}(\Gamma') \leq \mathrm{yw}(\Gamma)$.*

Proof. Let $x \in V(G)$ and let $y \in \mathrm{YW}^{\Gamma'}_x$, that is, $y >_{\Gamma'} x$ and there is some $z \leq_{\Gamma'} x$ with $yz \in E(G)$. Since $\leq_\Gamma$ is a superset of $\leq_{\Gamma'}$, we have $y >_\Gamma x \geq z$, implying $y \in \mathrm{YW}^\Gamma_x$. ◀

▶ **Lemma 4.** *Let σ be a linear order of the nodes of an undirected, connected graph G and let Γ^σ be its canonical tree. Then,*

(a) *for each u and v with $v \leq_{\Gamma^\sigma} u$, we have $v \leq_\sigma u$,*
(b) *for each $u, v \in V(G)$, we have $v \leq_{\Gamma^\sigma} u$ if and only if $u \rightsquigarrow_{G,\sigma} v$,*
(c) *Γ^σ is connected,*
(d) *Γ^σ is rooted at the last vertex r of σ,*
(e) *Γ^σ is a tree,*
(f) *for all $uv \in E(G)$ with $v <_\sigma u$, we have $v <_{\Gamma^\sigma} u$,*
(g) *Γ^σ agrees with G, and*
(h) *$\mathrm{YW}^{\Gamma^\sigma}_x = \mathrm{ZW}^\sigma_x$ for all $x \in V(G)$.*

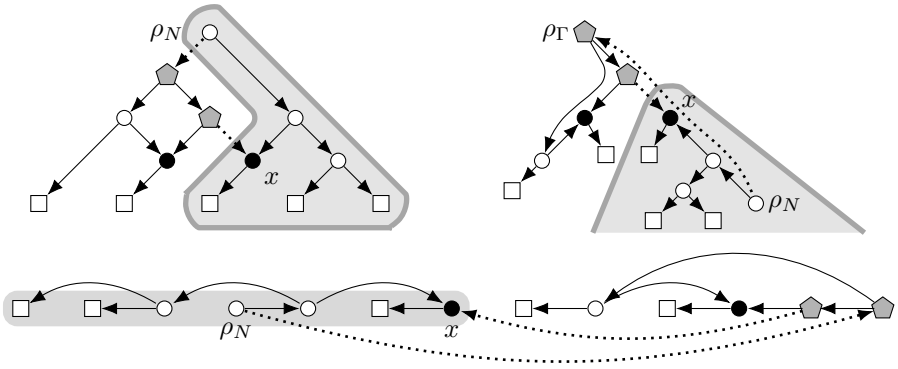


Figure 2 Example of a network N (left) with a linear order σ of its nodes (below) as well as their canonical tree Γ^σ (right) whose arcs are not drawn (the arcs of N are drawn in their stead). Reticulations are black, leaves are boxes. For the first (wrt. σ) reticulation x, the set $V(\Gamma^\sigma_x)$ is marked (gray area), the arcs $uv \in A_x(N)$ are dotted and the nodes in $\mathrm{YW}^\Gamma_v = \mathrm{ZW}^\sigma_v$ are gray pentagons.

▶ **Observation 5.** *Let Γ be a tree, let Γ' be a contraction of Γ, and let $x, y \in \Gamma'$ be distinct. Then, $x <_{\Gamma'} y$ if and only if $x <_\Gamma y$.*

For the following lemmas, it makes sense to "normalize" some aspects of the structure of agreeing trees. To this end, for a rooted tree T and for $X \subset V(T)$ that does not contain the root r of T, we let $T \uparrow X$ denote the result of (1) replacing each arc uv with $uv \cap X = \{u\}$ with the arc wv where w is the lowest ancestor of u that is not in X, and (2) removing all nodes in X from T. Note that $T \uparrow X$ may have strictly larger out-degree than T, but does not create new ancestor-descendant relations.

▶ **Observation 6.** *Let T be a tree, let $X \subseteq V(T)$ not contain its root, and let $u \leq_{T\uparrow X} v$. Then, $u \leq_T v$.*

▶ **Lemma 7.** *Let Γ be a rooted tree agreeing with an undirected graph G. There is some Γ^* agreeing with G such that $\mathrm{yw}(\Gamma^*) \leq \mathrm{yw}(\Gamma)$ and, for all $u, v \in V(G)$ with $v \leq_{\Gamma^*} u$, we have $u \rightsquigarrow_{G,\Gamma^*} v$.*

▶ **Lemma 8.** *Let Γ be a tree agreeing with a graph G and let p be a non-empty path in G. Then, p contains a unique maximum u with respect to Γ, that is, $v \leq_\Gamma u$ for all vertices v of p.*

Proof. Let x on p be maximal with respect to Γ (that is, for all z on p, we have $x \not<_\Gamma z$) and assume towards a contradiction that there is another vertex $y \neq x$ on p that is maximal w.r.t. Γ. Without loss of generality, let x precede y in p and let p_{xy} denote the unique x-y-subpath of p. Since $y \not\leq_\Gamma x$, there is an edge $st \in E(G)$ on p_{xy} with $s \leq_\Gamma x$ and $t \not\leq_\Gamma x$. Hence, $t \not\leq_\Gamma s$. Further, $s \not\leq_\Gamma t$ since, otherwise, the unique t-s-path in Γ contains x, contradicting its maximality. But then Γ does not agree with G. ◀

▶ **Lemma 9.** *Let G be a graph. Then, $\mathrm{zw}(G) = \mathrm{yw}(G)$.*

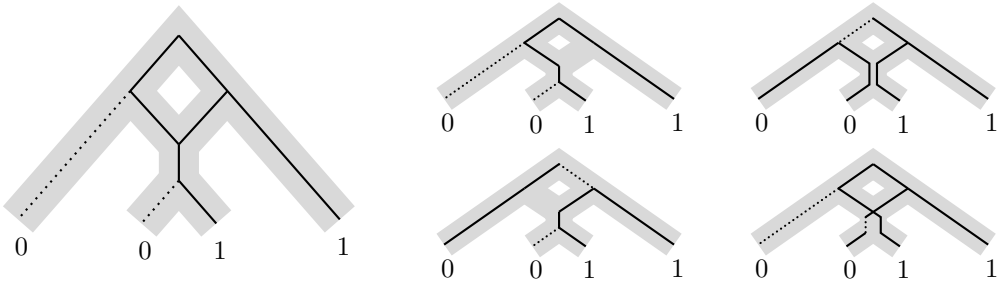


Figure 3 Example for parsimony scores of a network (in gray). Black edges participate in the score (solid = score 0, dotted = score 1). For the hardwired score (left), all edges of the network are considered. For the softwired score (2 possible trees: middle), only edges of any switching are considered. For the parental score (4 possible trees: middle & right), a tree is inscribed in the network.

▶ **Definition 10.** *Let G be a graph and let T be a rooted tree whose vertices are associated to subsets of $V(G)$ by a function $B : V(T) \to 2^{V(G)}$ such that*
(a) *for each $uv \in E(G)$, there is some $x \in V(T)$ with $uv \subseteq B(x)$ and*
(b) *for each $v \in V(G)$, the nodes $x \in V(T)$ with $v \in B(x)$ are weakly connected in T.*
We call (T, B) a tree decomposition *of G and its* width *is $\mathrm{tw}(T, B) := \max_{x \in V(T)} \mathrm{tw}_x^{T,B}$ with $\mathrm{tw}_x^{T,B} := |B(x)| - 1$. We call $\mathrm{tw}(G) := \min_{T,B} \mathrm{tw}(T, B)$ the* treewidth *of G. We call (T, B)* nice *if T is binary and all $x \in V(T)$ fall into one of the following categories*
***"leaf"**: x is a leaf of T and $B(x) = \varnothing$,*
***"root"**: x is the root of T and $B(x) = \varnothing$,*
***"introduce v"**: x has a single child y in T and $B(y) = B(x) - v$,*
***"forget v"**: x has a single child y in T and $B(x) = B(y) - v$,*
***"join"**: x has two children y and z and $B(x) = B(y) = B(z)$.*

All graphs G have a nice tree decomposition with $|V(T)| \in O(\mathrm{tw}(G) \cdot |G|)$ and width $\mathrm{tw}(G)$ [23]. Further, since all bags of (T, B) containing a vertex v of G are connected, we can observe the following.

▶ **Observation 11.** *Let (T, B) be a nice tree decomposition for an undirected graph G and let $v \in V(G)$. Then, T contains a single "forget v"-node x and $y <_T x$ for all y with $v \in B(y)$.*

▶ **Proposition 12.** *Let G be a graph. Then, $\mathrm{yw}(G) = \mathrm{tw}(G)$. Further, given a tree decomposition (T, B) for G, we can compute a tree Γ agreeing with G such that $\mathrm{yw}(\Gamma) = \mathrm{tw}(T, B)$ in linear time.*

4 Parsimony

Given states of a character, observed in extant species, as well as a species phylogeny, the small parsimony problem asks to infer states of the same character for all ancestral species such as to minimize the "parsimony score" of this assignment. This problem comes in three flavors called "hardwired", "softwired", and "parental" parsimony. Throughout this section, let C be a fixed finite set (a "character"). For convenient use of the $\trianglelefteq$-relation, let C be an anti-chain (that is, for each $x, y \in C$, we have $x \leq y$ only if $x = y$). Formally, for a phylogeny N and a function $\phi : V(N) \to 2^C$, we define the hardwired and softwired parsimony score as

$$\mathrm{par}_N^H(\phi) := \min_{\psi : V(N) \to C,\ \psi \trianglelefteq \phi} \sum_{uv \in A(N)} \delta_\psi(u, v) \qquad \mathrm{par}_N^S(\phi) := \min_{\substack{\psi : V(N) \to C,\ \psi \trianglelefteq \phi \\ T \in \mathcal{S}(N)}} \sum_{uv \in A(T)} \delta_\psi(u, v).$$

The "parental parsimony" is defined using "parental trees" but, in this work, we use the equivalent formulation using lineage functions [29].

▶ **Definition 13.** *A* lineage function *for a phylogeny N is any function $f : V(N) \to 2^C$. The* cost *of f is* $\text{cost}(f) := \sum_{v \in V(N)} \text{cost}_f(v)$ *where*

$$\text{cost}_f(v) := |f(v) \setminus \bigcup_{u \in \text{Pred}(v)} f(u)| + \begin{cases} -1 & \text{if } v = \rho_N \text{ and } |f(v)| = 1 \\ 0 & \text{if } v \neq \rho_N \text{ and } |f(v)| \leq \sum_{u \in \text{Pred}(v)} |f(u)| \\ \infty & \text{otherwise} \end{cases}$$

Given N and a function $\phi : V(N) \to 2^C$, we denote the set of all lineage functions f on N with $f \trianglelefteq \phi$ as $\mathcal{LF}_{N,\phi}$. Finally, the parental parsimony score *is*

$$\text{par}_N^P(\phi) := \min_{f \in \mathcal{LF}_{N,\phi}} \text{cost}(f) \tag{1}$$

For each of the presented variants, we give a dynamic programming formulation using a given tree Γ that agrees with the undirected graph G underlying the input network and corresponds to Lemma 7, that is, each non-leaf x of Γ has a child v with $x \in \text{YW}_v^\Gamma$. The running time of the resulting algorithm will depend on the width $\text{yw}(\Gamma)$ of Γ (recalling that $\text{yw}(\Gamma)$ coincides with the treewidth of G for optimal Γ).

As stated in the introduction, in this paper we focus on the case of analyzing a specific position in the genome. Since the function ϕ can associate several states to a same leaf, our definition permits to describe polymorphism in a population. While, in our current formulation, the algorithms "choose" an optimal state to associate to each leaf, the parental parsimony can be easily modified to explain *all* states of each leaf at the end of the run. This allows keeping the information on polymorphism in all steps of the algorithm (see Section 4.3). Note also that ϕ can associate information to internal nodes, thus permitting the user to impose restrictions on the states associated to ancestral species.

In the presentation of the dynamic programming, a table entry $Q_x^y[z]$ means that x and y are considered fix for this table and z is a variable index. Further, tables $Q_{x_1}^{y_1}$ and $Q_{x_2}^{y_2}$ are independent of one another, allowing an implementation to forget $Q_{x_1}^{y_1}$ if it is no longer needed, even if $Q_{x_2}^{y_2}$ still is. In the following, for an anti-chain Y in Γ and a class $\mathcal{G}$ of subnetworks of N, a *Y-substitution system* of $\mathcal{G}$ is a series of subnetworks $(N^y)_{y \in Y}$ of N such that, for all $N' \in \mathcal{G}$, the digraph $(V(N), (A(N') \setminus \bigcup_{y \in Y} A_y(N')) \cup \bigcup_{y \in Y} A_y(N^y))$ is also in $\mathcal{G}$. Roughly, we can "swap out" the arcs in $A_y(N')$ for $A_y(N^y)$ for each $y \in Y$ without loosing membership in $\mathcal{G}$. Note that the N^y are not necessarily distinct, so a trivial Y-substitution system for $\{N'\}$ would be $(N')_{y \in Y}$. The formulations are based on the following lemma about independent sub-solutions, showing that an optimal solution (S, ψ) for a sub-network (of G) "below" an anti-chain Z in Γ is also optimal on any sub-network "below" an anti-chain Y in Γ that is itself "below" Z (among all solutions with ψ's behavior on $\bigcup_{y \in Y} \text{YW}_y^\Gamma$).

▶ **Lemma 14** (see Figure 4)**.** *Let $Y, Z \subseteq V(N)$ be anti-chains in Γ such that $Y \subseteq \bigcup_{z \in Z} \Gamma_z$. Let $\mathcal{G}$ be a class of subnetworks of N and let $S \in \mathcal{G}$ and $\psi : V(N) \to C$ such that (a) $\sum_{z \in Z} \sum_{uw \in A_z(S)} \delta_\psi(u, w)$ is minimum among all such S and ψ. Let $(S^y)_{y \in Y}$ be a Y-substitution system for $\mathcal{G}$ and let $\psi_y : V(N) \to C$ for each $y \in Y$ such that (b) ψ_y and ψ coincide on YW_y^Γ. Then,*

$$\sum_{y \in Y} \sum_{uw \in A_y(S^y)} \delta_{\psi_y}(u, w) \geq \sum_{y \in Y} \sum_{uw \in A_y(S)} \delta_\psi(u, w).$$

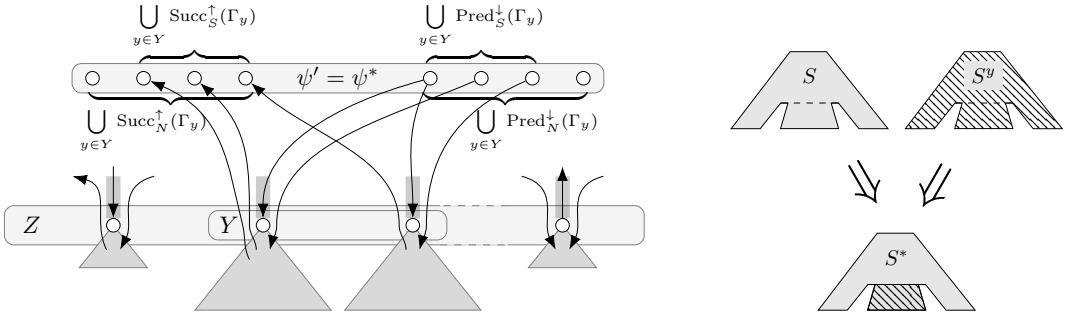


Figure 4 Lemma 14 proves that any solution (S, ψ) that is optimal on sub-trees rooted at Z in Γ must also be optimal (among all solutions with ψ's behavior on $\bigcup_{y\in Y} \mathrm{YW}^{\Gamma}_{y}$ (gray box on top)) on all sub-trees of Γ that are rooted below Z (at Y). That is, no solution (S^y, ψ_y) can be better than (S, ψ) on the sub-network induced by Γ_y for any $y \in Y$. To prove this, a new solution (S^*, ψ^*) is constructed by replacing the sub-solution of (S, ψ) below Y by the sub-solutions (S^y, ψ_y) below Y.

4.1 Hardwired Parsimony

To compute the hardwired parsimony score at a node v of N, we require knowledge of the character assigned to v and its neighbors. For all $u \in \mathrm{YW}^{\Gamma}_{v}$, we thus "guess" the character $\psi(u)$ assigned to u by an optimal assignment. In our dynamic programming, we scan Γ bottom-up, computing a table entry $T^{\mathcal{HW}}[x, \psi]$ for each $x \in V(\Gamma) = V(N)$ and each $\psi : \mathrm{YW}^{\Gamma}_{x} \to C$, containing the parsimony cost incurred by all arcs in $A_x(N)$, assuming that all nodes in YW^{Γ}_{x} receive their characters according to ψ. Note that $A_x(N) = \bigcup_i A_{v_i}(N) \cup A_{\{x\}}(N)$, where the v_i are the children of x in Γ. Thus, $T^{\mathcal{HW}}[x, \psi]$ can be calculated as follows.

▶ **Definition 15.** *Let Γ be a tree that agrees with N, let $x \in V(N)$ and let $\psi_x : \mathrm{YW}^{\Gamma}_{x} \to C$ with $\psi_x \trianglelefteq \phi$. Let $v_1, v_2, \ldots, v_t$ denote the children of x in Γ ($t = 0$ if x is a leaf). Then, we define a table entry*

$$T^{\mathcal{HW}}[x, \psi_x] := \min_{c_x \in \phi(x)} \left(\sum_{1 \le i \le t} T^{\mathcal{HW}}\left[v_i, \psi_x\,[x \to c_x]\,|_{\mathrm{YW}^{\Gamma}_{v_i}}\right] + \sum_{z \in \mathrm{Pred}^{\downarrow}_{N}(x) \cup \mathrm{Succ}^{\uparrow}_{N}(x)} \delta(c_x, \psi_x(z)) \right) \quad (2)$$

▶ **Lemma 16.** *Let $x \in V(N)$ and let $\psi_x : \mathrm{YW}^{\Gamma}_{x} \to C$ with $\psi_x \trianglelefteq \phi$. Let $\psi : V(N) \to C$ with $\psi_x \trianglelefteq \psi \trianglelefteq \phi$ such that ψ minimizes $\sum_{uw \in A_x(N)} \delta_\psi(u, w)$. Then,*

$$T^{\mathcal{HW}}[x, \psi_x] = \sum_{uw \in A_x(N)} \delta_\psi(u, w)$$

Proof Sketch. For "$\ge$", we construct a mapping ψ' from mappings ψ_i that are optimal on $A_{v_i}(N)$ among all mappings with $\psi_i(x) := c_x$. This is possible since all such ψ_i coincide with ψ' and ψ_x on YW^{Γ}_{x}. By induction hypothesis, the cost of ψ' on $A_x(N)$ is $\sum_{1 \le i \le t} T^{\mathcal{HW}}\left[v_i, \psi'\,|_{\mathrm{YW}^{\Gamma}_{v_i}}\right] + \sum_{uw \in A_{\{x\}}(N)} \delta_{\psi'}(u, w)$. Then, "$\ge$" follows from optimality of ψ on $A_x(N)$.

For "$\le$", it suffices to show that the cost of ψ on $A_x(N)$ is equal to the result of setting $c_x := \psi(x)$ in the right hand side of (2) (which is a valid choice for the minimum since $\psi(x) \in \phi(x)$). First, the cost of ψ on $A_{v_i}(N)$ is $T^{\mathcal{HW}}\left[v_i, \psi\,|_{\mathrm{YW}^{\Gamma}_{v_i}}\right]$ by independence of sub-solutions and the induction hypothesis. Second, the cost of ψ on $A^{\downarrow}_{\{x\}}(N)$ is $\sum_{z \in \mathrm{Pred}^{\downarrow}_{N}(x)} \delta(c_x, \psi_x(z))$ and the cost of ψ on $A^{\uparrow}_{\{x\}}(N)$ is $\sum_{z \in \mathrm{Succ}^{\uparrow}_{N}(x)} \delta(c_x, \psi_x(z))$ since ψ and ψ_x coincide on YW^{Γ}_{x}. ◀

In order to solve the hardwired parsimony problem given N, ϕ and Γ, all we have to do is compute $T^{\mathcal{HW}}[x, \psi_x]$ for each x bottom-up in Γ and each of the (at most) $|C|^{|\operatorname{YW}_x^\Gamma|}$ many choices of $\psi_x : \operatorname{YW}_x^\Gamma \to C$ with $\psi_x \trianglelefteq \phi$. Then, by Lemma 16, the hardwired parsimony score of N with respect to ϕ can be read from $T^{\mathcal{HW}}[\rho_\Gamma, \varnothing]$. To compute $T^{\mathcal{HW}}$, the sum over the children of x for all $x \in V(N)$ in (2) can be computed in amortized $O(|A(N)|)$ time and, with a bit of bookkeeping, it is possible to maintain the value of the second sum in (2) in $O(|A(N)|)$ amortized time per choice of ψ. Then the following holds:

▶ **Theorem 17.** *Given a network N, some $\phi : V(N) \to 2^C$ and a tree Γ agreeing with N, the hardwired parsimony score of (N, ϕ) can be computed in $O(|C|^{\operatorname{yw}(\Gamma)+1} \cdot |A(N)|)$ time.*

Proposition 12 lets us turn tree decompositions of N into trees Γ agreeing with N, allowing us to replace $\operatorname{yw}(\Gamma)$ by $\operatorname{tw}(N)$, incurring an additional running time of $|N| \cdot 2^{O(\operatorname{tw}(N)^3)}$ [4].

▶ **Corollary 18.** *Let (N, ϕ) be an instance of Hardwired Parsimony. Let $t \geq \operatorname{tw}(N)$ and let T be the time in which a width-t tree decomposition of N can be computed. Then, the hardwired parsimony score of (N, ϕ) can be computed in $O(T + |C|^{t+1} \cdot |A(N)|)$ time.*

4.2 Softwired Parsimony

In contrast to the hardwired parsimony score, where the computation of the cost of the incident edges of a node x only required knowledge of the characters assigned to neighbors of x, computing the *softwired* score additionally requires knowledge of which parent of x remains a parent in the sought switching. A table entry $T^{\mathcal{SW}}[x, \ldots]$ contains the smallest combined cost of all arcs in $A_x(S)$ for a switching S of N minimizing this cost. To be able to compute an entry for $x \in V(N)$, we not only need to "guess" ψ_x but, additionally, some representation of the switching S. In particular, in S, no child of x may have another parent than x. However, since children of x in N may be above x in Γ, we have to "guess" which children of x in N are still children of x in S. Such a guess manifests itself as an additional index R^x of the dynamic programming table (note that we clearly only have to store this information for children of x that are reticulations). Indeed, this information has to be stored for all nodes considered below x who still have children in $\operatorname{YW}_x^\Gamma$. Thus, we index our DP-table also by a subset $R^x \subseteq \operatorname{YW}_x^\Gamma \cap R(N)$ containing a reticulation $r \in R(N)$ if and only if Γ_x contains a parent v of r and vr is an arc of an optimal switching S for $N[\Gamma_x \cup \operatorname{YW}_x^\Gamma]$.

▶ **Definition 19.** *Let Γ be a tree that agrees with N, let $x \in V(N)$, let $\psi_x : \operatorname{YW}_x^\Gamma \to C$ with $\psi_x \trianglelefteq \phi$, and let $R^x \subseteq \operatorname{Succ}_N^{R\uparrow}(\Gamma_x)$. Let $v_1, v_2, \ldots, v_t$ denote the children of x in Γ ($t = 0$ if x is a leaf in Γ). Then, set*

$$T^{\mathcal{SW}}[x, \psi_x, R^x] := \min_{c_x \in \phi(x)} \min_{R^* \subseteq R^x \cap \operatorname{Succ}_N^{R\uparrow}(x)} \sum_{r \in R^* \cup \operatorname{Succ}_N^{T\uparrow}(x)} \delta(c_x, \psi_x(r)) + \min \begin{cases} Q_{x,c_x}^{\psi_x}[t, R^x \setminus R^*] + \min\limits_{y \in \operatorname{Pred}_N^\downarrow(x)} \delta(c_x, \psi_x(y)) & \text{if } \operatorname{Pred}_N^\downarrow(x) \neq \varnothing \\ Q_{x,c_x}^{\psi_x}[t, (R^x \setminus R^*) \cup (\{x\} \cap R(N))] & \text{if } \operatorname{Pred}_N^\uparrow(x) \neq \varnothing \end{cases} \tag{3}$$

where

$$Q_{x,c_x}^{\psi_x}[i, R'] := \begin{cases} \min\limits_{R^* \subseteq R' \cap \operatorname{Succ}_N^{R\uparrow}(\Gamma_{v_i})} Q_{x,c_x}^{\psi_x}[i-1, R' \setminus R^*] + T^{\mathcal{SW}}[v_i, \psi_i, R^*] & \text{if } i \neq 0 \\ 0 & \text{if } i = 0 \text{ and } R' = \varnothing \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

where $\psi_i := \psi_x[x \to c_x]|_{\operatorname{YW}_{v_i}^\Gamma}$ for all $i \leq t$. (Note how $Q_{x,c_x}^{\psi_x}[i, R']$ is used to assign the nodes in R^x to the v_i (with $v_0 = x$) such that every node in R^x has a parent in some Γ_{v_i}).

In the following, for any anti-chain$X$ in Γ and all $Z \subseteq \bigcup_{x\in X} \mathrm{YW}^\Gamma_x$, let $\mathcal{S}^{X\to Z}(N)$ denote the set of all switchings S of N with $\mathrm{Succ}^{\mathrm{R}\uparrow}_S(X) = Z$.

▶ **Lemma 20.** *Let Γ be a tree that agrees with N, let $x \in V(N)$, let $\psi_x : \mathrm{YW}^\Gamma_x \to C$ with $\psi_x \trianglelefteq \phi$, and let $R^x \subseteq \mathrm{Succ}^{R\uparrow}_N(\Gamma_x)$. If $\mathcal{S}^{\Gamma_x\to R^x}(N) = \varnothing$, then $T^{\mathcal{SW}}[x, \psi_x, R^x] = \infty$. Otherwise, let $S \in \mathcal{S}^{\Gamma_x\to R^x}(N)$ and $\psi : V(N) \to C$ such that*
(a) *$\psi_x \trianglelefteq \psi \trianglelefteq \phi$ and*
(b) *$\sum_{uw\in A_x(S)} \delta_\psi(u, w)$ is minimum among all such S and ψ.*
Then,

$$T^{\mathcal{SW}}[x, \psi_x, R^x] = \sum_{uw\in A_x(S)} \delta_\psi(u, w). \tag{5}$$

Proof Sketch. Let us abbreviate $Z_i := \bigcup_{j\le i} V(\Gamma_{v_j})$. We first show that the table Q does what we expect it to do.

▷ Claim 21. $Q^{\psi_x}_{x,c_x}[i, R'] = \sum_{j\le i} \sum_{uw\in A_{v_j}(S_i)} \delta_{\psi_i}(u, w)$ for optimal $S_i \in \mathcal{S}^{Z_i\to R'}$ and ψ_i coincides with $\psi_x[x \to c_x]$ on $\bigcup_{j\le i} \mathrm{YW}^\Gamma_{v_j}$.

Proof Sketch. For "$\ge$", let $R^* \subseteq R' \cap \mathrm{Succ}^{\mathrm{R}\uparrow}_N(\Gamma_{v_i})$ such that equality holds in (4). We consider a switching $S' \in \mathcal{S}^{Z_i\to R'}$ constructed from switchings $S_{i-1} \in \mathcal{S}^{Z_{i-1}\to R'\setminus R^*}$ and $S^* \in \mathcal{S}^{\Gamma_{v_i}\to R^*}$ as well as a mapping ψ' coinciding with $\psi_x[x \to c_x]$ on $\bigcup_{j<i} \mathrm{YW}^\Gamma_{v_j}$ constructed from mappings ψ_{i-1} and ψ^* such that
(a) ψ_{i-1} coincides with $\psi_x[x \to c_x]$ on $\bigcup_{j<i} \mathrm{YW}^\Gamma_{v_j}$,
(b) ψ^* coincides with $\psi_x[x \to c_x]$ on $\mathrm{YW}^\Gamma_{v_i}$,
(c) the cost of ψ_{i-1} is optimal on $A_{Z_{i-1}}(S_{i-1})$ and
(d) the cost of ψ^* is optimal on $A_{v_i}(S^*)$.
By induction hypotheses, these costs are $Q^{\psi_x}_{x,c_x}[i-1, R' \setminus R^*]$ and $T^{\mathcal{SW}}[v_i, \psi_x[x \to c_x], R^*]$, respectively. Then, "$\ge$" follows by optimality of S_i and ϕ_i.

For "$\le$", we let $R^* := \mathrm{Succ}^{\mathrm{R}\uparrow}_{S_i}(\Gamma_{v_i})$ and use independence of sub-solutions and the induction hypotheses to show that the cost of ϕ_i on $A_{Z_{i-1}}(S_i)$ is $Q^{\psi_x}_{x,c_x}[i-1, R' \setminus R^*]$ and the cost of ϕ_i on $A_{v_i}(S_i)$ is $T^{\mathcal{SW}}[v_i, \phi_i, R^*]$. Then, "$\le$" follows from the fact that R^* is only one of the possible choices for the minimum in (4). ◁

For "$\ge$", let $c_x \in \phi(x)$ and $R^* \subseteq R^x \cap \mathrm{Succ}^{\mathrm{R}\uparrow}_N(x)$ be such that equality holds in (3). We consider a switching $S' \in \mathcal{S}^{\Gamma_x\to R^x}$ constructed from switchings S_t and S^* with $S_t \in \mathcal{S}^{Z_t\to R^x\setminus R^*}$ (if $\mathrm{Pred}^\downarrow_N(x) \ne \varnothing$) or $S_t \in \mathcal{S}^{Z_t\to (R^x\setminus R^*)\cup\{x\}}$ (if $x \in R(N)$ and $\mathrm{Pred}^\uparrow_N(x) \ne \varnothing$), and $S^* \in \mathcal{S}^{\{x\}\to R^*}$, as well as a mapping ψ' coinciding with ψ_x on YW^Γ_x constructed from mappings ψ_t and ψ^* such that
1. ψ_t coincides with $\psi_x[x \to c_x]$ on $\bigcup_{i\le t} \mathrm{YW}^\Gamma_{v_i}$,
2. ψ^* coincides with ψ_x on YW^Γ_x,
3. $\psi^*(x) = c_x$,
4. the cost of ψ_t is optimal on $A_{Z_t}(S_t)$ and
5. the cost of ψ^* is optimal on $A_{\{x\}}(S^*)$.
Then, the cost of ψ^* on $A^\uparrow_{\{x\}}(S^*)$ is $\sum_{r\in R^*\cup \mathrm{Succ}^{\mathrm{T}\uparrow}_N(x)} \delta(c_x, \psi_x(r))$, the cost of ψ^* on $A^\downarrow_{\{x\}}(S^*)$ is $\min_{y\in \mathrm{Pred}^\downarrow_N(x)} \delta(c_x, \psi_x(y))$ if the parent of x in S_t is above x in Γ (that is, $x \notin \mathrm{Succ}^{\mathrm{R}\uparrow}_{S_t}(Z_t)$) and, by the claim above, the cost of ψ_t on $A_{Z_t}(S_t)$ is $Q^{\psi_x}_{x,c_x}\left[t, \mathrm{Succ}^{\mathrm{R}\uparrow}_{S_t}(Z_t)\right]$. Then, as $S' \in \mathcal{S}^{\Gamma_x\to R^x}$, "$\ge$" follows by optimality of S and ϕ.

For "$\leq$", let $c_x := \phi(x)$ and let $R^* := \mathrm{Succ}_S^{\mathrm{R}\uparrow}(\Gamma_x)$. We use independence of sub-solutions and the induction hypothesis to show that the cost of ϕ on $A_{Z_t}(S)$ is $Q^{\psi_x}_{x,c_x}[t, R' \setminus R^*]$ (if $x \notin R(N)$ or the parent of x in S is above x in Γ) or $Q^{\psi_x}_{x,c_x}[t, (R' \setminus R^*) \cup \{x\}]$ (if $x \in R(N)$ and the parent of x in S is in Γ_x). Further, the cost of ψ on $A^{\uparrow}_{\{x\}}(S)$ is $\sum_{r \in R^* \cup \mathrm{Succ}_N^{\mathrm{T}\uparrow}(x)} \delta(c_x, \psi_x(r))$, the cost of ψ on $A^{\downarrow}_{\{x\}}(S)$ is $\min_{y \in \mathrm{Pred}_N^{\downarrow}(x)} \delta(c_x, \psi_x(y))$ if the parent of x in S is above x in Γ. Then, "$\leq$" follows from the fact that our choices of c_x and R^* are only one of the possible choices for the minimum in (3). ◀

In order to solve the softwired parsimony problem given N, ϕ and Γ, all we have to do is compute $T^{\mathcal{SW}}[x, \psi_x, R^x]$ for each x bottom-up in Γ, each of the (at most) $|C|^{|\,\mathrm{YW}_x^\Gamma\,|}$ many choices of $\psi_x : \mathrm{YW}_x^\Gamma \to C$ with $\psi_x \trianglelefteq \phi$, and each $R^x \subseteq \mathrm{Succ}_N^{\mathrm{R}\uparrow}(x) \subseteq \mathrm{YW}_x^\Gamma \cap R(N)$. To this end, $Q^{\psi_x}_{x,c_x}[i, R^x \setminus R^*]$ and $Q^{\psi_x}_{x,c_x}[i, (R^x \setminus R^*) \cup \{x\}]$ have to be computed for each child v_i of x in Γ and each $R^* \subseteq R^x \cap \mathrm{Succ}_N^{\mathrm{R}\uparrow}(x)$. Then, by Lemma 20, the softwired parsimony score of N with respect to ϕ can be read from $T^{\mathcal{SW}}[\rho_\Gamma, \varnothing, \varnothing]$. In the following, let ψ_x be fix. Then, for fix c_x, we can compute $Q^{\psi_x}_{x,c_x}[i, R']$ for all choices of x, i and R' in $O(2^{|R' \cap \mathrm{Succ}_N^{\mathrm{R}\uparrow}(v_i)|} + \sum_{x \in \Gamma} |\,\mathrm{Succ}_\Gamma(x)|) \subseteq O(2^{|\,\mathrm{YW}_x^\Gamma\,|+1} + |\Gamma|)$ time total. Further, the values of $\min_{y \in \mathrm{Pred}_N^{\downarrow}(x)} \delta(c_x, \phi_x(y))$ can be pre-computed for all $x \in \Gamma$ in $O(|A(N)|)$ time total. Then, to compute $T^{\mathcal{SW}}[x, \psi_x, R^x]$ for all x and R^x, we have to check $|V(N)|$ choices for x, as well as $|\phi(x)| \leq |C|$ choices for c_x and $3^{|\,\mathrm{Succ}_N^{\mathrm{R}\uparrow}(x)|}$ choices for R^x and $R^* \subseteq R^x$ combined. Altogether, the table $T^{\mathcal{SW}}$ can be computed in $O(|C|^{|\,\mathrm{YW}_x^\Gamma\,|} \cdot (3^{|\,\mathrm{YW}_x^\Gamma\,|} \cdot |C| \cdot |V(N)| + |A(N)|))$ time. The computation of $Q^{\psi_x}_{x,c_x}$ in $O(2^{|\,\mathrm{YW}_x^\Gamma\,|} + |A(N)|)$ time is absorbed by this. For practical purposes, note that estimating $|\,\mathrm{Succ}_N^{\mathrm{R}\uparrow}(x)| \leq |\,\mathrm{YW}_x^\Gamma\,|$ is quite crude and equality will almost never be attained. Then, the following result holds:

▶ **Theorem 22.** *Given a network N, $\phi : V(N) \to 2^C$ and a tree Γ agreeing with N, the softwired parsimony score of (N, ϕ) can be computed in $O(|C|^{\mathrm{yw}(\Gamma)} \cdot (3^{\mathrm{yw}(\Gamma)} \cdot |C| \cdot |V(N)| + |A(N)|))$ time.*

Again, we can replace $\mathrm{yw}(\Gamma)$ by $\mathrm{tw}(N)$ using Proposition 12.

▶ **Corollary 23.** *Let (N, ϕ) be an instance of Softwired Parsimony. Let $t \geq \mathrm{tw}(N)$ and let T be the time in which a width-t tree decomposition of N can be computed. Then, the softwired parsimony score of (N, ϕ) can be computed in $O(T + |C|^t \cdot (3^t \cdot |C| \cdot |V(N)| + |A(N)|))$ time.*

4.3 Parental Parsimony

For ease of presentation, we introduce some additional notation. First, for any a and b, we abbreviate $\max\{a - b, 0\} =: a \dot{-} b$. Let ψ and ψ' be functions with the same codomain. If ψ maps all items to $\varnothing$ or to 0, then we say that ψ is a *zero-function* and we write $\psi = \overrightarrow{0}$. We use $\psi - \psi'$ to denote the function defined on the domain of ψ for which $(\psi - \psi')(x) = \psi(x)$ if $\psi'(x) = \perp$ and $(\psi - \psi')(x) = \psi(x) - \psi'(x)$, otherwise. This definition extends to functions mapping to sets in a natural way.

Each lineage function gives rise to one or more phylogenetic trees, called *lineages*, embedded in N. For each $x \in V(N)$, $f(x)$ represents the set of branches of such a lineage passing through x. Each such lineage-branch may "choose" a parent among the parents of x in N. This models the biological circumstance that a character trait may be inherited from any parent. We compute (the cost of) an optimal lineage function on N using a tree Γ that agrees with N. To compute $\mathrm{cost}_f(x)$, we require knowledge of $\sum_{y \in \mathrm{Pred}(x)} |f(y)|$ as well as $\bigcup_{y \in \mathrm{Pred}(x)} f(y)$. For all $y \in \mathrm{YW}_x^\Gamma$, we thus store the set $\lambda(y) := f(y)$ of lineages in y,

the subset $\psi(y)$ of lineages of y that also occur in parents (in N) of y that are below x in Γ, that is, $\mathrm{Pred}_N^{\uparrow x}(y)$ (such lineages are inherited by y at no cost), and the total number $\eta(y)$ of lineages of y that can be inherited from parents (in N) of y that are below x in Γ, that is, $\mathrm{Pred}_N^{\uparrow x}(y)$ (cost 0 or 1). Then, $\sum_{y\in\mathrm{Pred}_N(x)}|f(y)| = \eta(x) + \sum_{y\in\mathrm{Pred}_N^{\downarrow}(x)}|\lambda(y)|$ and $\bigcup_{y\in\mathrm{Pred}_N(x)} f(y) = \psi(x) \cup \bigcup_{y\in\mathrm{Pred}_N^{\downarrow}(x)} \lambda(y)$.

In order to compute an entry $T^{\mathcal{PT}}[x, \lambda_x, \psi_x, \eta_x]$, we "guess" the set $U \subseteq \phi(x)$ of lineages passing through x in an optimal solution, as well as the set $D \subseteq U$ of lineages inherited from nodes in $\mathrm{Pred}_N^{\uparrow}(x)$. Then, the cost incurred by x is the number of lineages of x that are not lineages of any $r \in \mathrm{Pred}_N(x)$, that is, the number of lineages in $U \setminus (D \cup \bigcup_{r\in\mathrm{Pred}_N^{\downarrow}(x)} \lambda(r))$. For the recursive table lookup, we have to make sure that $\lambda(x) = U$, $\psi(x) = D$, and that all lineage branches of x that do not come from $\mathrm{Pred}_N^{\downarrow}(x)$ can be inherited from $\mathrm{Pred}_N^{\uparrow}(x)$, that is, $\eta(x) = |\lambda(x)| \dot{-} \sum_{r\in\mathrm{Pred}_N^{\downarrow}(x)} |\lambda(r)|$. Further, each child y of x in N may inherit a lineage from x and, if y is above x in Γ, this has to be registered by removing the lineages of U from $\psi(y)$ and subtracting $|U|$ from $\eta(y)$. Finally, the lineage branches represented by ψ and η are distributed among the children of x in Γ using the table Q. In the following, in order to avoid treating the case that $x = \rho_N$ separately, we define $\rho(x) := 1 - \delta(x, \rho_N)$, that is, $\rho(x) = 1$ if and only if $x = \rho_N$.

▶ **Definition 24.** *Let Γ be a tree that agrees with N, $x \in V(N)$, $\lambda_x : \mathrm{YW}_x^\Gamma \to 2^C$ with $\lambda_x \trianglelefteq \phi$ and $\psi_x \trianglelefteq \lambda_x$. Let $\{v_1, v_2, \ldots, v_t\} = \mathrm{Succ}_\Gamma(x)$ ($t = 0$ if x is a leaf in Γ). Then, set $T^{\mathcal{PT}}[x, \lambda_x, \psi_x, \eta_x]$ to*

$$\min_{\substack{D\subseteq U\subseteq\phi(x)\\ U\neq\varnothing}} Q_x^{\lambda_x[x\to U]}\left[t, \psi_x\left[\begin{smallmatrix} x\to D \\ \forall_{w\in\mathrm{Succ}_N^{\uparrow}(x)}\, w\to\psi_x(w)\setminus U\end{smallmatrix}\right], \eta_x\left[\begin{smallmatrix} x\to |U|\dot{-}\sum_{u\in\mathrm{Pred}_N^{\downarrow}(x)}|\lambda_x(u)| \\ \forall_{w\in\mathrm{Succ}_N^{\uparrow}(x)}\, w\to\eta_x(w)\dot{-}|U|\end{smallmatrix}\right]\right] + \left|U \setminus \left(D \cup \bigcup_{u\in\mathrm{Pred}_N^{\downarrow}(x)} \lambda_x(u)\right)\right| \tag{6}$$

where $Q_x^\lambda[i, \psi, \eta]$ equals

$$\begin{cases} \min\limits_{\psi'\trianglelefteq\psi|_{\mathrm{YW}_{v_i}^\Gamma}} \min\limits_{\eta'\trianglelefteq\eta|_{\mathrm{YW}_{v_i}^\Gamma}} Q_x^\lambda[i-1, \psi-\psi', \eta-\eta'] + T^{\mathcal{PT}}\left[v_i, \lambda|_{\mathrm{YW}_{v_i}^\Gamma}, \psi', \eta'\right] & \text{if } i > 0 \\ -\rho(x) & \text{if } i = 0 \text{ and } \psi = \overrightarrow{0} \text{ and } \eta = \overrightarrow{0}\,[x \to \rho(x)] \\ \infty & \text{otherwise} \end{cases} \tag{7}$$

Note how the table Q_x^λ distributes the lineage branches of x whose parents are in Γ_x among the children of x in Γ. Observe that both $T^{\mathcal{PT}}$ and Q_x^λ are monotone in ψ and η (wrt. $\trianglelefteq$) by construction.

▶ **Lemma 25.** *Let $x \in V(N)$, let $i \in \mathbb{N}$, let $\lambda : \mathrm{YW}_x^\Gamma \to 2^C$, let $\eta, \eta' : \mathrm{YW}_x^\Gamma \to \mathbb{N}$, and let $\psi, \psi' : \mathrm{YW}_x^\Gamma \to 2^C$ such that $\psi' \trianglelefteq \psi \trianglelefteq \lambda$ and $\overrightarrow{0}\,[x \to \rho(x)] \trianglelefteq \eta' \trianglelefteq \eta$. Then,*

$$T^{\mathcal{PT}}[x, \lambda, \psi', \eta'] \leq T^{\mathcal{PT}}[x, \lambda, \psi, \eta] \qquad \text{and} \qquad Q_x^\lambda[i, \psi', \eta'] \leq Q_x^\lambda[i, \psi, \eta]$$

Proof Sketch. The lemma can be proved by induction on the height of x in Γ and the value of i. If x is a leaf, then $Q_x^\lambda[0, \psi, \eta]$ is finite only if $\psi = \overrightarrow{0}$ and $\eta = \overrightarrow{0}\,[x \to \rho(x)]$, implying the second inequality. For monotony of $T^{\mathcal{PT}}$, fix the sets $D \subseteq U \subseteq C$ for which the minimum in the formula of $T^{\mathcal{PT}}[x, \lambda, \psi, \eta]$ is attained. Then, by monotony of Q_x^λ, replacing ψ by ψ' and η by η' in this formula does not increase its value and this value is at most $T^{\mathcal{PT}}[x, \lambda, \psi', \eta']$

since it is obtained for one of several possible choices for D and U. If x is not a leaf in Γ then monotonicity of $Q_x^\lambda[i,\ldots]$ is implied by monotonicity of $Q_x^\lambda[i-1,\ldots]$ and monotonicity of $T^{\mathcal{PT}}[v,\ldots]$ for the children v of x. Finally, monotonicity of $T^{\mathcal{PT}}$ follows from monotonicity of Q_x^λ as in the induction base. ◀

▶ **Lemma 26.** *Let Γ be a tree agreeing with N, let $x \in V(N)$, let $\psi_x, \lambda_x : \mathrm{YW}_x^\Gamma \to 2^c$ and $\eta_x : \mathrm{YW}_x^\Gamma \to \mathbb{N}$. Let f minimize $\mathrm{cost}(f)$ among all lineage functions in $\mathcal{LF}_{N,\phi}$ such that, for all $w \in \mathrm{YW}_x^\Gamma$, $\lambda_x(w) = f(w)$, $\psi_x(w) = f(w) \cap \bigcup_{u \in \mathrm{Pred}_N^{\uparrow x}(w)} f(u)$, and $\eta_x(w) \leq \sum_{u \in \mathrm{Pred}_N^{\uparrow x}(w)} |f(u)|$. If there are no such f, then $T^{\mathcal{PT}}[x, \lambda_x, \psi_x, \eta_x] = \infty$. Otherwise,*

$$T^{\mathcal{PT}}[x, \lambda_x, \psi_x, \eta_x] = \sum_{z \leq_\Gamma x} \mathrm{cost}_f(z)$$

Proof Sketch. Let us abbreviate $Z_i := \bigcup_{j \leq i} V(\Gamma_{v_j})$. We first show that the table Q does what we expect it to do.

▷ Claim 27. Let $\lambda, \psi : \mathrm{YW}_x^\Gamma \cup \{x\} \to 2^C$ and $\eta : \mathrm{YW}_x^\Gamma \cup \{x\} \to \mathbb{N}$ such that $\psi \trianglelefteq \lambda \trianglelefteq \phi$. Let $f_i \in \mathcal{LF}_{N,\phi}$ have minimum cost on $\bigcup_{j \leq i} \Gamma_{v_j}$ among all lineage functions for N that, for all $w \in \bigcup_{j \leq i} \mathrm{YW}_{v_j}^\Gamma$, satisfy

(a) $\lambda(w) = f_i(w)$,
(b) $\psi(w) = f_i(w) \cap \bigcup_{j \leq i} \bigcup_{u \in \mathrm{Pred}_N^{\uparrow v_j}(w)} f_i(u)$, and
(c) $\eta(w) \leq \sum_{j \leq i} \sum_{u \in \mathrm{Pred}_N^{\uparrow v_j}(w)} |f_i(u)|$

Then, $Q_x^\lambda[i, \psi, \eta] = \sum_{j \leq i} \sum_{u \in \Gamma_{v_j}} \mathrm{cost}_{f_i}(u)$.

Proof Sketch. For "$\geq$", let $\psi' \trianglelefteq \psi\,|_{\mathrm{YW}_{v_i}^\Gamma}$ and $\eta' \trianglelefteq \eta\,|_{\mathrm{YW}_{v_i}^\Gamma}$ such that equality holds in (7). Let $f_{i-1} \in \mathcal{LF}_{N,\phi}$ minimize $\sum_{j<i} \sum_{u \in \Gamma_{v_j}} \mathrm{cost}_{f_{i-1}}(u)$ among all lineage functions satisfying (a)–(c) for $i-1$. Let $f^* \in \mathcal{LF}_{N,\phi}$ minimize $\sum_{u \in \Gamma_{v_i}} \mathrm{cost}_{f^*}(u)$ among all lineage functions that, for all $w \in \mathrm{YW}_{v_i}^\Gamma$, satisfy $\lambda(w) = f^*(w)$, $\psi'(w) = f^*(w) \cap \bigcup_{u \in \mathrm{Pred}_N^{\uparrow v_i}(w)} f^*(u)$ and $\eta'(w) = \sum_{u \in \Gamma_{v_i}} |f^*(u)|$. By induction hypotheses, the cost of f_{i-1} on Z_i is $Q_x^\lambda[i-1, \psi - \psi', \eta - \eta']$ and the cost of f^* on Γ_{v_i} is $T^{\mathcal{PT}}\left[v_i, \lambda\,|_{\mathrm{YW}_{v_i}^\Gamma}, \psi', \eta'\right]$. From f_{i-1} and f^*, we construct a lineage function $f' \in \mathcal{LF}_{N,\phi}$ whose cost on Z_i is $\sum_{j<i} \sum_{u \in \Gamma_{v_j}} \mathrm{cost}_{f_{i-1}}(u) + \sum_{u \in \Gamma_{v_i}} \mathrm{cost}_{f^*}(u)$. Then, "$\geq$" follows by optimality of f_i on Z_i.

For "$\leq$", let ψ' and η' be such that, for all $w \in \mathrm{YW}_{v_i}^\Gamma$, we have $\psi'(w) = f_i(w) \cap \bigcup_{u \in \mathrm{Pred}_N^{\uparrow v_i}(w)} f_i(u) \subseteq \psi(w)$ and $\eta'(w) = \sum_{u \in \mathrm{Pred}_N^{\uparrow v_i}(w)} |f_i(u)|$. By independence of subsolutions, f_i is optimal on Z_{i-1} and on Γ_{v_i} so, by induction hypotheses, the cost of f_i on Z_{i-1} is $Q_x^\lambda[i-1, \psi - \psi', \eta - \eta']$ and the cost of f_i on Γ_{v_i} is $T^{\mathcal{PT}}\left[v_i, \lambda\,|_{\mathrm{YW}_{v_i}^\Gamma}, \phi', \eta'\right]$. Since ψ' and η' are only one of the possible choices for the minimum in (7), "$\leq$" follows. ◁

For "$\geq$", let $D \subseteq U \subseteq \phi(x)$ such that equality holds in (6). We construct a lineage function f' that assigns $f'(x) = U$ and such that the lineages of D are inherited from parents of x (in N) that are below x in Γ. To this end, we ask the dynamic programming table for the cost of a lineage function that is optimal on Z_t and such that

1. $\psi'(x) = D$ *(lineages in D are inherited from parents of x in Γ_x)*
2. $\psi'(w) = \psi'(w) \setminus U$ for all $w \in \mathrm{Succ}_N^\uparrow(x)$ *(children of x in YW_x^Γ no longer need to inherit the lineages in U from Γ_x)*
3. $\eta'(x) = |U| \dot{-} \sum_{u \in \mathrm{Pred}_N^\downarrow(x)} |\lambda_x(u)|$ *(x needs to inherit $|U|$ lineages in total: $|\lambda_x(u)|$ come from every parent u of x in YW_x^Γ while the rest has to be inherited from Γ_x)* and
4. $\eta'(w) = \eta_x(w) \dot{-} |U|$ for all $w \in \mathrm{Succ}_N^\uparrow(x)$ *(children of x in YW_x^Γ can inherit a maximum of $|U|$ lineages from x).*

Since the functions $\lambda' := \lambda_x[x \to U]$, $\psi' := \psi_x\left[x \to D, \forall_{u \in \mathrm{Succ}^{\uparrow}_N(x)} w \to \psi_x(w) \setminus U\right]$ and $\eta' := \eta_x\left[x \to |U| \dot{-} \sum_{u \in \mathrm{Pred}^{\downarrow}_N(x)} |\lambda_x(u)|, \forall_{u \in \mathrm{Succ}^{\uparrow}_N(x)} w \to \eta_x(w) \dot{-} |U|\right]$ satisfy the conditions of Claim 27, the optimal cost of such a lineage function f' on Z_t is $Q^{\lambda}_x[t, \psi', \eta']$. Further, the cost of f' on x is the number of lineages in U that is not inherited "for free" from parents of x, that is, $|U \setminus (D \cup \bigcup_{u \in \mathrm{Pred}^{\downarrow}_N(x)} \lambda_x(u))|$. Then, "$\geq$" follows by optimality of f on Γ_x.

For "$\leq$", let $U := f(x)$ and let $D := U \cap \bigcup_{u \in \mathrm{Pred}^{\uparrow}_N(x)} f(x)$ be the set of lineages of U that are inherited from parents of x in N that are below x in Γ. By independence of sub-solutions, f is optimal on Z_t so, by Claim 27, its cost on Z_t is $Q^{\lambda}_x[t, \psi', \eta']$ where $\psi' := \psi_x[\ldots]$ and $\eta' := \eta_x[\ldots]$ are defined as in (6) and its cost on x is $|f(x) \setminus (\bigcup_{u \in \mathrm{Pred}^{\uparrow}_N(x)} f(x) \cup \bigcup_{u \in \mathrm{Pred}^{\downarrow}_N(x)} f(x))| = |U \setminus (D \cup \bigcup_{\mathrm{Pred}^{\downarrow}_N(x)} f(x))|$. Then, "$\leq$" follows from the fact that U and D are only one of the possible choices for the minimum in (6). ◀

To solve the parental parsimony problem given N, ϕ and Γ, we compute $T^{\mathcal{PT}}[x, \lambda_x, \psi_x, \eta_x]$ for each x bottom-up in Γ, each $\psi_x, \lambda_x : \mathrm{YW}^{\Gamma}_x \to 2^C$ with $\psi_x \trianglelefteq \lambda_x \trianglelefteq \phi$ and each $\eta_x : \mathrm{YW}^{\Gamma}_x \to \{0, \ldots, |C|\}$ (by Definition 24, no value larger than $|C|$ ever enters η_x and all modifications to η_x decrease the mapped-to values). To this end, $Q^{\lambda}_x[i, \psi, \eta]$ is computed for each x, i, λ, ψ, and η by making at most $2^{|C| \cdot |\mathrm{YW}^{\Gamma}_x|} \cdot |C|^{|\mathrm{YW}^{\Gamma}_x|}$ queries to $Q^{\psi_x}_{x, c_x}$ and $T^{\mathcal{PT}}$. As there are $O(|A(N)|)$ valid combinations of x and i, the table Q can be computed in $O(|A(N)| \cdot 3^{|C| \cdot \mathrm{yw}(N)} \cdot |C|^{\mathrm{yw}(N)} \cdot 2^{|C| \cdot \mathrm{yw}(N)} \cdot |C|^{\mathrm{yw}(N)}) = O(|A(N)| \cdot 6^{|C| \cdot \mathrm{yw}(N)|} \cdot 4^{\mathrm{yw}(N) \cdot \log |C|})$ time. Further, computing each $T^{\mathcal{PT}}[x, \lambda_x, \psi_x, \eta_x]$ requires testing $3^{|\phi(x)|} \leq 3^{|C|}$ choices for $D \subseteq U \subseteq \phi(x)$ and computing $|U \setminus (D \cup \bigcup_{u \in \mathrm{Pred}^{\downarrow}_N(x)} \lambda_x(u))|$ in $O(|C|)$ time (we precompute $\bigcup_{u \in \mathrm{Pred}^{\downarrow}_N(x)} \lambda_x(u)$ for each fix x and λ_x). Thus, the table $T^{\mathcal{PT}}$ can be computed in $O(3^{|C| \cdot \mathrm{yw}(N)} \cdot (|C|^{\mathrm{yw}(N)+1} \cdot 3^{|C|} + |A(N)|))$ time, which is dominated by the construction of Q.

▶ **Theorem 28.** *Given a network N, $\phi : V(N) \to 2^C$ and a tree Γ agreeing with N, the parental parsimony score of (N, ϕ) can be computed in $O(6^{\mathrm{yw}(\Gamma) \cdot |C|} \cdot 4^{\mathrm{yw}(\Gamma) \cdot \log |C|} \cdot |A(N)|)$ time.*

Again, we can replace $\mathrm{yw}(\Gamma)$ by $\mathrm{tw}(N)$ using Proposition 12.

▶ **Corollary 29.** *Let (N, ϕ) be an instance of* Parental Parsimony. *Let $t \geq \mathrm{tw}(N)$ and let T be the time in which a width-t tree decomposition of N can be computed. Then, the parental parsimony score of (N, ϕ) can be computed in $O(T + 6^{t \cdot |C|} \cdot 4^{t \cdot \log |C|} \cdot |A(N)|)$ time.*

Note that the parental parsimony setting supports assigning multiple states of a character to a single species, thereby modeling species carrying multiple alleles of a single gene. By forcing $D \subseteq U = \phi(x)$ instead of $D \subseteq U \subseteq \phi(x)$ if x is a leaf, we can trivially modify our dynamic programming to explain multiple character states in extant species.

Corollaries 18, 23 and 29 give the running times of our algorithms as depending on the treewidth of N. The state-of-the-art solutions for Hardwired Parsimony, Softwired Parsimony and Parental Parsimony have the following respective running times: $O(|C|^{r+2}|V(N)|)$ [21], $O(2^{\ell}|C|^2|V(N)||A(N)|)$ [13] and $O(|2^C|^{\ell+3}|V(N)|)$ [29]. Since the scanwidth of N is potentially much smaller than its level ℓ [27], and the treewidth of N is smaller than its scanwidth [3], we have $\mathrm{tw}(N) - 1 \leq \ell \leq r$. Thus, we expect that there will be several cases where our algorithms will be faster than the current best-known ones.

5 Discussion

In this paper, we focused on the small version of the parsimony problem for networks given a specific position in the genome. When markers can be assumed to be independent, as it is the case when a certain distance is preserved between genomic locations included in the matrix, each position can be analyzed separately, and the parsimony score of a network w.r.t. the

matrix is simply the sum of the parsimony scores of the network for each genomic location. Thus, the algorithms presented here can be easily expanded to several independent genomic locations. Moreover, our formulations are defined for networks that are not necessarily binary, can account for polymorphism and can impose restrictions on ancestral states. As discussed above, our algorithms can be orders of magnitude faster than the state-of-the-art solutions. A comparison of the reticulation number, the level, the scanwidth and the treewidth for practically relevant classes of networks would thus be an interesting project for future work.

Our results are slightly overshadowed by the fact that optimal tree decompositions are very hard to compute, with even the best-known parameterized algorithm being considered impractical (see survey [5]). However, the treewidth can be 2-approximated in single-exponential time [24] and, with development driven by recent issues of the PACE challenge [10], more practical exact algorithms are now available as well [28]. We would welcome similar efforts also for the scanwidth, which is also hard to compute [3].

The ability to fast-score phylogenetic networks under the parsimony framework could be a big help in designing likelihood-based heuristics or bayesian methods to infer networks from independent markers [31, 27] by providing fast heuristics to compute the initial networks with which to start the likelihood or bayesian search, or to design fast local-search techniques.

In the future, we would like to tackle the SMALL PARSIMONY problem for several *dependent* genomic locations (e.g. a gene). Little is known for this problem, except that it stays NP-hard even for binary characters even on level-1 networks [22] and that it is fixed-parameter tractable in the number of reticulations of the network [26]. Another important direction would be to study the BIG PARSIMONY problem, which is currently wide open, even lacking a consensus of the definition of optimality [26, 17, 30, 6].

References

1 Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey. *BIT Numerical Mathematics*, 25(1):1–23, 1985.

2 Various Authors. The graph parameter hierarchy. Available at `https://gitlab.com/gruenwald/parameter-hierarchy`, 2021.

3 Vincent Berry, Celine Scornavacca, and Mathias Weller. Scanning phylogenetic networks is NP-hard. In *Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'20)*, pages 519–530. Springer, 2020.

4 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

5 Hans L. Bodlaender. Discovering treewidth. In *Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, pages 1–16, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

6 Christopher Bryant, Mareike Fischer, Simone Linz, and Charles Semple. On the quirks of maximum parsimony and likelihood on phylogenetic networks. *Journal of Theoretical Biology*, 417:100–108, 2017.

7 David Bryant and Jens Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351(3):296–302, 2006.

8 Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: an overview. *Algorithms*, 12(12):256, 2019.

9 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

10 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

11 Nick D. Dendris, Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1):233–254, 1997.

12 Joseph Felsenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

13 Mareike Fischer, Leo Van Iersel, Steven Kelk, and Celine Scornavacca. On computing the maximum parsimony score of a phylogenetic network. *SIAM Journal on Discrete Mathematics*, 29(1):559–585, 2015.

14 Walter M Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971.

15 Jotun Hein. Reconstructing evolution of sequences subject to recombination using parsimony. *Mathematical Biosciences*, 98(2):185–200, 1990.

16 Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

17 G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Inferring phylogenetic networks by the maximum parsimony criterion: A case study. *Molecular Biology and Evolution*, 24(1):324–337, 2006.

18 G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Maximum likelihood of phylogenetic networks. *Bioinformatics*, 22(21):2604–2611, 2006.

19 Guohua Jin, L. Nakhleh, S. Snir, and T. Tuller. Parsimony score of phylogenetic networks: Hardness results and a linear-time heuristic. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3):495–505, 2009.

20 Lavanya Kannan and Ward C. Wheeler. Maximum Parsimony on Phylogenetic networks. *Algorithms for Molecular Biology*, 7(1):9, 2012.

21 Lavanya Kannan and Ward C. Wheeler. Exactly computing the parsimony scores on phylogenetic networks using dynamic programming. *Journal of Computational Biology*, 21(4):303–319, 2014.

22 Steven Kelk, Fabio Pardi, Celine Scornavacca, and Leo van Iersel. Finding a most parsimonious or likely tree in a network with respect to an alignment. *Journal of Mathematical Biology*, 78(1-2):527–547, 2019.

23 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

24 Tuukka Korhonen. Single-exponential time 2-approximation algorithm for treewidth. *CoRR*, abs/2104.07463, 2021. `arXiv:2104.07463`.

25 Guillaume Mescoff, Christophe Paul, and Dimitrios Thilikos. A polynomial time algorithm to compute the connected tree-width of a series-parallel graph, 2021. `arXiv:2004.00547v5`.

26 Luay Nakhleh, Guohua Jin, Fengmei Zhao, and John Mellor-Crummey. Reconstructing phylogenetic networks using maximum parsimony. In *2005 IEEE Computational Systems Bioinformatics Conference (CSB'05)*, pages 93–102. IEEE, 2005.

27 Charles-Elie Rabier, Vincent Berry, Marnus Stoltz, João D. Santos, Wensheng Wang, Glaszmann Jean-Christophe, Fabio Pardi, and Celine Scornavacca. On the inference of complicated phylogenetic networks by Markov Chain Monte-Carlo. Submitted.

28 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

29 Leo Van Iersel, Mark Jones, and Celine Scornavacca. Improved maximum parsimony models for phylogenetic networks. *Systematic Biology*, 67(3):518–542, 2018.

30 Ward C Wheeler. Phylogenetic network analysis as a parsimony optimization problem. *BMC Bioinformatics*, 16(1):1–9, 2015.

31 Jiafan Zhu, Dingqiao Wen, Yun Yu, Heidi M Meudt, and Luay Nakhleh. Bayesian inference of phylogenetic networks from bi-allelic genetic markers. *PLoS Computational Biology*, 14(1):e1005932, 2018.

32 Jiafan Zhu, Yun Yu, and Luay Nakhleh. In the light of deep coalescence: revisiting trees within networks. *BMC Bioinformatics*, 17(14):271–282, 2016.

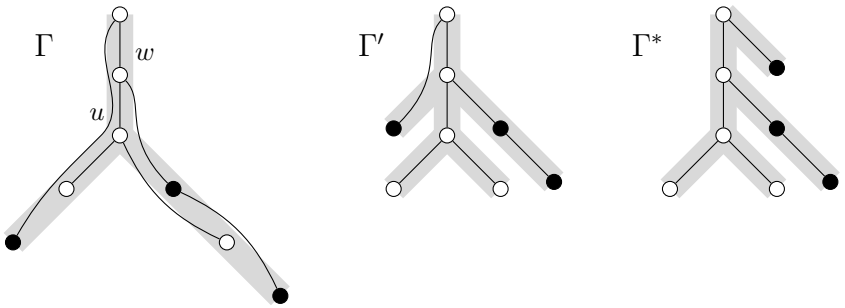


Figure 5 Example for the construction of Γ' (middle) from Γ (left) in Lemma 7. Repeated application yields Γ^* (right), for which $v \leq_{\Gamma^*} u \Rightarrow u \rightsquigarrow_{G,\Gamma^*} v$. The rooted trees Γ, Γ', and Γ^* are drawn with thick, gray lines. Thin, black lines are edges of G. For the indicated node u, the black nodes are in X, that is, they are below u in Γ but not connected to u in $G[\Gamma_u]$.

A Proofs of results in the main text

A.1 Proof of Lemma 4

Proof. (a), (b): We show for all vertices w on a u-v-path p in Γ^σ that $w \leq_\sigma u$ and $u \rightsquigarrow_{G,\sigma} w$. The base case $w = u$ holds trivially. For the induction step, let q preceed w in p. Since Γ^σ contains the arc qw, Definition 1 implies $q \rightsquigarrow_{G,\sigma} w$ and, since $q \leq_\sigma u$ by induction hypothesis, $w \leq_\sigma q \leq_\sigma u$ and $u \rightsquigarrow_{G,\sigma} w$. For the reverse direction of (b), note that, by Definition 1, uv is an arc of the DAG whose transitive reduction Γ^σ is.

(c),(d): Since $G[\sigma[1..r]] = G$ and G is connected, there is an r-x-path in $G[\sigma[1..r]]$ for all $x \in V(G)$ and, thus, Γ^σ is connected and rooted at r.

(e): To prove that Γ^σ is a tree, assume there is a vertex $x \in V(G)$ with two distinct parents y and z in Γ^σ. Without loss of generality, let $y <_\sigma z$. By (b), $y \rightsquigarrow_{G,\sigma} x$ and $z \rightsquigarrow_{G,\sigma} x$. Since $\sigma[1..y] \subsetneq \sigma[1..z]$, we conclude $z \rightsquigarrow_{G,\sigma} y$, implying $zy \in A(\Gamma^\sigma)$ and contradicting Γ^σ being a transitive reduction.

(f): Note that $u \rightsquigarrow_{G,\sigma} v$, implying $v \leq_{\Gamma^\sigma} u$ by (b).

(g): For each $uv \in E(G)$, either $u <_\sigma v$, implying $u \leq_{\Gamma^\sigma} v$, or $v <_\sigma u$, implying $v \leq_{\Gamma^\sigma} u$ (both by (f)).

(h) "$\subseteq$": Let $x \in V(G)$ and let $y \in \mathrm{YW}_x^{\Gamma^\sigma}$. By Definition 2, $y >_{\Gamma^\sigma} x$ (implying $y >_\sigma x$ by (a)) and there is some $z \leq_{\Gamma^\sigma} x$ (implying $z \leq_\sigma x$ by (a)) with $yz \in E(G)$. Then, by (b), $x \rightsquigarrow_{G,\sigma} z$. But then, $y \in \mathrm{ZW}_x^\sigma$ by Definition 1.

(h) "$\supseteq$": Let $x \in V(G)$ and let $y \in \mathrm{ZW}_x^{\Gamma^\sigma}$, that is, $x <_\sigma y$ and there is some $z \in \sigma[1..x]$ with $x \rightsquigarrow_{G,\sigma} z$ and $yz \in E(G)$. Then, $z \leq_\sigma x <_\sigma y$. By (b), $z \leq_{\Gamma^\sigma} x$ and, by (f), $z \leq_{\Gamma^\sigma} y$. Thus, as Γ^σ is a tree (by (e)), x and y are not unrelated in Γ^σ. Moreover, $y \not\leq_\sigma x$ implies $y \not\leq_{\Gamma^\sigma} x$ by (b) and, thus, $x <_{\Gamma^\sigma} y$. Together with $z \leq_{\Gamma^\sigma} x$ and $yz \in E(G)$, this implies $y \in \mathrm{YW}_x^{\Gamma^\sigma}$. ◀

A.2 Proof of Lemma 7

(See Figure 5).

Proof. Let $u \in V(G)$ such that $X := \{v <_\Gamma u \mid u \not\rightsquigarrow_{G,\Gamma} v\} \neq \varnothing$. We will modify Γ into Γ' with $\mathrm{yw}(\Gamma') \leq \mathrm{yw}(\Gamma)$ such that Γ' agrees with G and the relation $\leq_{\Gamma'}$ is a strict subset of $\leq_\Gamma$. To this end, note that u has a parent w in Γ as, otherwise, $G[\Gamma_u] = G$, implying $X = \varnothing$. Then, Γ' results from Γ by

1. replacing Γ by $\Gamma \uparrow (\Gamma_u \setminus X)$ and
2. dangling $\Gamma_u \uparrow X$ from w.

First, we show that Γ' agrees with G. To this end, let $xy \in E(G)$ and let x and y be unrelated in Γ'. If neither x nor y are in Γ_u then, by construction of Γ', they are also unrelated in Γ, contradicting that Γ agrees with G. So, without loss of generality, suppose $x \leq_\Gamma u$. Since $xy \in E(G)$ and Γ is a tree agreeing with G, we thus know that u and y are not unrelated in Γ. If $u <_\Gamma y$, then $w \leq_\Gamma y$ and, thus, $x \leq_{\Gamma'} y$. Thus, suppose $y \leq_\Gamma u$. Clearly, if $x, y \in X$ or $x, y \notin X$, then x and y are also unrelated in Γ, contradicting its agreement with G. Thus, without loss of generality, suppose $x \in X$ and $y \notin X$, that is, $u \not\rightsquigarrow_{G,\Gamma} x$ and $u \rightsquigarrow_{G,\Gamma} y$, contradicting $xy \in E(G)$.

Second, we show that $\leq_{\Gamma'}$ is a strict subset of $\leq_\Gamma$. To this end, let $xy \in A(\Gamma')$ and assume towards a contradiction that $y \not<_\Gamma x$. Clearly, if $x \not\leq_{\Gamma'} w$, then $xy \in A(\Gamma)$ contradicting $y \not<_\Gamma x$. Further, if $x = w$, then either $y \in X$ or y is a child of w in Γ, all of which imply $y <_\Gamma x$. Thus, $x <_{\Gamma'} w$. Since $xy \cap X = \{x\}$ or $xy \cap X = \{y\}$ contradicts $xy \in A(\Gamma')$, we have $x, y \in X$ or $x, y \notin X$. But then, $y <_\Gamma x$ by Observation 6. Thus, $\leq_{\Gamma'}$ is a subset of $\leq_\Gamma$ and it is strict since we have $v \leq_\Gamma u$ and $v \not\leq_{\Gamma'} u$ for all $v \in X \neq \varnothing$.

Third, $\operatorname{yw}(\Gamma') \leq \operatorname{yw}(\Gamma)$ follows by Lemma 3. ◀

A.3 Proof of Lemma 9

Proof. "$\geq$": Let σ be an ordering of $V(G)$ such that $\operatorname{zw}(\sigma) = \operatorname{zw}(G)$. By Lemma 4(h), we have $\operatorname{zw}(\sigma) = \operatorname{yw}(\Gamma^\sigma)$ for the canonical extension tree Γ^σ of σ. Thus, $\operatorname{zw}(G) = \operatorname{zw}(\sigma) = \operatorname{yw}(\Gamma^\sigma) \geq \operatorname{yw}(G)$.

"$\leq$": Let Γ be some rooted tree agreeing with G such that $\operatorname{yw}(\Gamma) = \operatorname{yw}(G)$ and, by Lemma 7, suppose

$$u \leq_\Gamma v \Rightarrow v \rightsquigarrow_{G,\Gamma} u. \tag{8}$$

Let σ be any ordering of $V(G)$ obtained by repeatedly picking and removing any leaf of Γ.

▷ Claim 30. For each $u, v \in V(G)$, we have $u \leq_\Gamma v$ if and only if $v \rightsquigarrow_{G,\sigma} u$.

Proof. First, note that all nodes below v in Γ are chosen before v, so $\Gamma_v \subseteq \sigma[1..v]$.

"$\Rightarrow$": Let $u \leq_\Gamma v$, that is, $u \in \Gamma_v$, implying $u \leq_\sigma v$. By (8), v is connected to u in $G[\Gamma_v]$ and, as $\Gamma_v \subseteq \sigma[1..v]$, also in $G[\sigma[1..v]]$.

"$\Leftarrow$": Let p be a v-u-path in $G[\sigma[1..v]]$. By Lemma 8, p has a unique maximum w in Γ. Hence, $v \leq_\Gamma w$ and, by "$\Rightarrow$", we have $v \leq_\sigma w$. Since p lives entirely in $G[\sigma[1..v]]$, that is, $V(p) \subseteq \sigma[1..v]$, we also have $w \leq_\sigma v$. Thus, $v = w$ and, since $u \in V(p)$, we have $u \leq_\Gamma w = v$ by maximality of w. ◁

To prove the lemma, we show $\mathrm{YW}_x^\Gamma \supseteq \mathrm{ZW}_x^\sigma$ for each $x \in V(G)$. Let $y \in \mathrm{ZW}_x^\sigma$, that is $y >_\sigma x$ and there is some $z \in \sigma[1..x]$ with $yz \in E(G)$ and $x \rightsquigarrow_{G,\sigma} z$. By Claim 30, $z \leq_\Gamma x$. Further, as $yz \in E(G)$ and Γ agrees with G, y and z are not unrelated in Γ and, since $z \leq_\Gamma x$, neither are x and y. Since $y <_\Gamma x$ implies $y <_\sigma x$ by Claim 30, contradicting $y >_\sigma x$, we conclude $x <_\Gamma y$. Together with $z \leq_\Gamma x$ and $yz \in E(G)$, this implies $y \in \mathrm{YW}_x^\Gamma$. ◀

A.4 Proof of Proposition 12

Proof. "$\leq$": Let (T, B) be a nice tree decomposition for G of width $\operatorname{tw}(G)$ and let $F \subset V(T)$ denote the set of all "forget"-nodes in T (noting that the root of T is in F). We construct Γ from T by contracting all nodes in $V(T) \setminus F$ onto their respective parents[1] and identifying all nodes $x \in F$ with the vertex $v \in V(G) \setminus B(x)$ of G that is forgotten in x. By Observation 11, $V(\Gamma) = V(G)$.

First, we show that Γ agrees with G. To this end, let $uv \in E(G)$ and let $f_u, f_v \in V(T)$ denote the unique "forget u" and "forget v"-nodes in T, which are distinct since T is nice. By Definition 10(a), there is a node $q \in V(T)$ with $uv \subseteq B(q)$ and, by Observation 11, $q <_T f_u, f_v$. Thus, f_u and f_v are not unrelated in T and, by Observation 5, neither in Γ.

Second, we show for all $v \in \Gamma$ and the unique "forget v"-node f_v in T that $\mathrm{YW}_v^\Gamma \subseteq B(f_v)$. Let $u \in \mathrm{YW}_v^\Gamma$, that is, $u >_\Gamma v$ and there is some $w \leq_\Gamma v$ with $uw \in E(G)$ (note that $w \neq u$ but $w = v$ is possible). Let f_u and f_w be the unique "forget u" and "forget w"-nodes in T, which are distinct since T is nice. Then, $w \leq_\Gamma v <_\Gamma u$ and, by Observation 5, $f_w \leq_T f_v <_T f_u$. Since $uw \in E(G)$, Definition 10(a) implies that there is a node q of T with $uw \subseteq B(q)$, implying $q <_T f_u, f_w$. Then, by Definition 10(b), $u \in B(x)$ for all x with $q \leq_T x <_T f_u$ and, since $q <_T f_w <_T f_v <_T f_u$, we have $u \in B(f_v)$. Thus, $\mathrm{YW}_v^\Gamma \subseteq B(f_v)$, implying $\operatorname{yw}(G) \leq \mathrm{YW}_v^\Gamma \leq |B(f_v)|$ and, since f_v has a child x with $B(x) = B(f_v) \cup \{v\}$, we know $|B(f_v)| = |B(x)| - 1 \leq \operatorname{tw}(T, B) = \operatorname{tw}(G)$.

"$\geq$": Let Γ be a tree with $\operatorname{yw}(\Gamma) = \operatorname{yw}(G)$ that agrees with G. For all $u \in V(G)$, we define $B(u) := \mathrm{YW}_u^\Gamma \cup \{u\}$ and show that (Γ, B) is a tree-decomposition for G noting that its width is $\operatorname{yw}(\Gamma) = \operatorname{yw}(G)$.

First, to prove Definition 10(a), let $uv \in E(G)$. Since Γ agrees with G, either $u <_\Gamma v$ or $v <_\Gamma u$. Without loss of generality, suppose the latter. Then, $u \in \mathrm{YW}_v^\Gamma$ by Definition 2 (using $w = v$), implying that $uv \in B(v)$.

Second, let $u, v \in V(G)$ be distinct such that $u \in B(v) = \mathrm{YW}_v^\Gamma \cup \{v\}$, implying $u \in \mathrm{YW}_v^\Gamma$ since $u \neq v$. By Definition 2, there is some $w \leq_\Gamma v$ with $uw \in E(G)$ and $v <_\Gamma u$, implying that Γ contains a unique u-v-path p. To show Definition 10(b), it suffices to prove $u \in B(x)$ for all $x \in V(p)$ (since v has been chosen arbitrarily, a path with these properties exists for all v' with $u \in B(v')$, so they all contain the node u and are, thus, connected). For $x = u$ this follows by definition of $B(u)$. Otherwise, $x <_\Gamma u$ since $x \in V(p)$. But then, $w \leq_\Gamma v \leq_\Gamma x <_\Gamma u$ and $uw \in E(G)$, implying $u \in \mathrm{YW}_x^\Gamma \subseteq B(x)$. ◀

A.5 Proof of Lemma 14

Proof. Towards a contradiction, assume that the lemma is false. We construct $\psi^* : V(N) \to C$ with

$$\psi^*(u) = \begin{cases} \psi_y(u) & \text{if } u \in \Gamma_y \text{ for any } y \in Y \\ \psi(u) & \text{otherwise} \end{cases}$$

Note that ψ^* and ψ coincide with ψ_y on YW_y^Γ for all $y \in Y$. Thus, $\delta_{\psi^*}(u, w) = \delta_{\psi_y}(u, w)$ if $uw \in A_y(S^*)$ for any $y \in Y$ and $\delta_{\psi^*}(u, w) = \delta_\psi(u, w)$, otherwise. Further, we construct a digraph $S^* := (V(N), (A(S) \setminus \bigcup_{y \in Y} A_y(S)) \cup \bigcup_{y \in Y} A_y(S^y))$ which is in $\mathcal{G}$ since $(S^y)_{y \in Y}$ is a Y-substitution system for $\mathcal{G}$. Since all S^y are subnetworks of N, we know that Γ agrees with S^*. Furthermore, since $Y \subseteq \bigcup_{z \in Z} \Gamma_z$, we know that each $y \in Y$ has a $z \in Z$ with $y \leq_\Gamma z$. Thus,

[1] One can also describe Γ as the transitive reduction of $(F, >_T \cap (F \times F))$.

$$\begin{aligned}
\sum_{z\in Z}\sum_{uw\in A_z(S^*)}\delta_{\psi^*}(u,w) &= \sum_{z\in Z}\sum_{v\in\Gamma_z}\sum_{uw\in A_{\{v\}}(S^*)}\delta_{\psi^*}(u,w)\\
&= \sum_{z\in Z}\sum_{\substack{v\in\Gamma_z\\ v\notin\bigcup_{y\in Y}\Gamma_y}}\sum_{uw\in A_{\{v\}}(S^*)}\delta_{\psi^*}(u,w) + \sum_{y\in Y}\sum_{uw\in A_y(S^*)}\delta_{\psi^*}(u,w)\\
&= \sum_{z\in Z}\sum_{\substack{v\in\Gamma_z\\ v\notin\bigcup_{y\in Y}\Gamma_y}}\sum_{uw\in A_{\{v\}}(S)}\delta_{\psi}(u,w) + \sum_{y\in Y}\sum_{uw\in A_y(S^y)}\delta_{\psi_y}(u,w)\\
&\overset{\text{assumption}}{<} \sum_{z\in Z}\sum_{\substack{v\in\Gamma_z\\ v\notin\bigcup_{y\in Y}\Gamma_y}}\sum_{uw\in A_{\{v\}}(S)}\delta_{\psi}(u,w) + \sum_{y\in Y}\sum_{uw\in A_y(S)}\delta_{\psi}(u,w)\\
&= \sum_{z\in Z}\sum_{uw\in A_z(S)}\delta_{\psi}(u,w)
\end{aligned}$$

contradicting optimality of S and ψ (that is, Lemma 14(a) since $S^* \in \mathcal{G}$. ◀

Tree Diet: Reducing the Treewidth to Unlock FPT Algorithms in RNA Bioinformatics

Bertrand Marchand
LIX CNRS UMR 7161, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-vallée, France

Yann Ponty[1]
LIX CNRS UMR 7161, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Laurent Bulteau[1]
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-vallée, France

Abstract

Hard graph problems are ubiquitous in Bioinformatics, inspiring the design of specialized Fixed-Parameter Tractable algorithms, many of which rely on a combination of tree-decomposition and dynamic programming. The time/space complexities of such approaches hinge critically on low values for the treewidth tw of the input graph. In order to extend their scope of applicability, we introduce the Tree-Diet problem, *i.e.* the removal of a minimal set of edges such that a given tree-decomposition can be slimmed down to a prescribed treewidth tw'. Our rationale is that the time gained thanks to a smaller treewidth in a parameterized algorithm compensates the extra post-processing needed to take deleted edges into account.

Our core result is an FPT dynamic programming algorithm for Tree-Diet, using $2^{O(tw)}n$ time and space. We complement this result with parameterized complexity lower-bounds for stronger variants (e.g., NP-hardness when tw' or $tw-tw'$ is constant). We propose a prototype implementation for our approach which we apply on difficult instances of selected RNA-based problems: RNA design, sequence-structure alignment, and search of pseudoknotted RNAs in genomes, revealing very encouraging results. This work paves the way for a wider adoption of tree-decomposition-based algorithms in Bioinformatics.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Parameterized complexity and exact algorithms; Applied computing → Bioinformatics

Keywords and phrases RNA, treewidth, FPT algorithms, RNA design, structure-sequence alignment

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.7

Related Version *Full Version*: `https://hal.inria.fr/hal-03206132/`

Supplementary Material *Software (Source Code)*: `https://gitlab.inria.fr/amibio/tree-diet` archived at `swh:1:dir:14dddf3079b8dca1f2cbea271a110cfb5592d23c`

Funding This work was supported by the French Agence Nationale de la Recherche through the Decrypted collaborative project (ANR-19-CE30-0021).

Acknowledgements The authors would like to thank Julien Baste for pointing out prior work on treewidth modulators, and providing valuable input regarding vertex deletion problems.

1 Introduction

Graph models and parameterized algorithms are found at the core of a sizable proportion of algorithmic methods in bioinformatics addressing a wide array of subfields, spanning sequence processing [48], structural bioinformatics [50], comparative genomics [9], phylogenetics [2], and further examples that can be found in a review by Bulteau and Weller [10]. RNA bioinformatics is no exception, with the prevalence of the secondary structure, an outer

[1] To whom correspondence should be addressed.

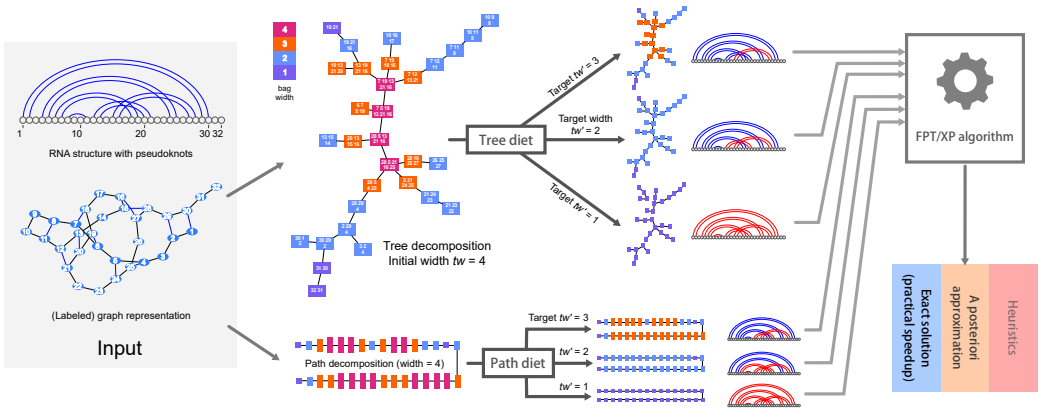


Figure 1 General description of our approach and rationale. Starting from a structured instance, e.g. an RNA structure with pseudoknots, our tree-diet/path-diet algorithms extract simplified tree/path decompositions, having prescribed target width tw'. Those can be used within existing parameterized algorithms to yield efficient heuristics, *a posteriori* approximations or even exact solutions.

planar graph [47], as an abstraction of RNA conformations, and the notable utilization of graph models to represent complex topological motifs called pseudoknots [49], inducing the hardness of several tasks, such as structure prediction [1, 29, 37], structure alignment [5], or structure/sequence alignment [32]. Such motifs are functionally important and conserved, as witnessed by their presence in the consensus structure of 336 RNA families in the 14.5 edition of the RFAM database [23]. Moreover, methods in RNA bioinformatics [36] are increasingly considering non-canonical base pairs and modules [25, 31], further increasing the density of RNA structural graphs and outlining the need for scalable algorithms.

A parameterized complexity approach can be used to circumvent the frequent NP-hardness of relevant problems. It generally considers one or several parameters, whose values are naturally bounded (or much smaller than the input size) within real-life instances. Once relevant parameters have been identified, one aims to design a Fixed Parameter Tractable (FPT) algorithm, having polynomial complexity for any fixed value of the parameter, and reasonable dependency on the parameter value. The treewidth is a classic parameter for FPT algorithms, and intuitively captures a notion of distance of the input to a tree. It is popular in bioinformatics due to the existence of efficient heuristics [19, 8] for computing tree-decompositions of reasonable treewidth. Given a tree-decomposition, many combinatorial optimization tasks can be solved using dynamic programming (DP), in time/space complexities that remain polynomial for any fixed treewidth value. Resulting algorithms remain correct upon (almost) arbitrary modifications of the objective function parameters, and can be adapted to study statistical properties of search spaces through changes of algebra.

Unfortunately, the existence of a parameterized (or FPT) algorithm does not necessarily imply that of a practically-efficient implementation, even when the parameter takes low typical values. Indeed, the dependency of the complexity on the treewidth may be prohibitive, both in terms of time and memory requirements. This limitation is particularly obvious while searching and aligning structured RNAs, giving rise to an algorithmic problem called the RNA structure-sequence alignment [39, 21, 32], for which the best known exact algorithm is in $\Theta(n.m^{tw+1})$, where n is the structure length, m the sequence/window, and tw is the treewidth of the structure (inc. backbone). Similar complexities hold for problems that can be expressed as (weighted) constraint satisfaction problems, with m representing the

cardinality of the variable domains. Such frameworks are frequently used for molecular design, both in proteins [44] and RNA [51], and may require the consideration of tree-widths up to 20 or more [20].

In this paper, we investigate a pragmatic strategy to increase the practicality of parameterized algorithms based on the treewidth parameter [6]. We put our instance graphs on a diet, *i.e.* we introduce a preprocessing that reduces their treewidth to a prescribed value by removing a minimal cardinality set of edges. As discussed previously, the practical complexity of many algorithms greatly benefits from the consideration of simplified instances, having lower treewidth. Moreover, specific countermeasures for errors introduced by the simplification can sometimes be used to preserve the correctness of the algorithm. For instance, searching structured RNAs using RNA structure-sequence alignment [39], an iterated filtering strategy could use instances of increasing treewidth to restrict potential hits, weeding them early so that a – costly – full structure is reserved to (quasi-)hits. This strategy could remain exact while saving substantial time. Alternative countermeasures could be envisioned for other problems, such as a rejection approach to correct a bias introduced by simplified instances in RNA design.

After stating our problem(s) in Section 2, we study in Section 3 the parameterized complexity of the Graph-Diet problem, the removal of edges to reach a prescribed treewidth. We propose, in Section 4,a practical Dynamic Programing FPT algorithm for Tree-Diet, along with possible further optimizations for Path-Diet, two natural simplifications of the Graph-Diet problem, where a tree (resp. path) decomposition is provided as input and used as a guide. Finally, we show in Section 5 how our algorithm can be used to extract hierarchies of graphs/structural models of increasing complexity to provide alternative sampling strategies for RNA design, and speed-up the search for pseudoknotted non-coding RNAs. We conclude in Section 6 with future considerations and open problems.

2 Statement of the problem(s) and results

A *tree-decomposition* $\mathcal{T}$ (over a set V of vertices) is a tree whose nodes are subsets of V, known as bags. The bags containing any $v \in V$ induce a (connected) subtree of $\mathcal{T}$. A *path-decomposition* is a tree-decomposition whose underlying tree $\mathcal{T}$ is a path. The *width* of $\mathcal{T}$ (denoted $w(\mathcal{T})$) is the size of its largest bag minus 1. An edge $\{u, v\}$ is *visible* in $\mathcal{T}$ if some bag contains both u and v, otherwise it is *lost*. $\mathcal{T}$ is a *tree-decomposition of* G if all edges of G are visible in $\mathcal{T}$. The *treewidth* of a graph G is the minimum width over all tree-decompositions of G.

▶ **Problem** (Graph-Diet). *Given a graph $G = (V, E)$ of treewidth tw, and an integer $tw' < tw$, find a tree-decomposition over V of width at most tw' losing a minimum number of edges from G.*

A *tree-diet* of $\mathcal{T}$ is any tree-decomposition $\mathcal{T}'$ obtained by removing vertices from the bags of $\mathcal{T}$. $\mathcal{T}'$ is a d-tree-diet if $w(\mathcal{T}') \leq w(\mathcal{T}) - d$.

▶ **Problem** (Tree-Diet). *Given a graph G, a tree-decomposition $\mathcal{T}$ of G of width tw, and an integer $tw' < tw$, find a $(tw - tw')$-tree-diet of $\mathcal{T}$ losing a minimum number of edges.*

Note that for Tree-Diet, $\mathcal{T}$ does not have to be optimal, so the width tw of the input tree decomposition might be larger than the actual treewidth of G, thus Tree-Diet can be used to reduce the width of *any* input decomposition. We define Binary-Tree-Diet and Path-Diet analogously, where $\mathcal{T}$ is restricted to be a binary tree (respectively, a path). An example of an instance of Graph-Diet and of Tree-Diet are given in Figure 2.

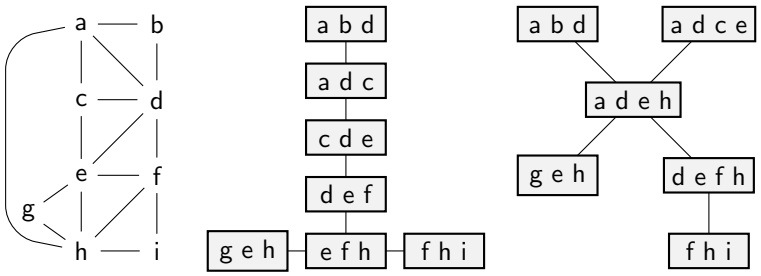


Figure 2 Illustrations for the Graph-Diet and Tree-Diet problems. Given a graph G on the left (treewidth 3), an optimal solution for Graph-Diet, with target treewidth 2, yields the tree-decomposition in the middle (edge ah is lost). On the other hand, any 1-tree-diet for the tree-decomposition on the right loses at least 3 edges.

Parameterized Complexity in a Nutshell

The basics of parameterized complexity can be loosely defined as follows (see [17] for the formal background). A *parameter* k for a problem is an integer associated with each instance which is expected to remain small in practical instances (especially when compared to the input size n). An exact algorithm, or the problem it solves, is FPT if it takes time $f(k)\text{poly}(n)$, and XP if it takes time $n^{g(k)}$ (for some functions f, g). Under commonly accepted conjectures (see for instance [15] for details), W[1]-hard problems may not be FPT, and Para-NP-hard problems (NP-hard even for some fixed value of k) are not FPT nor XP.

2.1 Our results

Our results are summarized in Table 1. Although the Graph-Diet problem would give the most interesting tree-decompositions in theory, it seems unlikely to admit efficient algorithms in practice (see Section 3).

Thus we focus on the Tree-Diet relaxation, where an input tree-decomposition is given, which we use as a guide/restriction towards a thinner tree-decomposition. Seen as an additional constraint, it makes the problem harder (the case $tw' = 1$ becomes NP-hard, Theorem 3, although for Graph-Diet it corresponds to the Spanning Tree problem and is polynomial). With parameter tw however, it does help reduce the search space. In Theorem 10 we give an $O((6\Delta)^{tw}\Delta^2 n)$ Dynamic Programming algorithm, where Δ is the maximum number of children of any bag in the tree-decomposition. This algorithm can thus be seen as XP in general, but FPT on bounded-degree tree-decompositions (e.g. binary trees and paths). This is not a strong restriction, since the input tree may safely and efficiently be transformed into a binary one (see Appendix A for more details). Moreover, the duplications of bags which are used in the conversion may only decease the number of lost edges incurred by Tree-Diet.

We also consider the case where the treewidth needs to be reduced by $d = 1$ only, this without constraining the source treewidth. We give a polynomial-time algorithm for Path-Diet in this setting (Theorem 13) which generalizes into an XP algorithm for larger values of d, noting that an FPT algorithm for d is out of reach by Theorem 5. We also show that the problem is Para-NP-hard if the tree degree is unbounded (Theorem 4).

Table 1 Parameterized results for our problems. Algorithm complexities are given up to polynomial time factors (O^* notation), Δ denotes the maximum number of children in the input tree-decomposition. (*) see Theorem 2 statement for a more precise formulation.

<table>
<tr><th>Parameter
Problem</th><th colspan="2">Source treewidth
tw</th><th>Target treewidth
tw'</th><th>Difference
$d = tw - tw'$</th></tr>
<tr><td>Graph-Diet</td><td colspan="2">open</td><td>Para-NP-hard
$tw' = 2$
EDP(K_4) [18]</td><td>Para-NP-hard*
$d = 1$
Theorem 2</td></tr>
<tr><td>Tree-Diet</td><td>XP
$O^*((6\Delta)^{tw})$
Theorem 10</td><td>FPT open</td><td rowspan="3">Para-NP-hard
$tw' = 1$
Theorem 3</td><td>Para-NP-hard
$d = 1$
Theorem 4</td></tr>
<tr><td>Binary-Tree-Diet</td><td colspan="2" rowspan="2">FPT
$O^*(12^{tw})$
Theorem 10</td><td rowspan="2">W[1]-hard
Theorem 5</td><td>XP open</td></tr>
<tr><td>Path-Diet</td><td>XP
$O^*(tw^d)$
Theorem 13</td></tr>
</table>

3 Algorithmic Limits: Parameterized Complexity Considerations

Graph-Diet can be seen as a special case of the Edge Deletion Problem (EDP) for the family of graphs $\mathcal{H}$ of treewidth tw' or less: given a graph G, remove as few edges as possible to obtain a graph in $\mathcal{H}$. Such edge modification problems are more often parameterized by the number k of edited edges (see [14] for a complete survey). Given our focus on increasing the practicality of treewdith-based algorithms in bioinformatics, we restrict our focus to treewidth related parameters tw, tw' and $d = tw - tw'$.

Considering the target treewidth tw', we note that EDP is NP-hard when $\mathcal{H}$ is the family of treewidth-2 graphs [18], namely K_4-free graphs, hence the notation EDP(K_4). It follows that Graph-Diet is Para-NP-hard for the target treewidth parameter tw'.

3.1 Graph-Diet: practical solutions seem unlikely

For a combination of the parameters tw' and k, we could imagine graph minor theorems yielding parameterized algorithms "for free", as it is often the case with treewidth-based problems. In this respect, Graph-Diet corresponds to deciding if a graph G belongs to the family of graphs having treewidth tw', augmented by k additional edges, denoted as Treewidth-tw'+ke since its introduction by Cai [12]. If this family were minor-closed, polynomial minor-free-testing [27, 34] would yield an FPT algorithm. However, this is not the case: for some graphs in the family, an edge contraction yields a graph G' not in Treewidth-tw'+ke, as illustrated by Figure 3.

Regarding the source graph treewidth tw, a theoretical approach could be via Courcelle's Theorem [13] and Monadic Second Order (MSO) formulas: it suffices to express the property of being in Treewidth-tw'+ke using MSO to prove that Graph-Diet is FPT for tw. We presume this is feasible (the main brick being minor testing, which is expressible in MSO), however it is not clear whether this is doable with formulas independent of k, in order to obtain an algorithm for the treewidth alone. In any case, this approach would probably not yield practical algorithms. Indeed, Courcelle's theorem typically lead to running times involving towers of exponentials on the relevant parameters, so we do not investigate it further within the scope of this paper.

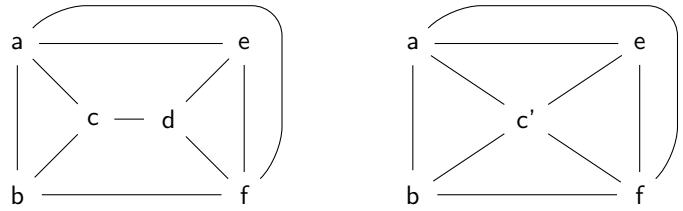


Figure 3 A graph G (left) with treewidth 3. Deleting edge cd gives treewidth 2, implying that $G \in$ Treewidth2 + 1e. However, if one contracts edge cd, then the resulting graph (right) has treewidth 3, and deleting any single edge does not decrease the treewidth. This example shows that the graph family Treewidth 2+1e is not minor-closed.

Another meta-theorem by Cai [11] may yield an FPT algorithm if Treewidth-tw'+ke can be described through a finite number forbidden induced subgraphs, but again k would most likely be a parameter as well. On a related note, it is worth noting that Edge Deletion to other graph classes (interval, permutation, ...) does admit efficient algorithms when parameterized by the treewidth alone [35], painting a contrasted picture.

The vertex deletion equivalent of Graph-Diet, where one asks for a minimum subset of vertices to remove to obtain a given treewidth, is known as a Treewidth Modulator. This problem has been better-studied than its edge-deletion counterpart [16], and has been shown to be FPT for the treewidth [3], with a reasonable dependency in the parameter. However, it is currently unclear how this can be adapted into an edge deletion algorithm.

Overall an FPT algorithm for Graph-Diet does not seem out of reach, and could result from one of the above-mentioned meta-theorems. However it seems unlikely to induce a "practical" exact algorithms. Indeed, any algorithm for Graph-Diet can be used to compute the Treewidth of an arbitrary graph, for which current state-of-the-art exact algorithms require time in $tw^{O(tw^3)}$ [6]. We thus have the following conjecture, which motivates the Tree-Diet relaxation of the problem.

▶ **Conjecture 1.** Graph-Diet *is* FPT *for the source treewidth parameter (tw), but no algorithm with single-exponential running time exists.*

Finally, for parameter d, any polynomial-time algorithm for constant d would allow to compute the treewidth of any graph in polynomial time. Since treewidth is NP-hard we have the following result

▶ **Theorem 2.** *There is no* XP *algorithm for* Graph-Diet *with parameter d unless* P = NP.

3.2 Lower Bounds for Tree-Diet

Parameters tw' and d would be the most interesting in practice, since parameterized algorithms would be efficient for small diets *or* small target treewidth. However, we prove strong lower-bounds for Tree-Diet on each of these parameters, leaving very little hope for parameterized algorithms (we thus narrow down the possible algorithms to the combined parameter $tw' + d$, i.e. tw, see Section 4). Only XP for parameter d when $\mathcal{T}$ has a constant degree remains open (cf. Table 1).

▶ **Theorem 3.** Tree-Diet *and* Path-Diet *are* Para-NP*-hard for the target treewidth parameter tw' (*NP*-hard for $tw' = 1$).*

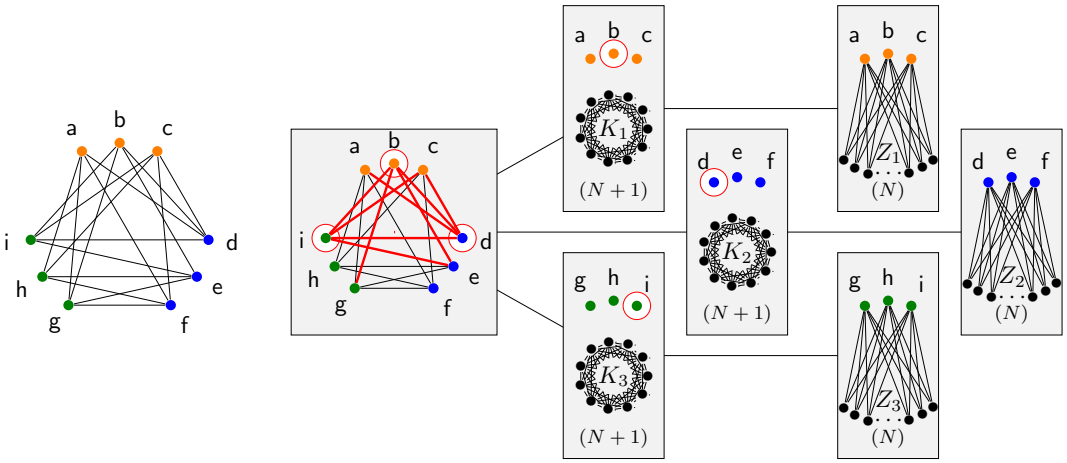


Figure 4 Reduction for Theorem 4 showing that Tree-Diet is NP-hard even for $d = 1$, from a graph G (left) with $k = 3$ and $n = 3$ to a graph G' (right, given by its tree-decomposition of width $N + n + 1$): a 1-tree-diet for G' amounts to selecting a k-clique in the root bag, i.e. in G.

Proof. By reduction from the NP-hard problem Spanning Caterpillar Tree [42]: given a graph G, does G has a spanning tree C that is a caterpillar? Given $G = (V, E)$ with $n = |V|$, we build a tree-decomposition $\mathcal{T}$ of G consisting of $n - 1$ bags containing all vertices (the width of the decomposition is therefore $n - 1$) connected in a path. Then $(G, \mathcal{T})$ admits a tree-diet to treewidth 1 with $n - 1$ visible edges if, and only if, G admits a caterpillar spanning tree. Indeed, the subgraph of G with visible edges must be a graph with pathwidth 1, i.e. a caterpillar [30]. With $n - 1$ visible edges, the caterpillar connects all n vertices together, i.e. it is a spanning tree. ◀

▶ **Theorem 4.** Tree-Diet *is* Para-NP*-hard for parameter* d*. More precisely, it is* W[1]*-hard for parameter* Δ*, the degree of* $\mathcal{T}$*, even when* $d = 1$*.*

Proof. By reduction from Multi-Colored Clique. Consider a k-partite graph $G = (V, E)$ with $V = \bigcup_{i=1}^{k} V_i$. We assume that G is regular (each vertex has degree δ and that each V_i has the same size n (Multi Colored Clique is W[1]-hard under these restrictions [17, 15]). Let $L := \delta k - \binom{k}{2}$ and $N = \max\{|V|, L + 1\}$. We now build a graph G' and a tree-decomposition $\mathcal{T}'$: start with $G' := G$. Add k independent cliques $K_1, \ldots, K_k$ of size $N + 1$. Add k sets of N vertices Z_i ($i \in [k]$) and, for each $i \in [k]$, add edges between each $v \in V_i$ and each $z \in Z_i$. Build $\mathcal{T}$ using $2k + 1$ bags $T_0, T_{1,i}, T_{2,i}$ for $i \in [k]$, such that $T_0 = V$, $T_{1,i} = V_i \cup K_i$ and $T_{2,i} = V_i \cup Z_i$. The tree-decomposition is completed by connecting $T_{2,i}$ to $T_{1,i}$ and $T_{1,i}$ to T_0 for each $i \in [k]$. Thus, $\mathcal{T}$ is a tree-decomposition of G' with $\Delta = k$ and maximum bag size $n + N + 1$ (vertices of V induce a size-3 path in $\mathcal{T}$, other vertices appear in a single bag, edges of G appear in T_0, edges of K_i in $T_{1,i}$, and finally edges between V_i and Z_i appear in $T_{2,i}$). The following claim completes the reduction:

$\mathcal{T}$ has a 1-tree-diet losing at most L edges from $G' \Leftrightarrow G$ has a k-clique.

$\boxed{\Leftarrow}$ Assume G has a k-clique $X = \{x_1, \ldots, x_k\}$ (with $x_i \in V_i$). Build $\mathcal{T}'$ by removing each x_i from bags T_0 and $T_{1,i}$. Then $\mathcal{T}'$ is a 1-tree-diet of $\mathcal{T}$. There are no edges lost by removing x_i from $T_{1,i}$ (since x_i is not connected to K_i), and the edges lost in T_0 are all edges of G adjacent to any x_i. Since X forms a clique and each x_i has degree δ, there are $L = k\delta - \binom{k}{2}$ such edges.

$\boxed{\Rightarrow}$ Consider a 1-tree-diet $\mathcal{T}'$ of $\mathcal{T}$ losing L edges. Since each bag $T_{1,i}$ has maximum size, $\mathcal{T}'$ must remove at least one vertex x_i in each $T_{1,i}$. Note that $x_i \in V_i$ (since removing $x_i \in K_i$ would loose at least $N \geq L+1$ edges). Furthermore, x_i may not be removed from $T_{2,i}$ (otherwise N edges between x_i and Z_i would be lost), so x_i must also be removed from T_0. Let K be the number of edges in $G[\{x_1 \dots x_k\}]$. The total number of lost edges in T_0 is $\delta k - K$. Thus, we have $\delta k - K \leq L$ and $K \geq \binom{k}{2}$: $\{x_1, \dots, x_k\}$ form a k-clique of G. ◀

▶ **Theorem 5.** Path-Diet *is* W[1]*-hard for parameter* d.

4 FPT Algorithm

4.1 For general tree-decompositions

We describe here a $O(3^{tw}n)$-space, $O(\Delta^{tw+2} \cdot 6^{tw}n)$-time dynamic programming algorithm for the Tree-Diet problem, with Δ and tw being respectively the maximum number of children of a bag in the input tree-decomposition and its width. On *binary* tree-decompositions (where each bag has at most 2 children), it yields a $O(3^{tw}n)$-space $O(12^{tw}n)$-time FPT algorithm.

4.1.1 Coloring formulation

We aim at solving the following problem: given a tree-decomposition $\mathcal{T}$ of width tw of a graph G, we want to remove vertices from the bags of $\mathcal{T}$ to reach a target width tw' while *losing* as few edges from G as possible. We tackle the problem through an equivalent *coloring* formulation: our algorithm will assign a color to each occurrence of a vertex in the tree decomposition. We work with three colors: red (`r`), orange (`o`), and green (`g`). Green means that the vertex is kept in the bag, while orange and red means removal of the vertex. An edge is thus visible within a bag when both its ends are green. It is lost if there is no bag where it is visible. To ensure equivalence with the original problem, the colors will be assigned following local rules, which we now describe.

We first root the tree-decomposition arbitrarily.

▶ **Definition 6.** *A coloring of vertices in the bags of the decomposition is said to be* valid *if it follows the following rules:*

- *A vertex of a bag not present in its parent may be green or orange (R1)*
- *A green vertex in a bag may be either green or red in its children (R2)*
- *A red vertex in a bag must stay red in its children (R3)*
- *An orange vertex in a bag has to be either orange or green in exactly one child (unless there is no child with this vertex), and must be red in the other children (R4)*

These rules are summarized in Figure 5 (a).

When going down the tree, a green vertex may only stay green or permanently become red. An immediate consequence of these rules is therefore that the green occurences of a given vertex form a (possibly empty) connected subtree. Informally, orange vertices are locally absent but "may potentially be found further down the tree", while red vertices are removed from both the current bag and its entire subtree. Figure 5 (b) shows an example sketch for a valid coloring of the occurrences of a given vertex in the tree-decomposition. A vertex may only be orange along a path starting form its highest occurrence in the tree, with any part branching off that path entirely red. It ends at the top of a (potentially empty) green subtree, whose vertices may also be parents to entirely red subtrees.

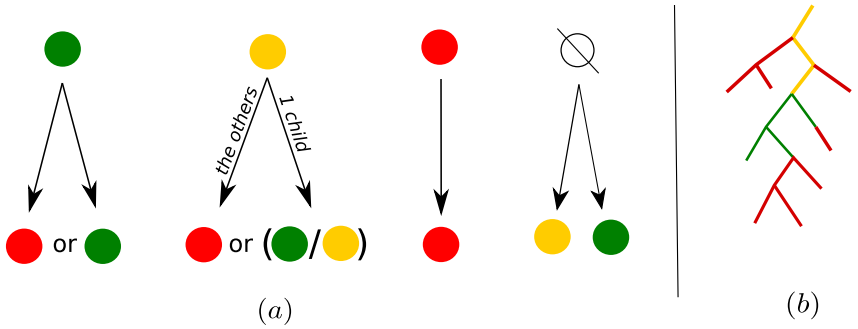


Figure 5 (*a*) Color assignation rules for vertices, when going down-tree. (*b*) Sketch of the general pattern our color assignation rules create on $\mathcal{T}_u$, the subtree of bags containing a given vertex u.

We will now more formally prove the equivalence of the coloring formulation to the original problem. Let us first introduce two definitions. Given a valid coloring $\mathcal{C}$ of a tree-decomposition of G, an edge (u, v) of G is said to be *realizable* if there exists a bag in which both u and v are green per $\mathcal{C}$. Given an integer d, a coloring $\mathcal{C}$ of $\mathcal{T}$ is said to be *d-diet-valid* if removing red/orange vertices reduces the width of $\mathcal{T}$ from $w(T)$ to $w(T) - d$.

▶ **Proposition 7.** *Given a graph G, a tree-decomposition $\mathcal{T}$ of width tw, and a target width $tw' < tw$, The* Tree-Diet *problem is equivalent to finding a $(tw - tw')$-diet-valid coloring $\mathcal{C}$ of $\mathcal{T}$ allowing for a number of realizable edges in G as large as possible.*

Proof. Given a $(tw - tw')$-tree-diet of $\mathcal{T}$, specifying which vertices are removed from which bags, we obtain a valid coloring $\mathcal{C}$ for $\mathcal{T}$ incurring the same number of lost (unrealizable) edges. To start with, a vertex u is colored green in the bags where it is not removed. By the validity of $\mathcal{T}'$ as a decomposition, this set of bags forms a connected subtree, that we denote $\mathcal{T}_u^{\mathrm{g}}$. We also write $\mathcal{T}_u$ for the subtree of bags containing u in the original decomposition $\mathcal{T}$. If $\mathcal{T}_u^{\mathrm{g}}$ and $\mathcal{T}_u$ do not have the same root, then u is colored orange on the the path in $\mathcal{T}$ from the root of $\mathcal{T}_u$ (included) and the root of $\mathcal{T}_u^{\mathrm{g}}$ (excluded). Vertex u is colored red in any other bag of $\mathcal{T}_u$ not covered by these two cases. The resulting coloring follows rules (R1-4) and induces the same set of lost/non-realizable edges as the original $(tw - tw')$-tree-diet. Conversely, an equivalent $(tw - tw')$-tree-diet is obtained from a $(tw - tw')$-diet-valid coloring by removing red/orange vertices and keeping green ones. If a given vertex has no green occurences, it is entirely removed from the tree decomposition and all its edges are *lost* (it becomes an isolated vertex). We may add it back to the tree decomposition by introducing a new bag containing only this vertex, which we connect arbitrarily to the tree decomposition. ◀

4.1.2 Decomposition of the search space and sub-problems

Based on this coloring formulation, we now describe a dynamic programming scheme for the Tree-Diet problem. We work with sub-problems indexed by tuples (X_i, f), with X_i a bag of the input tree decomposition and f a coloring of the vertices of X_i in green, orange or red (in particular, $f^{-1}(\mathtt{g})$ denotes the green vertices of X_i, and similarly for `o` and `r`).

Let us introduce some notations before giving the definition of our dynamic programming table. Given an edge (u, v) of G, realizable when coloring a tree-decomposition $\mathcal{T}$ of G with $\mathcal{C}$, we write $\mathcal{T}_{uv}^{\mathrm{g}}$ the subtree of $\mathcal{T}$ in which both u and v are green. We denote by $\mathcal{T}_i$ the subtree of the decomposition rooted at X_i, and $C(i, f)$ the d-diet-valid colorings of $\mathcal{T}_i$ agreeing with f on i, with $d = tw - tw'$. Our dynamic programming table is then defined as:

$$c(X_i, f) = \begin{cases} \max\limits_{\mathcal{C} \in C(i,f)} \left| \left\{ \begin{array}{l} \text{Edges } (u,v) \text{ of } G, \text{ realizable within } \mathcal{T}_i \text{ colored with } \mathcal{C} \\ \text{such that } \mathcal{T}^{\mathrm{g}}_{uv} \text{ is entirely contained strictly below } X_i \end{array} \right\} \right| \\ \qquad\qquad \text{if } f \text{ assigns green to at most } tw' + 1 \text{ vertices} \\ -\infty \qquad\qquad \text{otherwise} \end{cases}$$

The cell $c(X_i, f)$ therefore aggregates all edges realizable *strictly below* X_i. As we shall see through the recurrence relation below and its proof, edges with both ends green in X_i will be accounted for *above* X_i in $\mathcal{T}$.

We assume w.l.o.g that the tree-decomposition is rooted at an empty bag R. Given the definition of the table, the maximum number of realizable edges, compatible with a tree-diet of $(tw - tw')$ to $\mathcal{T}$, can be found in $c(R, \emptyset)$.

The following theorem presents a recurrence relation obeyed by $c(X_i, f)$:

▶ **Theorem 8.** *For a bag X_i of $\mathcal{T}$, with children $Y_1, ...Y_\Delta$, we have:*

$$c(X_i, f) = \max_{m : f^{-1}(o) \to [1..\Delta]} \left[\sum_{1 \leq j \leq \Delta} \left(\max_{f'_j \in compatible(Y_j, f, m)} c(Y_j, f'_j) + \left| count(f, f'_j) \right| \right) \right]$$

with

- *m: a map from the orange vertices in X_i to the children of X_i. It decides for each orange vertex u, which child, among those which contain u, will color u orange or green; If there are no orange vertices in X_i, only the trivial empty map is considered.*
- *$compatible(Y_j, f, m)$: the set of colorings of Y_j compatible with f on X_i and m;*
- *$count(f, f'_j)$: set of edges of G involving two vertices of Y_j green by f'_j, but such that one of them is either not in X_i or not green by f.*

Note that $compatible(Y_j, f, m)$ may contain colorings f'_j that colour too many vertices in Y_j in green to reach target width tw'. In that case $c(Y_j, f'_j) = -\infty$.

Theorem 8 relies on the following separation lemma for realizable edges under a valid coloring of a tree-decomposition. Recall that we suppose w.l.o.g that the tree-decomposition is rooted at an empty bag.

▶ **Lemma 9.** *An edge (u, v) of G, realizable in $\mathcal{T}$ under $\mathcal{C}$, is contained in exactly one set of the form $count(C_{|P}, C_{|X})$ with X a bag of $\mathcal{T}$ and P its parent, $C_{|P}, C_{|X}$ the restrictions of $\mathcal{C}$ to P and X, respectively, and "count" defined as above. In addition, X is the root of the subtree of $\mathcal{T}$ in which both u and v are green.*

Dynamic programming algorithm

The recurrence relation of Theorem 8 naturally yields a dynamic programming algorithm for the Tree-Diet problem, as stated below:

▶ **Theorem 10.** *There exists a $O(\Delta^{tw+2} \cdot 6^{tw} \cdot n)$-time, $O(3^{tw} \cdot n)$-space algorithm for the* Tree-Diet *problem, with Δ the maximum number of children of a bag in the input tree-decomposition, and tw its width.*

▶ **Corollary 11.** Binary-Tree-Diet *($\Delta = 2$) admits an* FPT *algorithm for the tw parameter.*

A pseudo-code implementation of the algorithm, using memoization, is included in Appendix B.

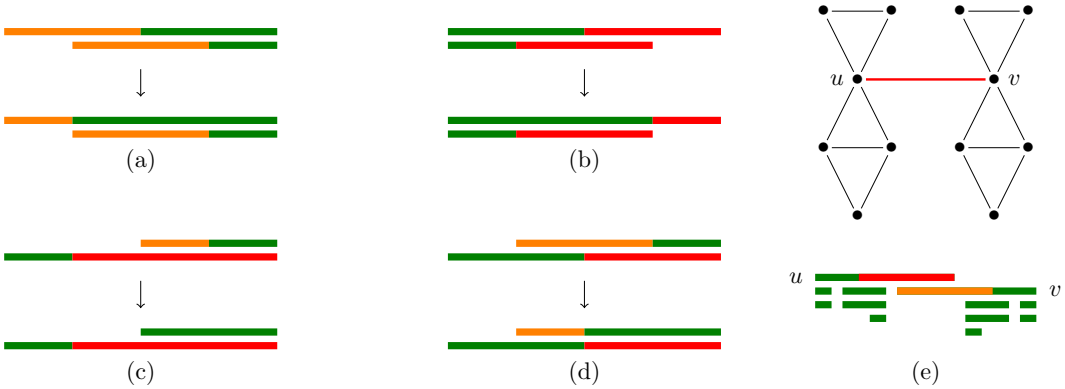


Figure 6 Five cases where two vertices are deleted in the same bag with $d = 1$. Bags are points in the line, and an interval covering all bags containing v is drawn for each v (with an equivalent coloring, see Proposition 7). Cases (a) to (d) can be safely avoided by applying the given transformations. In the example for case (e), however, it is necessary to delete both vertices u and v form a central bag. It is sufficient to avoid cases (a) and (b) in order to obtain an XP algorithm for d.

4.2 For path decompositions

In the context of paths decompositions, we note that the number of removed vertices per bag can be limited to at most $2d$ without losing the optimality. More precisely, we say that a coloring $\mathcal{C}$ is *d-simple* if any bag has at most d orange and d red vertices. We obtain the following result, using transformations given in Figure 6.

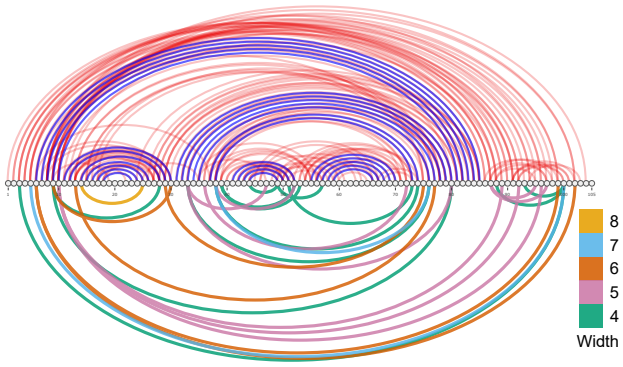
▶ **Proposition 12.** *Given a graph G and a path-decomposition $\mathcal{T}$, if $\mathcal{C}$ is a d-diet-valid coloring of $\mathcal{T}$ losing k edges, then $\mathcal{T}$ has a d-diet-valid coloring that is d-simple, and loses at most k edges.*

Together with Proposition 7, this shows that it is sufficient to restrict our algorithm to d-simple colorings. (See also Figure 6). In particular, for any set X_i, choosing which $\leq d$ vertices are orange and which $\leq d$ are red, among the total of n vertices, is enough to fix a coloring. The number of such colorings is therefore bounded by $O(tw^{2d})$. Applying this remark to our algorithm presented in Section 4.1 yields the following result:

▶ **Theorem 13.** Path-Diet *can be solved in $O(tw^{2d}n)$-space and $O(tw^{4d}n)$-time.*

5 Proofs of concept

We now illustrate the relevance of our approach, and the practicality of our algorithm for Tree-Diet, by using it in conjunction with FPT algorithms for three problems in RNA bioinformatics. We implemented in C++ the dynamic programming scheme described in Theorem 10 and Appendix B. Its main primitives are made available for Python scripting through pybind11 [22]. It actually allows to solve a generalized *weighted* version of Tree Diet, as explained in Appendix B. This feature allows to favour the conservation of important edges (e.g. RNA backbone) during simplification, by assigning them a much larger weight compared to other edges. Our implementation is freely available at `https://gitlab.inria.fr/amibio/tree-diet`.



tw'	#Preserved BPs EteRNA22	EteRNA77
9	–	**183**
8	–	182
7	–	180
6	**465**	176
5	460	168
4	456	157
3	445	144
2	418	121
1	320	86

Figure 7 (Left) Target secondary structure (blue BPs), full set of disruptive base pairs (DPB; top) inferred by RNAPond on the Eterna77 puzzle, and subsets of DBPs (bottom) cumulatively removed by the tree-diet algorithm to reach prescribed treewidths. (Right) Number of BPs retained by our algorithm, targeting various treewidth values for the EteRNA22 and EteRNA77 puzzles.

5.1 Memory-parsimonious unbiased sampling of RNA designs

As a first use case for our simplification algorithm, we strive to ease the sampling phase of a recent method, called RNAPond [51], addressing RNA negative design. The method targets a set of base pairs S, representing a secondary structure of length n, and infers a set $\mathcal{D}$ of m disruptive base pairs (DBPs) that must be avoided. It relies on a $\Theta(k \cdot (n+m))$ time algorithm for sampling k random sequences (see Appendix C for details) after a preprocessing in $\Theta(n \cdot m \cdot 4^{tw})$ time and $\Theta(n \cdot 4^{tw})$ space. Here, the input consists of a graph $G = ([1,n], S \cup \mathcal{D})$ and a tree decomposition $\mathcal{T}$ of G, having width tw. In practice, the preprocessing largely dominates the overall runtime, even for large values of k, and its large memory consumption represents the main bottleneck.

This discrepancy in the complexities/runtimes of the preprocessing and sampling suggests an alternative strategy: relaxing the set of constraints to $(S', \mathcal{D}')$, with $(S' \cup \mathcal{D}') \subset (S \cup \mathcal{D})$, and compensating it through a rejection of sequences violating constraints in $(S, \mathcal{D}) \setminus (S', \mathcal{D}')$. The relaxed algorithm would remain unbiased, while the average-case time complexity of the rejection algorithm would be in $\Theta(k \cdot \overline{q} \cdot (n+m))$ time, where $\overline{q}$ represents the relative increase of the partition function ($\approx$ the sequence space) induced by the relaxation. The preprocessing step would retain the same complexity, but based on a (reduced) treewidth $tw' \leq tw$ for the relaxed graph $G' = ([1,n], S' \cup \mathcal{D}')$.

These complexities enable a tradeoff between the rejection (time), and the preprocessing (space), which may be critical to unlock future applications of RNA design. Indeed, the treewidth can be decreased by removing relatively few base pairs, as demonstrated below using our algorithm on pairs inferred for hard design instances.

We considered sets of DBPs inferred by RNAPond over two puzzles in the EteRNA benchmark. The EteRNA22 puzzle is an empty secondary structure spanning 400 nts, for which RNAPond obtains a valid design after inferring 465 DBPs. A tree decomposition of the graph formed by these 465 DPBs is then obtained with the standard min-fill-ordering heuritic [8], giving a width of 6. The EteRNA77 puzzle is 105 nts long, and consists in a collection of helices interspersed with destabilizing internal loops. RNApond failed to produce a solution,and its final set of DBPs consists of 183 pairs, for which the same heuristic yields a tree decomposition of width 9. We further make both tree decompositions binary through bag duplications (see Appendix A), giving an FPT runtime to our algorithm, while potentially lowering the number of lost edges.

Executing the tree-diet algorithm (Theorem 10) on both graphs and their tree decompositions, we obtained simplified graphs, having lower treewidth while typically losing few edges, as illustrated and reported in Figure 7. Remarkably, the treewidth of the DBPs inferred for EteRNA22 can be decreased to $tw' = 5$ by only removing 5 DBPs/edges (460/465 retained), and to $tw' = 4$ by removing 4 further DBPs (456/465). For EteRNA77, our algorithm reduces the treewidth from 9 to 6 by only removing 7 DBPs.

Rough estimates can be provided for the tradeoff between the rejection and preprocessing complexities, by assuming that removing a DBP homogeneously increases the value of $\mathcal{Z}$ by a factor $\alpha := 16/10$ (#pairs/#incomp. pairs). The relative increase in partition function is then $\overline{q} \approx \alpha^b$, when b base pairs are removed. For EteRNA22, reducing the treewidth by 2 units (6→4), *i.e.* a 16 fold reduction of the memory and preprocessing time, can be achieved by removing 9 DBPs, *i.e.* a 69 fold expected increase in the time of the generation phase. For EteRNA77, the same 16 fold ($tw' = 9 \to 7$) reduction of the preprocessing time/space can be achieved through an estimated 4 fold increase of the generation time. A more aggressive 256 fold memory gain can be achieved at the expense of an estimated 1 152 fold increase in generation time. Given the large typical asymmetry in runtimes and implementation constants between the computation-heavy preprocessing and, relatively light, generation phases, the availability of an algorithm for the TREE-DIET problem provides new options, especially to circumvent memory limitations.

5.2 Structural alignment of complex RNAs

Structural homology is often posited within functional families of non-coding RNAs, and is foundational to algorithmic methods for multiple RNA alignments [23], considering RNA base pairs while aligning distant homologs. In the presence of complex structural features (pseudoknots, base triplets), the sequence-structure alignment problem becomes hard, yet admits XP solutions based on the treewidth of the base pair + backbone graph. In particular, Rinaudo *et al.* [32] describe a $\Theta(n.m^{tw+1})$ algorithm for optimally aligning a structured RNA of length n onto a genomic region of length m. It optimizes an alignment score that includes: i) substitution costs for matches/mismatches of individual nucleotides and base pairs (including arc-breaking) based on the RIBOSUM matrices [24]; and ii) an affine gap cost model [33]. We used the implementation of the Rinaudo *et al.* algorithm, implemented in the LicoRNA software package [45, 46].

5.2.1 Impact of treewidth on the structural alignment of a riboswitch

In this case study, we used our tree-diet algorithm to modulate the treewidth of complex RNA structures, and investigate the effect of the simplification on the quality and runtimes of structure-sequence alignments. We considered the Cyclic di-GMP-II riboswitch, a regulatory motif found in bacteria that is involved in signal transduction, and undergoes conformational change upon binding the second messenger c-di-GMP-II [40, 41]. A 2.5Å resolution 3D model of the c-di-GMP-II riboswitch in *C. acetobutylicum*, proposed by Smith *et al.* [38] based on X-ray crystallography, was retrieved from the PDB [4] (PDBID: 3Q3Z). We annotated its base pairs geometrically using the DSSR method [28]. The canonical base pairs, supplemented with the backbone connections, were then accumulated in a graph, for which we heuristically computed an initial tree decomposition $\mathcal{T}_4$, having treewidth $tw = 4$.

We simplified our the initial tree decomposition $\mathcal{T}_4$, and obtained simplified models $\mathcal{T}_3$, and $\mathcal{T}_2$, having width $tw' = 3$ and 2 respectively. As controls, we included tree decompositions based on the secondary structure (max. non-crossing set of BPs; $\mathcal{T}_{2D}$) and sequence ($\mathcal{T}_{1D}$).

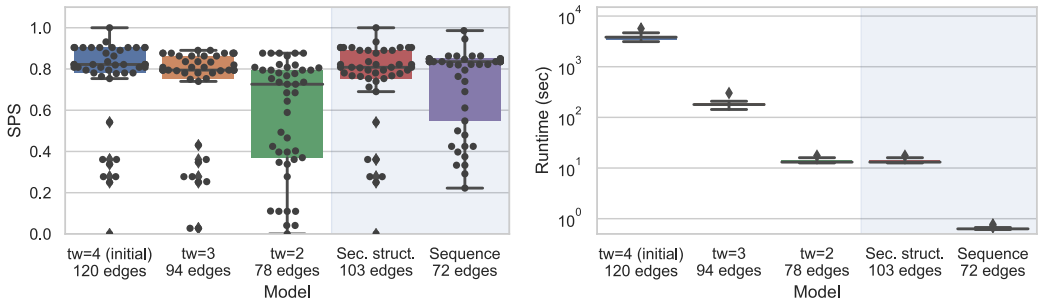


Figure 8 Impact on alignment quality (SPS; Left) and runtime (Right) of simplified instances for the RNA sequence-structure alignment of the pseudoknotted c-di-GMP-II riboswitch. The impact of simplifications on the quality of predicted alignments, using RFAM RF01786 as a reference, appears limited while the runtime improvement is substantial.

We used LicoRNA to predict an alignment $a_{\mathcal{T},w}$ of each original/simplified tree decomposition $\mathcal{T}$ onto each sequence w of the c-di-GMP-II riboswitch family in the RFAM database [23] (RF01786). Finally, we reported the LicoRNA runtime, and computed the Sum of Pairs Score (SPS) [43] as a measure of the accuracy of $a_{\mathcal{T},w}$ against a reference alignment $a^\star_w$:

$$\mathsf{SPS}(a_{\mathcal{T},w}; a^\star_w) = \frac{|\,\mathsf{MatchedCols}(a_{\mathcal{T},w}) \cap \mathsf{MatchedCols}(a^\star_w)\,|}{|\,\mathsf{MatchedCols}(a^\star_w)\,|},$$

using as reference the alignment $a^\star_w$ between the 3Q3Z sequence and w induced by the manually-curated RFAM alignment of the RF01786 family.

The results, presented in Figure 8, show a limited impact of the simplification on the quality of the predicted alignment, as measured by the SPS in comparison with the RFAM alignment. The best average SPS (77.3%) is achieved by the initial model, having treewidth of 4, but the average difference with simplified models appears very limited (e.g. 76.5% for $\mathcal{T}_3$), especially when considering the median. Meanwhile, the runtimes mainly depend on the treewidth, ranging from 1h for $\mathcal{T}_4$ to 300ms for $\mathcal{T}_{1D}$. Overall, $\mathcal{T}_{2D}$ seems to represent the best compromise between runtime and SPS, although its SPS may be artificially inflated by our election of RF01786 as our reference (built from a covariance model, *i.e.* essentially a 2D structure). Finally, the difference in number of edges (and induced SPS) between $\mathcal{T}_{2D}$ and $\mathcal{T}_2$, both having $tw = 2$, exemplifies the difference between the Tree-Diet and Graph-Diet problems, and motivates further work on the latter.

5.2.2 Exact iterative strategy for the genomic search of ncRNAs

In this final case study, we consider an exact filtering strategy to search new occurrences of a structured RNA within a given genomic context. In this setting, one attempts to find all ε-admissible (cost $\leq \varepsilon$) occurrences/hits of a structured RNA S of length n within a given genome of length $g \gg n$, broken down in windows of length $\kappa.n$, $\kappa > 1$. Classically, one would align S against individual windows, and report those associated with an admissible alignment cost. This strategy would have an overall $\Theta(g \cdot n^{tw+2})$ time complexity, applying for instance the algorithm of [32].

Our instance simplification framework enables an alternative strategy, that incrementally filters out unsuitable windows based on models of increasing granularity. Indeed, for any given target sequence, the min alignment cost c_δ obtained for a simplified instance of treewidth $tw - \delta$ can be corrected (*cf* Appendix D) into a lower bound $c^\star_\delta$ for the min alignment

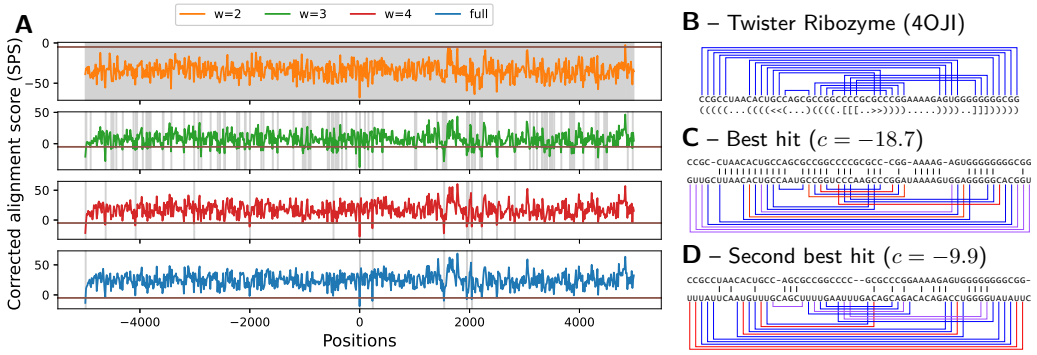


Figure 9 Corrected costs associated with the search for structured homologs of the Twister ribozyme in chromosome 5 of *S. bicolor*, using simplified instances of various treewidth (A). Gray areas represent scores which, upon correction, remain below the cutoff, and have to be considered for further steps of the iterated filtering. Canonical base pairs of the ribozyme (PDBID 4OJI; B), mapped onto to the best hit (C) and second best hit (D) found along the search colored depending on their support in the target sequence (Red: incompatible; Purple: unstable G-U; Blue: stable).

cost $c_0^\star$ of the full-treewidth instance tw. Any window such that $c_\delta^\star > \varepsilon$ thus also obeys $c_0^\star > \varepsilon$, and can be safely discarded from the list of putative ε-admissible windows, without having to perform a full-treewidth alignment. Given the exponential growth of the alignment runtime for increasing treewidth values (see Figure 8-right) this strategy is expected to yield substantial runtime savings.

We used this strategy to search occurrences of the Twister ribozyme (PDBID 4OJI), a highly-structured ($tw = 5$) 54nts RNA initially found in *O. sativa* (asian rice) [26]. We targeted the *S. bicolor* genome (sorghum), focusing on a 10kb region centered on the 2,485,140 position of the 5th chromosome, where an instance of the ribozyme was suspected within an uncharacterized transcript (LOC110435504). The 4OJI sequence and structure were extracted from the 3D model as above, and included into a tree decomposition $\mathcal{T}_5$ (73 edges), simplified into $\mathcal{T}_4$ (71 edges), $\mathcal{T}_3$ (68 edges) and $\mathcal{T}_2$ (61 edges) using the tree-diet algorithm.

We aligned all tree decompositions against all windows of size 58nts using 13nts offset, and measured the score and runtime of the iterative filtering strategy using a cost cutoff $\varepsilon = -5$. The search recovers the suspected occurrence of twister as its best result (Figure 9.C), but produced hits (*cf* Figure 9.D) with comparable sequence conservation that could be the object of further studies. Regarding the filtering strategy, while $\mathcal{T}_2$ only allows to rule out 3 windows out of 769, $\mathcal{T}_3$ allows to eliminate an important proportion of putative targets, retaining only 109 windows, further reduced to 15 windows by $\mathcal{T}_4$, 6 of which end up as final hits for the full model $\mathcal{T}_5$ (*cf* Figure 9.A). The search remains exact, but greatly reduces the overall runtime from 24 hours to 34 minutes (42 fold!).

6 Conclusion and discussion

We have established the parameterized complexity of three treewidth reduction problems, motivated by applications in Bioinformatics, as well as proposed practical algorithms for instances of reasonable treewidths. The reduced widths obtained by our proposed algorithm can be used to obtain: i) sensitive heuristics, owing to the consideration of a maximal amount of edges/information in the thinned graphs; ii) *a posteriori* approximation ratios, by comparing the potential contribution of removed edges to the optimal score obtained of the

thinned instance by a downstream FPT/XP algorithm; iii) substantial practical speedups without loss of correctness, e.g. when partial filtering can be safely achieved based on simplified input graphs

Open questions. Regarding the parameterized complexity of Graph-Diet and Tree-Diet, some questions remain open (see Table 1): an FPT algorithm for Tree-Diet (ideally, with $2^{O(tw)} \cdot n$ running time), would be the most desirable, if possible satisfying the backbone constraints. We also aim at settling the parameterized complexity of the Graph-Diet problem, and try to give efficient exact algorithms for this problem (possibly using some tree-decomposition in input). Finally, we did not include the number of deleted edges in our multivariate analysis: even though in practice it is more difficult, a priori, to guarantee it has a small value, we expect it can be used to improve the running time in many cases.

Backbone Preservation. In two of our applications, the RNA secondary structure graph contains two types of edges: those representing the *backbone* of the sequence (i.e., between consecutive bases) and those representing base pair bounds. In practice, we want all backbone edges to be visible in the resulting tree-decomposition, and only base pairs may be lost. This can be integrated to the Tree-Diet model (and to our algorithms) using weighted edges, using the total weight rather than the count of deleted edges for the objective function. Note that some instances might be unrealizable (with no tree-diet preserving the backbone, especially for low tw'). In most cases, ad-hoc bag duplications can help avoid this issue.

From a theoretical perspective, weighted edges may only increase the algorithmic complexity of the problems. However, a more precise model could consider graphs which already include a hamiltonian path (the backbone), and the remaining edges form a degree-one or two subgraph. Such extra properties may, in some cases, actually reduce the complexity of the problem. As an extreme case, we conjecture the Path-Diet problem for $tw' = 1$ becomes polynomial in this setting.

References

1 Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Appl. Math.*, 104(1-3):45–62, 2000. doi:10.1016/S0166-218X(00)00186-4.

2 Julien Baste, Christophe Paul, Ignasi Sau, and Celine Scornavacca. Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees. *Bulletin of Mathematical Biology*, 79(4):920–938, February 2017. doi:10.1007/s11538-017-0260-y.

3 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. i. general upper bounds. *SIAM J. Discret. Math.*, 34(3):1623–1648, 2020. doi:10.1137/19M1287146.

4 H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic acids research*, 28:235–242, 2000. doi:10.1093/nar/28.1.235.

5 Guillaume Blin, Alain Denise, Serge Dulucq, Claire Herrbach, and Helene Touzet. Alignments of RNA structures. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):309–322, April 2010. doi:10.1109/tcbb.2008.28.

6 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

7 Hans L Bodlaender and Arie MCA Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

8 Hans L Bodlaender and Arie MCA Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

9 Laurent Bulteau, Guillaume Fertin, Minghui Jiang, and Irena Rusu. Tractability and approximability of maximal strip recovery. *Theoretical Computer Science*, 440:14–28, 2012.

10 Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: An overview. *Algorithms*, 12(12):256, December 2019. doi:10.3390/a12120256.

11 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

12 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

13 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

14 Christophe Crespelle, Pål Grønås Drange, Fedor V Fomin, and Petr A Golovach. A survey of parameterized algorithms and the complexity of edge modification. *arXiv preprint*, 2020. arXiv:2001.06867.

15 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

16 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. On the hardness of losing width. In *International Symposium on Parameterized and Exact Computation*, pages 159–168. Springer, 2011.

17 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

18 Ehab S El-Mallah and Charles J Colbourn. The complexity of some edge deletion problems. *IEEE transactions on circuits and systems*, 35(3):354–362, 1988.

19 Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. *arXiv preprint*, 2012. arXiv:1207.4109.

20 Stefan Hammer, Wei Wang, Sebastian Will, and Yann Ponty. Fixed-parameter tractable sampling for RNA design with multiple target structures. *BMC Bioinformatics*, 20(1), April 2019. doi:10.1186/s12859-019-2784-7.

21 Buhm Han, Banu Dost, Vineet Bafna, and Shaojie Zhang. Structural alignment of pseudoknotted RNA. *Journal of Computational Biology*, 15(5):489–504, 2008. doi:10.1089/cmb.2007.0214.

22 Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between c++11 and python, 2017. URL: https://github.com/pybind/pybind11.

23 Ioanna Kalvari, Eric P Nawrocki, Nancy Ontiveros-Palacios, Joanna Argasinska, Kevin Lamkiewicz, Manja Marz, Sam Griffiths-Jones, Claire Toffano-Nioche, Daniel Gautheret, Zasha Weinberg, Elena Rivas, Sean R Eddy, Robert D Finn, Alex Bateman, and Anton I Petrov. Rfam 14: expanded coverage of metagenomic, viral and microRNA families. *Nucleic Acids Research*, 49(D1):D192–D200, November 2020. doi:10.1093/nar/gkaa1047.

24 Robert J Klein and Sean R Eddy. Rsearch: finding homologs of single structured RNA sequences. *BMC bioinformatics*, 4(1):44, 2003.

25 Neocles B Leontis and Eric Westhof. Geometric nomenclature and classification of RNA base pairs. *RNA*, 7(4):499–512, 2001.

26 Yijin Liu, Timothy J Wilson, Scott A McPhee, and David MJ Lilley. Crystal structure and mechanistic investigation of the twister ribozyme. *Nature chemical biology*, 10(9):739–744, 2014.

27 László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006.

28 Xiang-Jun Lu, Harmen J. Bussemaker, and Wilma K. Olson. DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucleic Acids Research*, 43(21):e142–e142, July 2015. doi:10.1093/nar/gkv716.

29 R. B. Lyngsø and C. N. S. Pedersen. RNA pseudoknot prediction in energy-based models. *Journal of Computational Biology*, 7(3-4):409–427, 2000.

30 Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3(4), 2006.

31 Vladimir Reinharz, Antoine Soulé, Eric Westhof, Jérôme Waldispühl, and Alain Denise. Mining for recurrent long-range interactions in RNA structures reveals embedded hierarchies in network families. *Nucleic Acids Research*, 46(8):3841–3851, March 2018. doi:10.1093/nar/gky197.

32 Philippe Rinaudo, Yann Ponty, Dominique Barth, and Alain Denise. Tree decomposition and parameterized algorithms for RNA structure-sequence alignment including tertiary interactions and pseudoknots. In *Lecture Notes in Computer Science*, pages 149–164. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-33122-0_12.

33 Elena Rivas and Sean R Eddy. Parameterizing sequence alignment with an explicit evolutionary model. *BMC bioinformatics*, 16(1):406, 2015.

34 Neil Robertson and Paul D Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

35 Toshiki Saitoh, Ryo Yoshinaka, and Hans L. Bodlaender. Fixed-treewidth-efficient algorithms for edge-deletion to interval graph classes. In *WALCOM: Algorithms and Computation - 15th International Conference and Workshops, 2021, Proceedings*, volume 12635 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2021. doi:10.1007/978-3-030-68211-8_12.

36 Roman Sarrazin-Gendron, Hua-Ting Yao, Vladimir Reinharz, Carlos G. Oliver, Yann Ponty, and Jérôme Waldispühl. Stochastic sampling of structural contexts improves the scalability and accuracy of RNA 3d module identification. In *Lecture Notes in Computer Science*, pages 186–201. Springer International Publishing, 2020. doi:10.1007/978-3-030-45257-5_12.

37 Saad Sheikh, Rolf Backofen, and Yann Ponty. Impact Of The Energy Model On The Complexity Of RNA Folding With Pseudoknots. In Juha Kärkkäinen and Jens Stoye, editors, *CPM - 23rd Annual Symposium on Combinatorial Pattern Matching - 2012*, volume 7354 of *Combinatorial Pattern Matching*, pages 321–333, Helsinki, Finland, 2012. Juha Kärkkäinen, Springer. doi:10.1007/978-3-642-31265-6_26.

38 Kathryn D. Smith, Carly A. Shanahan, Emily L. Moore, Aline C. Simon, and Scott A. Strobel. Structural basis of differential ligand recognition by two classes of bis-(3'-5')-cyclic dimeric guanosine monophosphate-binding riboswitches. *Proceedings of the National Academy of Sciences*, 108(19):7757–7762, 2011. doi:10.1073/pnas.1018857108.

39 Yinglei Song, Chunmei Liu, Russell Malmberg, Fangfang Pan, and Liming Cai. Tree decomposition based fast search of RNA structures including pseudoknots in genomes. In *Computational Systems Bioinformatics Conference, 2005. Proceedings. 2005 IEEE*, pages 223–234. IEEE, 2005.

40 N. Sudarsan, E. R. Lee, Z. Weinberg, R. H. Moy, J. N. Kim, K. H. Link, and R. R. Breaker. Riboswitches in eubacteria sense the second messenger cyclic di-gmp. *Science*, 321(5887):411–413, 2008. doi:10.1126/science.1159519.

41 Rita Tamayo. Cyclic diguanylate riboswitches control bacterial pathogenesis mechanisms. *PLOS Pathogens*, 15(2):1–7, February 2019. doi:10.1371/journal.ppat.1007529.

42 Jinsong Tan and Louxin Zhang. The consecutive ones submatrix problem for sparse matrices. *Algorithmica*, 48(3):287–299, 2007.

43 J D Thompson, F Plewniak, and O Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, January 1999. doi:10.1093/bioinformatics/15.1.87.

44 Jelena Vucinic, David Simoncini, Manon Ruffini, Sophie Barbe, and Thomas Schiex. Positive multistate protein design. *Bioinformatics*, 36(1):122–130, June 2019. doi:10.1093/bioinformatics/btz497.

45 Wei Wang. *Practical sequence-structure alignment of RNAs with pseudoknots*. PhD thesis, Université Paris-Saclay, School of Computer Science, 2017.

46 Wei Wang, Alain Denise, and Yann Ponty. Licorna: alignment of complex rnas v1.0, 2017. URL: https://licorna.lri.fr.

47 M. S. Waterman. Secondary structure of single stranded nucleic acids. *Advances in Mathematics Supplementary Studies*, 1(1):167–212, 1978.

48 Mathias Weller, Annie Chateau, and Rodolphe Giroudeau. Exact approaches for scaffolding. *BMC Bioinformatics*, 16(S14), October 2015. doi:10.1186/1471-2105-16-s14-s2.

49 A. Xayaphoummine, T. Bucher, F. Thalmann, and H. Isambert. Prediction and statistics of pseudoknots in RNA structures using exactly clustered stochastic simulations. *Proc. Natl. Acad. Sci. U. S. A.*, 100(26):15310–15315, 2003.

50 Jinbo Xu. Rapid protein side-chain packing via tree decomposition. In *Lecture Notes in Computer Science*, pages 423–439. Springer Berlin Heidelberg, 2005. doi:10.1007/11415770_32.

51 Hua-Ting Yao, Jérôme Waldispühl, Yann Ponty, and Sebastian Will. Taming Disruptive Base Pairs to Reconcile Positive and Negative Structural Design of RNA. In *RECOMB 2021 - 25th international conference on research in computational molecular biology*, Padova, France, 2021.

A Editing Trees before the Diet

Any tree decomposition can be transformed into a binary one through the duplications of bags having more than 2 children. To do so in practice, one will, as long as the tree decomposition is not binary, apply the following transformation:

1. Find a bag X with children $Y_1, \ldots, Y_\Delta$ and $\Delta > 2$.
2. Introduce a new bag X' with the same content as X and locally modify the tree decomposition in the following way: X will now have Y_1 and X' as children, while X' will have $Y_2 \cdots Y_\Delta$.

When it is no longer possible to apply this transformation, the tree decomposition is binary. For each bag having originally $\Delta > 2$ children in the decomposition, $\Delta - 1$ new bags have been introduced. In total, with N_{bags} the original number of bags in the decomposition, strictly less than N_{bags} new bags have been introduced (each new bag is associated to an edge of the original tree decomposition).

This tranformation is in fact the first step towards obtaining a *nice tree decomposition* [7, 15].

A question that arises then is what impact these modifications may have on the output of Tree-Diet, when applied to the tree decomposition given as input. We argue that duplication operations (as used above to get a binary tree decomposition) can only improve the solution, i.e decrease the number of *lost* edges. Indeed, within the coloring formulation of the problem, new bags yield new opportunities for an edge to be *represented*, with both its end-points green in some bag. See Figure 10 for an illustration.

More generally, any operation on the input tree decomposition that does not suppress any of the original bags can only improve the solution to the Tree Diet problem. We do not tackle here the problem of finding the best edition operations to apply onto a tree decomposition given as input to Tree Diet, which is an a priori difficult task.

B Pseudo-code

Algorithm 1 present a pseudo-code of our dynamic programming algorithm for Tree Diet, with a memoization approach. The C++/pybind11 [22] implementation is available at https://gitlab.inria.fr/amibio/tree-diet.

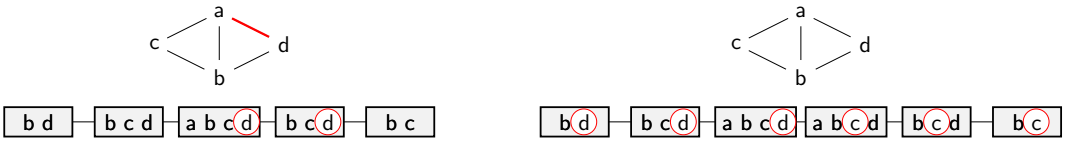


Figure 10 Left: A graph and a path-decomposition whose optimal 1-tree-diet loses an edge (ad). However, duplicating the bag abcd (right) yields a tree-decomposition with a lossless 1-tree-diet.

Note that the implementation allows to solve a more general *weighted* version of Tree Diet, where each edge is given a weight, and the objective is to find a $(tw - tw')$-diet of the input tree decomposition preserving a set of edges of maximum total weight.

In the context of RNA applications, this feature allows to favour as much as possible preservation of the backbone of RNA molecules, i.e. edges between consecutive nucleotides along the string, by assigning them a weight greater than the number of non-backbone edges.

Edge weights are passed to the function in the form of a dictionary/map W associating a real weight to each edge. Within Algorithms 1, the only place where it is taken into account is the the *count* function, which computes the weight of edges accounted for by the bag that is currently visited.

C Correctness of the rejection-based sampling of RNA designs

A recent method for RNA design, called RNAPond [51], implements a sampling approach to tackle the inverse folding of RNA. Targeting a secondary structure S of length n, it performs a Boltzmann-weighted sampling of sequences and, at each iteration, identifies Disruptive Base Pairs (DBPs) that are not in S, yet are recurrent in the Boltzmann ensemble of generated sequences. Those base pairs are then added to a set $\mathcal{D}$ of DBPs, and excluded in subsequent generations through an assignment of non-binding pairs of nucleotides, outside of $\mathcal{B} := \{(\mathsf{G},\mathsf{C}),(\mathsf{C},\mathsf{G}),(\mathsf{A},\mathsf{U}),(\mathsf{U},\mathsf{A}),(\mathsf{G},\mathsf{U}),(\mathsf{U},\mathsf{G})\}$.

At the core of the method, one finds a random generation algorithm which takes as input a secondary structure S and a set $\mathcal{D}$ of DBPs. The algorithm generates from the set $\mathcal{W}_{S,\mathcal{D}}$ of sequences $w \in \{\mathsf{A},\mathsf{C},\mathsf{G},\mathsf{U}\}^n$ which are: i) compatible with all $(i,j) \in S$, *i.e.* $(w_i, w_j) \in \mathcal{B}$; and ii) incompatible with all $(k,l) \in \mathcal{D}$, *i.e.* $(w_k, w_l) \notin \mathcal{B}$. The algorithm then enforces a (dual) Boltzmann distribution over the sequences in $\mathcal{W}_{S,\mathcal{D}}$:

$$\forall w \in \mathcal{W}_{S,\mathcal{D}} : \mathbb{P}(w \mid \mathcal{D}, S) = \frac{e^{-\beta.E_{w,S}}}{\mathcal{Z}_{S,\mathcal{D}}} \quad \text{with} \quad \mathcal{Z}_{S,\mathcal{D}} := \sum_{w' \in \mathcal{W}_{S,\mathcal{D}}} e^{-\beta.E_{w',S}} \tag{1}$$

where $\beta > 0$ is an arbitrary constant akin to a temperature. Yao *et al.* describe an algorithm which generates k sequences in $\Theta(k(n + |\mathcal{D}|))$ time, after a preprocessing in $\Theta(n.|\mathcal{D}|.4^{tw})$ time and $\Theta(n.4^{tw})$ space, where tw is the treewidth of the graph having edges in $S \cup \mathcal{D}$.

The discrepancy in the preprocessing and sampling complexities suggests an alternative strategy, utilizing rejection on top of a relaxed sampling. Namely, we consider a rejection algorithm, which starts from a relaxation $(S', \mathcal{D}')$ of the initial constraints $(S' \cup \mathcal{D}' \subset S \cup \mathcal{D})$, and iterates Yao *et al.*'s algorithm to generate sequences in $\mathcal{W}_{S',\mathcal{D}'} \supset \mathcal{W}_{S,\mathcal{D}}$, rejecting those outside of $\mathcal{W}_{S,\mathcal{D}}$, until k suitable ones are obtained. The rejection algorithm generates a given sequence $w \in \mathcal{W}_{S,\mathcal{D}}$ on its first attempt with probability $p := e^{-\beta.E_{w,S}}/\mathcal{Z}_{S',\mathcal{D}'}$ and, more generally, after r rejections with probability $(1-q)^r\, p$ with $q := \mathcal{Z}_{S,\mathcal{D}}/\mathcal{Z}_{S',\mathcal{D}'}$. The overall probability of emitting w is thus

$$p \cdot \sum_{r \geq 0} (1-q)^r = \frac{p}{q} = \frac{e^{-\beta.E_{w,S}}}{\mathcal{Z}_{S,\mathcal{D}}} = \mathbb{P}(w \mid \mathcal{D}, S).$$

Algorithm 1 Dynamic programming algorithm for TREE-DIET.

Input : Tree-decomposition $\mathcal{T}$, graph G, target width tw', edge weights W
Output : Maximum total weight of a set of realizable/non-lost edges in a $(tw - tw')$-diet of $\mathcal{T}$
Side-Product : A filled table $c[X_i, f]$, $\forall X_i$ bag and f coloring of X_i

```
1  Function optim_num_real_edges(X_i, f, G, tw', W):
2      if c[X_i, f] already computed then return c[X_i, f]; ;
3      if |f^{-1}(o) ∪ f^{-1}(r)| ≤ (|X_i| − tw' − 1) then
4          //not enough removals.;
5          c[X_i, f] = −∞;
6          return c[X_i, f];
7      end
8      if X_i == leaf then
9          c[X_i, f] = 0;
10         return c[X_i, f];
11     end
12     int ans = −∞;
13     for m ∈ orange_maps(X_i, f) do
14         int ans_m = 0;
15         for Y_j ∈ X_i.children do
16             int ans_j = −∞;
17             for f'_j ∈ compatible(f, m, X_i, Y_j) do
18                 int val = 0;
19                 val += count(f, f'_j, W);
20                 val += optim_num_real_edges(Y_j, f'_j, G, tw');
21                 if val ≥ ans_j then ans_j = val;
22                 ;
23             end
24             ans_m += ans_j;
25         end
26         if ans_m ≥ ans then ans = ans_m; ;
27     end
28     c[X_i, f] = ans
29     return c[X_i, f];
30 end
```

In other words, our relaxed generator coupled with the rejection step, represents an unbiased algorithm for the Boltzmann distribution of Eq. (1) over $\mathcal{W}_{S,\mathcal{D}}$.

Meanwhile, the average-case complexity can be impacted by the strategy. Indeed, the relaxed instance $(S', \mathcal{D}')$ can accelerate the preprocessing due to a reduced treewidth $tw' \leq tw$. The rejection step only increases the expected number of generations by a factor $\overline{q} := \mathcal{Z}_{S',\mathcal{D}'}/\mathcal{Z}_{S,\mathcal{D}}$, representing the inflation of the sequence space, induced by the relaxation of the constraints. Overall, the average-case time complexity of the rejection algorithm is in $\Theta(n \cdot |\mathcal{D}'| \cdot 4^{tw'} + k \cdot \overline{q} \cdot (n + |\mathcal{D}'|))$ time and $\Theta(n \cdot 4^{tw'})$ space. This space improvement is notable when $tw' < tw$, and could be key for the practical applicability of the method, especially given that memory represents the bottleneck of most treewidth-based DP algorithms.

D Lower bound for the min. alignment cost from simplified models

Here, we justify the filtering strategy described in Section 5.2.2. Namely, we formally prove that, given a structured RNA S and a targeted genomic region w, a lower bound for the minimal alignment cost of S and w can be obtained from the minimal alignment cost of some $S' \subseteq S$ and w. If this lower bound for $S' \subseteq S$ is higher than the specified cutoff ε, then there is no need to align w to S the full model, as the resulting cost is guaranteed to stay above the selection cutoff ε.

Let S be an arc-annotated sequence of length m (S_i denotes the ith character of S), w be a target (flat) sequence of length m, and $\mu : [1, n] \to [1, m] \cup \{\perp\}$ represents an alignment[2]. We consider the following cost function, adapted from [32], which quantifies the quality of an alignment μ for S and w:

$$C(S, w, \mu) = \sum_{\substack{i \text{ unpaired in } S, \\ k:=\mu_i}} \gamma(S_i, w_k) + \sum_{\substack{(i,j)\in S, \\ (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l) + \sum_{g\in gaps(S)} \lambda_q(g) + \sum_{g\in gaps(w)} \lambda_T(g)$$

where

- $\gamma(a, b)$ returns the *substitution cost* which penalizes (mismatches) or rewards (matches) the substitution of a into b (set to 0 and handled in gaps if $b = \perp$);
- $\phi(a, b, c, d)$ return a *base pair substitution cost*, penalizing (arc breaking) or rewarding (conservation or compensatory mutations) the transformation of nucleotides (a, b) into nucleotides/gaps (c, d) (set to 0 and handled in gaps if $(c, d) = (\perp, \perp)$);
- λ_S and λ_T penalize gaps introduced by μ respectively in S and w (affine cost model).

Given this definition, consider a simplified model $S' \subset S$, associated with a minimal cost

$$c' := \min_{\mu} C(S, w, \mu)$$

and denote by $c^\star$ the minimal cost of the full model S, we have the following inequality.

▶ **Proposition 14.**

$$c' - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S}} \max_b \gamma(S_i, b) + \sum_{(i,j)\in S\setminus S'} \min_{a,b} \phi(S_i, S_j, a, b) \leq c^\star \tag{2}$$

Proof. For any alignment, we have, per the definition of $C(S, w, \mu)$:

$$C(S, w, \mu) = C(S', w, \mu) - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S, \\ \text{and } k:=\mu_i}} \gamma(S_i, w_k) + \sum_{\substack{(i,j)\in S\setminus S' \\ \text{s.t. } (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l).$$

Minimizing over all alignment μ, one obtains

$$\min_{\mu} C(S, w, \mu) = \min_{\mu} C(S', w, \mu) - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S, \\ \text{and } k:=\mu_i}} \gamma(S_i, w_k) + \sum_{\substack{(i,j)\in S\setminus S' \\ \text{s.t. } (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l).$$

[2] An alignment μ is subject to further constraints, notably including some restricted form of monotonicity, when represented as a function. However, those constraints are reasonably intuitive and we omit them is this discussion for the sake of simplicity.

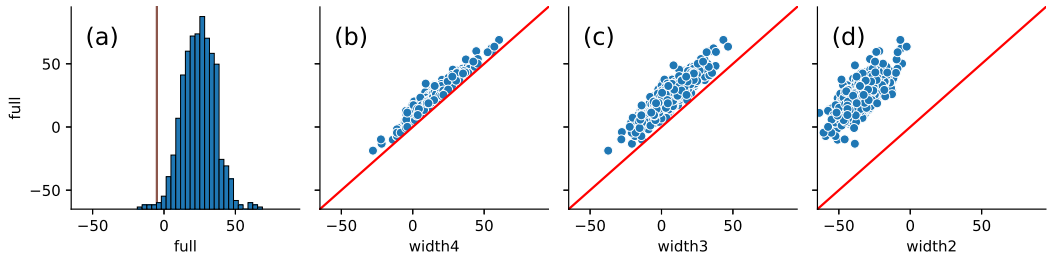


Figure 11 (a) Histogram of alignment scores obtained by aligning the full structure ($tw = 5$) model of the Twister ribozyme (pdb-id: 4OJI) with $\kappa \cdot n$-sized windows in a 10kb region of the 5^{th} chromosome of *S. bicolor*. A vertical line is positioned at the ϵ threshold. (b;c;d) Corrected alignment scores obtained for reduced-treewidth models for each window, plotted against the corresponding score of the full model. The corrected alignment score indeed acts as a lower bound to the full-model score (points above the $y = x$ red line), allowing a iterative filtering strategy.

Independently minimizing each term of the right-hand-side, we obtain a first lower bound

$$c^\star \geq c' - \max_{\mu} \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S, \\ \text{and } k:=\mu_i}} \gamma(S_i, w_k) + \min_{\mu} \sum_{\substack{(i,j)\in S\setminus S' \\ \text{s.t. } (k,l):=(\mu_i,\mu_j)}} \phi(S_i, S_j, w_k, w_l).$$

further coarsened by an independent optimization of the elements in the sums

$$\begin{aligned} c^\star &\geq c' - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S}} \max_{\mu} \gamma(S_i, w_k) + \sum_{(i,j)\in S\setminus S'} \min_{\mu} \phi(S_i, S_j, w_k, w_l) \\ &= c' - \sum_{\substack{i \text{ unpaired in } S,' \\ \text{paired in } S}} \max_{a} \gamma(S_i, a) + \sum_{(i,j)\in S\setminus S'} \min_{a,b} \phi(S_i, S_j, a, b). \end{aligned}$$

where the last line is obtained by considering the worst-case contributors to nucleotides and base pairs substitutions. Importantly, the right-hand side no longer depends on μ any more, and can be used to easily computed a corrected score/lower bound. ◀

The corrected expression, shown in the left hand side of Equation (2) allows, when lower than a cutoff ε, to safely discard w as a potential hit for the full model S. This corrected score score is plotted in Figure 9A, allowing for a gradual reduction of the search space for ε-admissible hits. We show in Figure 11 the corrected scores obtained for simplified structures S' of various treewidths, plotted against the scores of the full target structure.

Space-Efficient Representation of Genomic k-Mer Count Tables

Yoshihiro Shibuya
LIGM, CNRS, Univ. Gustave Eiffel, Marne-la-Vallée, France

Djamal Belazzougui
CAPA, DTISI, Centre de Recherche sur l'Information Scientifique et Technique, Algiers, Algeria

Gregory Kucherov
LIGM, CNRS, Univ. Gustave Eiffel, Marne-la-Vallée, France
Skolkovo Institute of Science and Technology, Moscow, Russia

Abstract

Motivation. k-mer counting is a common task in bioinformatic pipelines, with many dedicated tools available. Output formats could rely on quotienting to reduce the space of k-mers in hash tables, however counts are not usually stored in space-efficient formats. Overall, k-mer count tables for genomic data take a considerable space, easily reaching tens of GB. Furthermore, such tables do not support efficient random-access queries in general.

Results. In this work, we design an efficient representation of k-mer count tables supporting fast random-access queries. We propose to apply Compressed Static Functions (CSFs), with space proportional to the empirical zero-order entropy of the counts. For very skewed distributions, like those of k-mer counts in whole genomes, the only currently available implementation of CSFs does not provide a compact enough representation. By adding a Bloom Filter to a CSF we obtain a Bloom-enhanced CSF (BCSF) effectively overcoming this limitation. Furthermore, by combining BCSFs with minimizer-based bucketing of k-mers, we build even smaller representations breaking the empirical entropy lower bound, for large enough k. We also extend these representations to the approximate case, gaining additional space. We experimentally validate these techniques on k-mer count tables of whole genomes (*E.Coli* and *C.Elegans*) as well as on k-mer document frequency tables for 29 *E.Coli* genomes. In the case of exact counts, our representation takes about a half of the space of the empirical entropy, for large enough k's.

2012 ACM Subject Classification Applied computing

Keywords and phrases k-mer counting, data structures, compression, minimizers, compressed static function, Bloom filter, empirical entropy

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.8

Supplementary Material *Software (Source Code)*: `https://github.com/yhhshb/locom.git`

Funding *Gregory Kucherov*: partially funded by RFBR project 20-07-00652 and joint RFBR and JSPS project 20-51-50007.

1 Introduction

Nowadays, many bioinformatics pipelines rely on k-mers to perform a multitude of different tasks. Representing sequences as sets of words of length k generally leads to more time-efficient algorithms than relying on traditional alignments. For these reasons, alignment-free algorithms have started to replace their alignment-based counterparts in a wide range of practical applications, from sequence comparison and phylogenetic reconstruction [34, 36,

7, 10] to finding SNPs [28, 15] and other tasks. These algorithms often require to associate some kind of information to k-mers involved in the analysis, that is, to build maps where keys are k-mers.

Typical values to associate to k-mers are their frequencies in a particular dataset. Actual counting can be performed by one of several available k-mer counting tools developed in recent years [16, 29, 24, 33]. Counting tables generally include both k-mers and counts requiring considerable amounts of disk space to be stored. For example, the KMC output for a human genome with $k = 32$ weights in at around 28GB.

In many scenarios, it is useful to store k-mer count tables independently from the sequence data in order to retrieve counts multiple times and avoid redundant heavy counting steps. The original sequence dataset can be used as the primary source of k-mers while a random-access data structure will then allow retrieving their counts efficiently. Among potential applications of such a data structure, efficient representation of k-mer counts can be useful for read correction [21]. More generally, information about k-mer counts is increasingly used in other applications [34, 28, 15, 23, 13, 14, 26] which can benefit from space-efficient solutions.

In many applications, space can be significantly reduced by representing the mapping without actually storing k-mers. Minimal Perfect Hash Functions (MPHFs for short) implement such an approach [27, 37, 9]. A MPHF bijectively maps each item from a set S to an index in the range $[0, |S| - 1]$. Any additional information can then be stored in an array indexed by the values returned by the MPHF. Practical applications of MPHFs in bioinformatics include [22] and [38].

Frequency distributions of k-mer counts in genomic data are low-entropy distributions, unless k is small. It is in fact known that k-mer counts for genomic sequences follow a heavy-tail distribution [6, 4]. For large enough k-mer lengths, counts tend to follow a skewed power-law distribution with the majority of k-mers occurring only few times, mostly once. For these reasons, the multiset of counts of k-mers will typically have a fairly low empirical zero-order entropy and therefore could be effectively compressed to save further space. However, simply compressing the count array does not maintain queryability, which requires specialized algorithms for this task. Note also that MPHFs themselves encompass a non-negligible space overhead, with the theoretical minimum of 1.44 bits/key. BBHash [22], a popular MPHF implementation for bioinformatics, requires around 3 bits per element in addition to the space for storing the values.

Maps on static sets of keys can be encoded using so-called Static Functions [1, 11]. Unlike MPHFs, the actual hash function and the values are encoded into the same structure. Compressed Static Functions (CSFs) try to benefit from the compressibility of the value array and approach the number of bits defined by the empirical entropy. This feature makes them particularly useful for representing different k-mer annotations, such as counts or presence information across sequences of a given sample [23, 13, 14, 26]. CSFs can be used as readily available drop-in replacements of MPHFs whenever the set of queryable k-mers is known in advance. Solutions based on MPFHs and Static Functions assume that only k-mers present in the datasets can be queried for their frequency. In many cases, this is not restrictive as the "universe" of query k-mers can be effectively specified: for example, it can be restricted to k-mers from a given genome or a pan-genome. It is also conceivable to add an appropriate structure providing presence-absence information, in order to benefit from the reduction of space provided by a compact count representation.

The goal of this paper is to study data structures for storing genomic k-mer count tables using the smallest possible space. For this, we combine different tools: CSFs and Bloom filters on the one hand, and minimizers on the other hand.

Our first contribution is the enhancement of CSFs with a Bloom Filter to deal with datasets of very small entropy and to achieve better space usage. We chose an *E.Coli* and a *C.Elegans* genome to test our implementation. With these examples, we demonstrate the advantages of our BCSF implementation over a simple CSF.

Our second improvement takes advantage of the fact that similar k-mers tend to have identical (or similar) counts (see also [23]). Following this insight, we introduce a minimizer-based bucketing scheme to cluster together count values of k-mers with the same minimizer. A similar idea is used by some k-mer counting algorithms [29, 16, 18] with the difference that in our case buckets contain counts rather than k-mers themselves. By choosing a representative value for each bucket, we obtain a "bucket table" that we encode using Bloom-enhanced CSF. Bucketing allows us to break the empirical entropy lower bound, as we show on both *E.Coli* and *C.Elegans* examples. To demonstrate the advantages of bucketing on higher-entropy distributions, we apply it to represent a table of "document frequencies" [12, 5, 25, 17] of each k-mer across multiple strains of *E.Coli*.

We study different implementation schemes based on these ideas and compare their space performance, as well as associated query time. For large enough k (and large enough minimizers lengths), we are able to consistently break the barrier of the empirical entropy of the input table. To the best of our knowledge, this is the first implementation proposing such a compact representation. We also study an extension to the approximate case when query answers are within a pre-defined absolute error from the true value, for which we achieve an even smaller space.

2 Technical preliminaries

Throughout the paper we consider a k-mer count table to be an associative array f mapping a set of k-mers K, considered static, to their counts, i.e. number of occurrences in a given dataset. $||f||_1$ stands for the L1-norm of f, that is $\sum_{q \in K} f(q)$.

2.1 Minimizers

Minimizers are a popular technique used in different applications involving k-mer analysis. Given a k-mer q of length k, its minimizer of length m, with $m \leq k$, is the smallest substring of q of length m w.r.t. some order defined on m-mers. The use of minimizers for biosequence analysis goes back to [30], whereas a similar concept, named *winnowing*, has been earlier applied in [32] to document search. The guiding idea is that a minimizer can be considered as a "footprint" (hash value) of a corresponding k-mer so that similar (e.g. neighboring in the genome) k-mers are likely to have the same minimizer. Thus, if the order of m-mers is randomly chosen, minimizers can be seen as a specific instance of locality-sensitive hashing, in particular of MinHash sketching [3].

Minimizers have been successfully applied to various data-intensive sequence analysis problems in bioinformatics, such as metagenomics (Kraken [35]) or minimizing cache misses in k-mer counting (KMC [16]), or mapping and assembling long single-molecule reads [19, 20]. Recently, there has been a series of works on both theoretical and practical aspects of designing efficient minimizers, see e.g. [39, 8] and references therein.

2.2 Bloom filters

Bloom filter is a very common probabilistic data structure that supports membership queries for a given set S drawn from a large universe U, admitting a controlled fraction of *false positives*. To insure a false positive rate ε, that is the probability ε for an item from $U \setminus S$ to

be erroneously classified as belonging to S, a Bloom filter B requires $|S| \log e \log \frac{1}{\varepsilon}$ bits, i.e. $\approx 1.44 \log \frac{1}{\varepsilon}$ bits per element of S. For a set $T \subseteq U \setminus S$, we denote $FP_B(T)$ the set of false positives of T, of expected size $\varepsilon|T|$.

2.3 Compressed static functions

A static function (SF) is a representation of a function defined on a given subset S of a universe U such that an invocation of the function on any element from S yields the function value, while an invocation on an element from $U \setminus S$ produces an arbitrary output. The problem has been studied in several works (see references in [1, 11]) resulting in several solutions that allow function values to be retrieved without storing elements of S themselves. One natural solution comes through MPHFs: one can build an MPHF for S and then store function values in order in a separate array. This solution, however, incurs an overhead associated with the MPHF, known to be theoretically lower-bounded by about 1.44 bits per element of S.

This overhead is especially unfortunate when the distribution of values is very skewed, in which case the value array may be compressed into a much smaller space. Compressed Static Functions try to solve this problem by proposing a static function representation whose size depends on the *compressed* value array. The latter is usually estimated through the zero-order empirical entropy, defined by $H_0(f) = \sum_{\ell \in L} \frac{|f^{-1}(\ell)|}{|K|} \log(\frac{|K|}{|f^{-1}(\ell)|})$, where L is the set of all values (i.e. $L = \{f(t) \,|\, t \in K\}$) and $f^{-1}(\ell) = \{t \,|\, f(t) = \ell\}$ is the set of k-mers with count ℓ. $|K| \cdot H_0(f)$ can be viewed as a lower bound on the size of compressed value array, in absence of additional assumptions. Thus, the goal of CSFs is to approach the bound of $H_0(f)$ bits per element as closely as possible, in representing a static function f.

An overview of different algorithmic solutions for SFs and CSFs is out of scope of this paper, we refer the reader to [1, 11] and references therein. [1] proposed a solution for CSF taking an asymptotically optimal $nH_0(f) + o(nH_0(f))$ space (n size of the underlying value set), however the solution is rather complex and probably not suitable for practical implementation. As of today, to our knowledge, the only practical implementation of a CSF is GV3CompressedFunction [11], found in the Java package Sux4J (`https://sux.di.unimi.it/`). Although entropy-sensitive, the method of [11], however, has an intrinsic limitation of using at least 1 bit per element, due to involved coding schemes. This is a serious limitation when dealing with very skewed distributions of values, where one value occurs predominantly often and the empirical entropy can be much smaller than 1. This is precisely the case for count distributions in whole genomes, studied in this paper.

3 Representation of low-entropy data

As mentioned earlier, Compressed Static Functions (CSF) of [11] do not properly deal with datasets generated by low-entropy distributions, in particular with entropy smaller than 1. This case occurs when the dataset has a dominant value representing a large fraction (say, more than a half) of all values. This is typically the case with genomic k-mer count data, especially whole-genome data, where a very large fraction of k-mers occur just once. For example, in *E.Coli* genome (≈5.5Mbp), about 97% of all distinct 15-mers occur once, and only the remaining 3% of 15-mers occur more than once.

For such datasets, the method of [11] does not approximate well the empirical entropy, as it cannot achieve less than 1 bit per key. Here we propose a technique to circumvent this deficiency in order to achieve, in combination with CSFs of [11], a compression close to the empirical entropy.

We build a Bloom filter for all k-mers whose value is not the dominant one, and then construct a CSF on all positives (i.e. true and false positives) of this filter. At query time, we first check the query k-mer against the Bloom filter and, if the answer is positive, recover its value with the CSF.

Formally, let K_0 be the k-mers with the most common frequency. Let $|K_0| = \alpha|K|$. Assume that our Bloom filter implementation takes $C_{BF} \log \frac{1}{\varepsilon}$ bits per key (a standard value is $C_{BF} = \log e \approx 1.44$) and our CSF implementation takes C_{CSF} bits per key. As explained earlier, C_{CSF} depends on the data, however, for the purpose of this section, we abstract from this dependency. In Sec. 6 below, we will specify C_{CSF} for the implementation we use.

We store keys $K \setminus K_0$ in a Bloom filter B and build a CSF for $(K \setminus K_0) \cup FP_B(K_0)$. The total space is

$$C_{BF}(1-\alpha)|K| \log \frac{1}{\varepsilon} + C_{CSF}|K|((1-\alpha) + \varepsilon\alpha). \tag{1}$$

The Bloom filter enables space saving only if α is sufficiently large. To decide if we need a Bloom filter, we have to verify if the inequality

$$C_{BF}(1-\alpha)|K| \log \frac{1}{\varepsilon} + C_{CSF}|K|((1-\alpha) + \varepsilon\alpha) < C_{CSF}|K|. \tag{2}$$

holds for some $\varepsilon < 1$. Note again that C_{CSF} on the left and right sides are not exactly the same in reality, however assuming them the same is not reductive because of specificities of the CSF implementation we use. We will elaborate further on this later on. Then (2) rewrites to

$$\frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} \log \frac{1}{\varepsilon} + \varepsilon < 1. \tag{3}$$

Using simple calculus, we obtain that if $\frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} > \ln 2$ (that is, $\frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} \log e > 1$), then (3) never holds for $0 < \varepsilon < 1$. The left-hand side of (3) reaches its minimum for

$$\varepsilon_0 = \frac{C_{BF}}{C_{CSF}} \frac{1-\alpha}{\alpha} \log e, \tag{4}$$

and this minimum is smaller than 1 if $\varepsilon_0 < 1$. We conclude that in order to decide if a Bloom filter enables space saving, we have to check the value ε_0. If $\varepsilon_0 \geq 1$, we do not need a Bloom filter, otherwise we need one with $\varepsilon = \varepsilon_0$. This shows that a Bloom filter is needed whenever

$$\alpha > \frac{C_{BF} \log e}{C_{CSF} + C_{BF} \log e} \tag{5}$$

For $C_{BF} = C_{CSF}$, this gives $\alpha > 0.59$.

In the rest of the paper we use the term *Bloom-enhanced Compressed Static Function*, BCSF for short, to speak about CSF possibly augmented by a prior Bloom filter, as described in this section. Algorithm 1 summarizes the computation of the BCSF data structure.

4 Using minimizers

4.1 Bucketing

A key idea to reduce the computational burden of counting k-mers, is to use minimizers to bucket k-mers and split the counting process across multiple tables (cf e.g. [16]). Here we use the same principle to bucket count values instead of k-mers themselves. Let $M_m(K) =$

Algorithm 1 BCSF construction.

Data: A counting table f between keys and (integer) values
Result: A BCSF for f
Compute R, the k-mer spectrum of f;
Compute ε with (4);
if $\varepsilon < 1$ **then**
  let $K_0 \subseteq K$ be the k-mers with the most common item in R;
  $C = K \setminus K_0$;
  Initialise Bloom filter B of $\lceil \log(e)|C| \log_2(\frac{1}{\varepsilon})/\rceil$ bits;
  Insert C into Bloom filter;
  Compute $E = FP_B(K_0)$;
  $S = C \cup E$;
else
  $S = K$
end
Construct CSF for S;

$\{\mu_m(q) \mid q \in K\}$ be the set of minimizers of all k-mers of K of a given length $m < k$. We map the input set K onto the (smaller) set $M_m(K)$. To each minimizer $s \in M_m(K)$, corresponds the bucket $\{f(q) \mid q \in K, \mu_m(q) = s\}$. We call a minimizer and the corresponding bucket *ambiguous* if this set contains more than one value. The guiding idea is to replace f by a mapping g of $M_m(K)$ to $\mathbf{N}$. Querying value $f(q)$ for a k-mer $q \in K$ will reduce to first querying $g(\mu_m(q))$ and then possibly "correcting" the retrieved value. In other words, for each bucket, we replace its set of counts with one representative value and we split the query into two operations: retrieving the representative from the buckets and correcting to reconstruct the original value. The rationale is that k-mers having the same minimizer tend to have the same count allowing multiple values to be dealt with by a single bucket.

We consider two implementations which differ on how the representatives are chosen and how corrections are applied. In the *first implementation*, that we name FIL (from FILtration, see Algorithm 2), $g(s)$ is defined to be the majority value among all values of its bucket, ties resolved arbitrarily. In particular, if s is a non-ambiguous minimizer then $g(s)$ is set to the unique value of the bucket. In practice, computing the majority value may incur a computational overhead as this requires storing bucket values until all values are known. An option to cope with this is to use the "approximate majority" computed by the online Boyer-Moore majority algorithm [2].

We then store a "correcting mapping" $h : K \to \mathbf{N}$ defined by $h(q) = f(q) - g(\mu_m(q))$. That is, we construct another counting table h where each k-mer is associated to the correction factor $h(q)$, which, added to the representative $g(s)$ results in the original count c. Both mappings g and h are stored using BCSFs.

The rationale for this scheme is that, due to properties of minimizers, $h(q)$ is supposed to be often 0, which makes h well compressible using BCSF. Note that because of the majority rule, 0 will always be the majority value of h. Therefore, the Bloom filter of the BCSF storing h (if any) will hold k-mers q with $f(q) \neq g(\mu_m(q))$ (i.e. $h(q) \neq 0$). Then the CSF will store h restricted to k-mers with $h(q) \neq 0$ together with a subset of k-mers (false positives of the Bloom filter) for which $h(q) = 0$.

In our *second implementation*, named AMB (from AMBiguity, see Algorithm 3), for non-ambiguous minimizers u, $g(u)$ is again defined to be the unique value of the bucket. For ambiguous minimizers v, we set $g(v) = 0$, where 0 is viewed as a special value marking

Algorithm 2 FIL construction algorithm.

Data: A mapping f of keys to (integer) values, a minimizer length m_0
Result: FIL compressed structure
Sort L by increasing order;
$T = f$;
Initialise an array A of buckets;
for (q, c) *in* f **do**
  $z = \mu_{m_0}(q)$ Insert c into $g(z)$;
end
for b *in* A **do**
  Select representative r of bucket b by majority rule;
end
Compress A by using BCSF;
Create output table O;
for (q, c) *in* f **do**
  if $g(\mu_m(q)) \neq c$ **then**
    Write q and $c - g(\mu_m(q))$ to O;
  end
end
Compress O by using BCSF;

ambiguous buckets (k-mers with count 0 are not present in the input). This has the disadvantage of providing no information about the values of ambiguous buckets, and also of making g less compressible (because of an additional value). On the other hand, this has the advantage of distinguishing between ambiguous and non-ambiguous buckets and allows the query to immediately return the answer for k-mers hashing to non-ambiguous buckets. As a consequence, unambiguous k-mers are not propagated to the second layer, and if $g(\mu_m(q)) \neq 0$ it can be immediately returned as $f(q)$. We then have to store mapping f restricted only to k-mers from ambiguous buckets, which we denote $\tilde{f}$. Both mappings g and $\tilde{f}$ are stored using BCSFs.

4.2 Cascading

An intermediate layer corresponding to a minimizer length $m < k$, introduced in Section 4.1, can be viewed as a "filter" providing values for some k-mers and "propagating" the other k-mers to the next layer. Therefore, both implementations can be cascaded into more than one layer. This construction is reminiscent of the BBHash algorithm [22] or to cascading Bloom filters from [31].

For $m_1 < m_2 < ...m_\ell \leq k$, each layer i is then input some map f_{i-1} defined on a subset of k-mers $K_{i-1} \subseteq K$ ($f_0 = f$, $K_0 = K$) and outputs another map f_i defined on a smaller subset $K_i \subseteq K_{i-1}$. Each layer stores a bucket table for minimizers $M_{m_i}(K) = \{\mu_{m_i}(q) \,|\, q \in K_{i-1}\}$. The specific definition of f_i and K_i depends on the implementation.

The multi-layer scheme is particularly intuitive for the AMB implementation, where each layer stores a unique value for non-ambiguous minimizers and a special value 0 otherwise. In this case, K_i consists of those k-mers of K_{i-1} hashed to ambiguous buckets, and f_i is simply a restriction of f to those k-mers. Algorithm 3 shows a pseudo-code of multi-level AMB extended to the approximate case (see Section 5 below). The multi-layer version of the FIL scheme is shown in Appendix (Algorithm 4).

Algorithm 3 AMB multi-layer construction algorithm. Exact AMB can be obtained by setting $\delta = 0$.

Data: A mapping f of keys to (integer) values, a list L of minimizer lengths, a maximum absolute error δ
Result: One BCSFs for each layer
Sort L by increasing order;
$T = f$;
for m *in* L **do**
  Initialise an array A of buckets;
  for (q, c) *in* T **do**
    $z = \mu_m(q)$;
    if z *in* A **then** $(r_{min}, r_{max}) = g(z)$;
    else $(r_{min}, r_{max}) = (\infty, -\infty)$;
    $g(z) = (\min(r_{min}, c), \max(r_{max}, c))$;
  end
  for $b = (r_{min}, r_{max})$ *in* A **do**
    if $r_{max} - r_{min} > \delta$ **then** $b = 0$;
    else $b = r_{min}$;
  end
  Compress A by using BCSF;
  Create output table O;
  for (q, c) *in* T **do**
    if $g(\mu_m(q)) = 0$ **then**
      Write q and c to O;
    end
  end
  $T = O$;
end

5 Extension to approximate counts

In addition to cascading, the AMB implementation can also be easily extended to work as an approximation algorithm. Consider, to this end, the layered bucketing procedure desribed in 4.2. In the exact case, a bucket is marked as colliding whenever it contains two or more distinct count values. In the approximate case, a collision is defined if a bucket contains a pair of counts, c_i, c_j such that $|c_i - c_j| > \delta$ with δ a pre-defined maximum absolute error. With this modification, the algorithm guarantees to output a value within the absolute error δ from the true count.

Implementing this modification is simple when the majority is computed with the Boyer-Moore majority vote algorithm. Another option is to define $g(s)$ to be the minimum, instead of majority. The rationale of using minimum is the decreasing behavior of k-mer spectra which implies that smaller counts are more frequent and therefore more likely to constitute the majority. It is then sufficient to only remember the maximum $max(s)$ and minimum $min(s)$ values seen by each bucket and check if $max(s) - min(s) > \delta$. If that is the case, then the bucket is marked as colliding, otherwise $min(s)$ is chosen as representative. The latter solution is reported in Algorithm 3.

6 Experimental results

We report experiments on three datasets, two with lower and one with higher empirical entropy. The first is the k-mer counts computed on the *Sakai* strain of *E.Coli* from [36] (NCBI accession number B000007). The second one is a full genome of *C.Elegans*, strain *Bristol N2* downloaded from RefSeq (accession number GCF_000002985.6). The last one is the whole dataset from the same paper [36], hereafter referred to as "df", of k-mer "document frequencies" across 29 *E.Coli* genomes made of approximately 25 million k-mers. Here the document frequency of a k-mer is the number of genomes containing this k-mer.

Experiments were performed on a machine equipped with an Intel® Core™ i7-4770k (Haswell), 8 GB of RAM and Kubuntu 18.04. All construction code is written in python, except for the CSF part which is handled by a simple Java program using Sux4J [11]. Time measurements are performed by a program written in C using the code provided by Sux4J for reading and querying its CSFs. We use xxHash[1] as $\mu_m(q)$ to define an ordering over the minimizers of a given k-mer q. All code is available at `https://github.com/yhhshb/locom.git`.

We only report the best methods for each case with the following naming convention:

- CSF: baseline CSF implementation from Sux4J.
- BCSF: extended CSF with Bloom Filter from Section 3. It may get reduced to a simple CSF if the Bloom Filter is not useful.
- FIL m_1 k: our *first implementation*, saving into each bucket a majority-selected representative and saving corrections into its second layer.
- FIL m_1 m_2 k: same as before but with an additional layer.
- AMB m_1 k: our *second implementation*, selecting each representative by minimum and marking colliding buckets with a special value.
- AMB m_1 m_2 k: same as before but with an additional layer.

In order to apply equation (4), we have to have estimates of C_{BF} and C_{CSF}, that is, estimates of the number of bits per element taken by our implementations of Bloom filter and CSF. For C_{BF}, we have $C_{BF} = 1.44$ corresponding to the theoretical coefficient. For C_{CSF}, we empirically estimated the value as a function of the empirical entropy H_0 of experimental data, and obtained the following estimate:

$$C_{CSF} = \begin{cases} 0.22H_0^2 + 0.18H_0 + 1.16, & \text{if } H_0 < 2 \\ 1.1H_0 + 0.2, & \text{otherwise.} \end{cases} \tag{6}$$

To better understand how different minimizer lengths affect the final compression ratio, we ran FIL and AMB on all possible combinations of 2 and 3 minimizers lengths for $k =$ 10,11,12,13,15,18,21.

6.1 Compression of skewed data

Figure 1 reports memory usage for $k = 13, 15, 18, 21$, when compressing the Sakai dataset. As mentioned earlier, simple CSF takes more than 1 bit/k-mer, which is considerably larger than the entropy of our data. Bloom-enhanced CSF (BCSF) considerably reduces space bringing it closer to the entropy value. For relatively small k's ($k = 13$) AMB and FIL give almost the same results as BCSF, that is, bucketing is not helpful. For larger k's, however,

[1] `https://github.com/Cyan4973/xxHash`

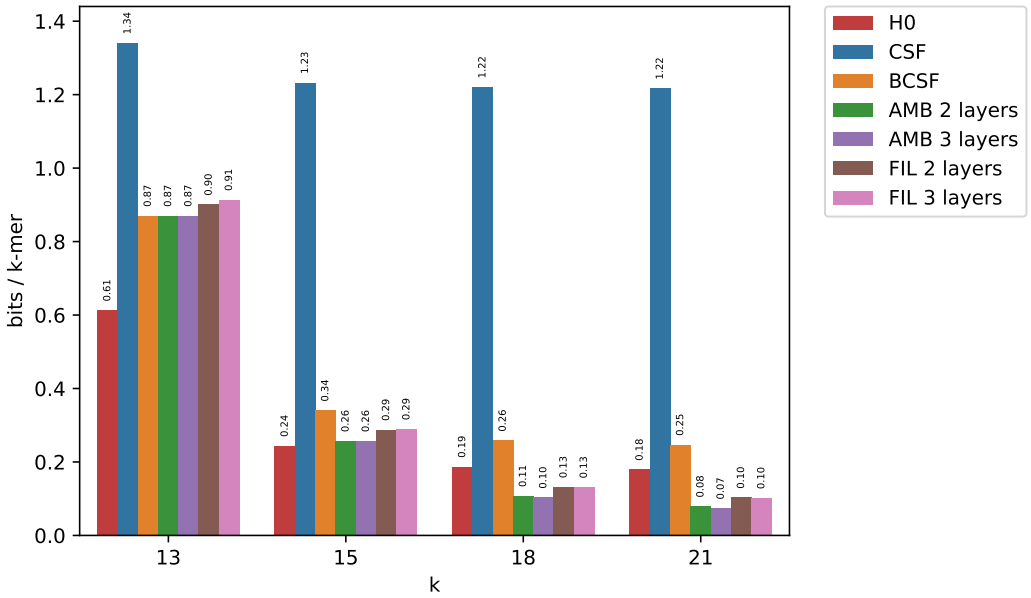


Figure 1 Results for the Sakai dataset for big values of k. For presentation purposes, H0 is represented as an additional red column in each subgroup.

minimizer-based schemes, AMB and FIL, lead to a reduction of space, eventually breaking the entropy barrier for larger values of k ($k = 18, 21$). This demonstrates that for larger k's, minimizers provide an effective way of factoring the space of k-mers in such a way that k-mers with equal counts tend to have the same minimizer.

More in detail, for larger k, the overwhelming majority of buckets are unambiguous (e.g. more than 99% of them, for $k = 18, m = 13$). As a consequence, AMB "filters out" a very large number of k-mers at the first layer, propagating to the second layer only a small fraction of them – those corresponding to ambiguous buckets. The special collision value then becomes the dominant value of the bucket table, making it highly compressible with BCSF. Note also that due to the skewedness of the distribution, i.e. the prevalence of one value, the k-mer tables of the last layer are well compressible as well. Altogether, this enables breaking the empirical entropy lower bound. The situation is similar for FIL: its first layer is even better compressible than the one of AMB, due to the absence of the additional special value which makes the table of AMB slightly less compressible. On the other hand, the BCSF of the second layer table of FIL turns out to take more space than that of AMB. This is because its Bloom filter operates on the large set of all k-mers, which implies a very small value of ε to keep the set of false positives under control, and as a consequence, a relatively large Bloom filter. Overall, FIL turns our to yield a slightly larger space.

The advantages of AMB and FIL tend to vanish for smaller values of k. For small k's, none of the methods beats the empirical entropy, which means that minimizers do not provide an efficient mean to factor the space of k-mers according to count values. We observe that in this case, applying BCSF to the input table provides the most efficient solution.

Since longer k-mers lead to more skewed data, and by extension, to a smaller entropy, both AMB and FIL better compress whole genome count tables for increasing ks. In order to test our algorithms in a more complex and challenging situation, we chose to compress the reference genome of *C.Elegans* (around 100Mbp). Taking into account the considerations

presented previously, we quickly found that the best results for AMB and FIL were given by $k = 18$ when using 2 layers. We randomly chose $m_1 = 19$ and $m = 21$ for three-layer AMB and FIL, respectively. Figure 2 demonstrates that our algorithms are not limited to bacterial genomes. Larger values of k only reduce the entropy of the data, leading to more succinct representations whereas simple CSF could not go below 1.2 bits/k-mer.

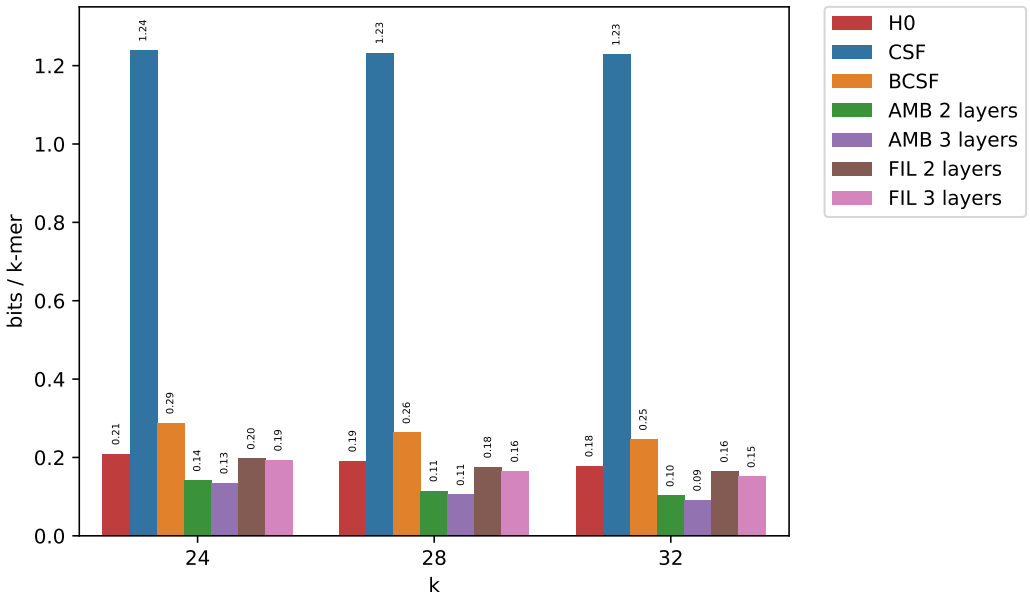


Figure 2 Results when compressing the reference genome of *C.Elegans*.

6.2 Compression of higher entropy data

With very skewed data, collisions of k-mer counts may happen between unrelated k-mers simply because one counter value strongly dominates the spectrum. In order to demonstrate that minimizers are useful as well for less skewed distributions than whole genome count tables, we applied our methods to the df dataset, see Figure 3.

The use of minimizers for larger k's, proves to be beneficial again, with AMB and FIL requiring much less space that the empirical entropy of the data. A similar scenario to the previous case represents itself for relatively small k ($k = 13$), for which both AMB and FIL do not have an advantage over a simpler (B)CSF. For even smaller k-mers (B)CSF remains the best option (results not shown).

The seemingly erroneous exceptions (BCSF taking more space than simple CSF) are explained by the approximation carried by formula (2) (assumption of equal values of C_{CSF} in both sides).

6.3 Approximate counts

In many applications, it is acceptable to tolerate a small absolute error in retrieved counts. Figure 5 shows that, in such case, it is possible to achieve a better memory consumption than simple CSFs even for small values of k. For medium values of k, neither too small nor too big, approximation can lead to the smallest compressed size, even when $\delta = 1$.

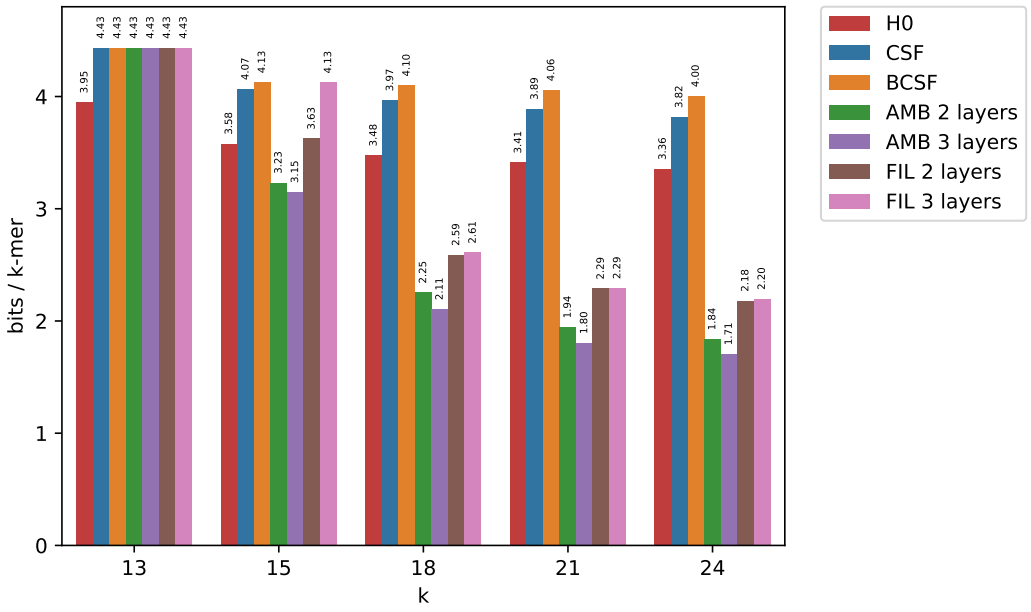


Figure 3 Compressed space usage for the high entropy df dataset.

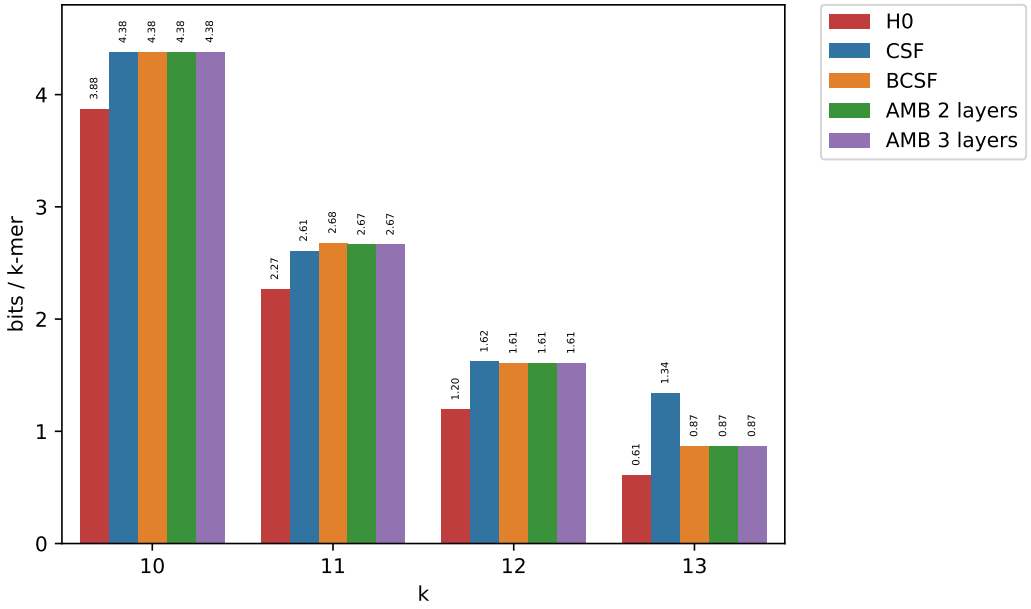


Figure 4 Space usage for the Sakai dataset with small k when using AMB (FIL is slightly worse and was omitted). Minimizer lengths vary between 1 and 5 indicating that the best option is to use a simple (B)CSF.

Unlike Figure 4, reported here for comparison, Figure 5 does not use the best minimizer lengths found for AMB with two layers. This is because we want to use δ to remove ambiguity from as many buckets as possible in each layer, by ignoring small collisions. For small k's this would not be possible with the best solutions found before, because minimizer lengths

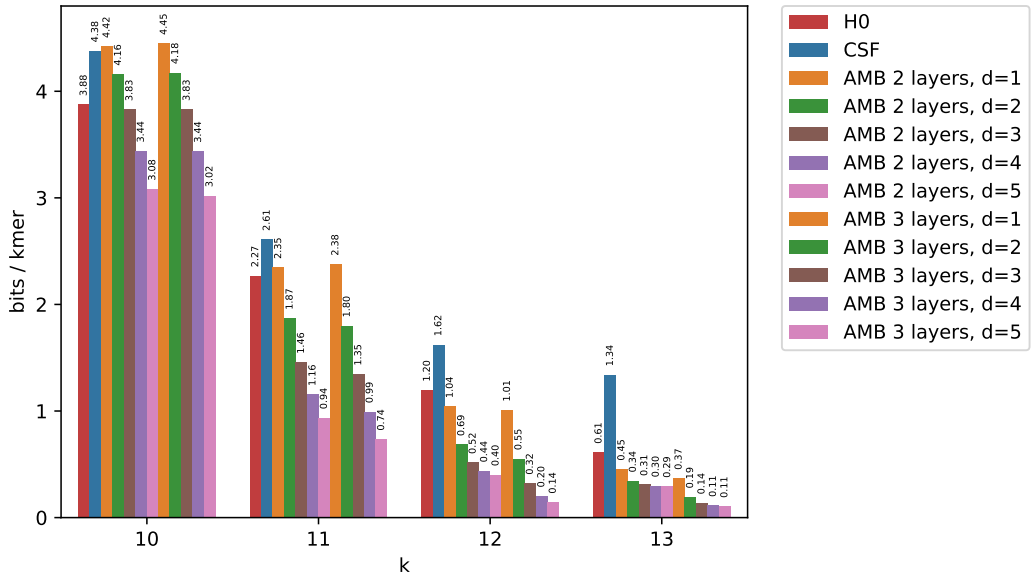


Figure 5 Space usage when using the approximated version of AMB. Entropy (red columns) and CSF (blue columns) are reported for comparison. Unlike Figure 4, AMB is able to break the empirical entropy lower bound when small errors are acceptable.

are too small to allow unambiguity even for $\delta > 0$. Therefore, in this case only, we just use contiguous minimizer lengths for each layer (e.g. if $k = 10$, layers will be $8, 9, 10$ for three-layer AMB).

Another interesting observation about the approximate case is that AMB with three layers is substantially better than AMB with two layers only for $k = 12$ and $k = 13$. For $k = 10$ and $k = 11$ both versions give almost the same results.

6.4 Query speed

Figure 6 shows query time averaged over all distinct k-mers, in ns/k-mer. Simple CSFs, not surprisingly, are the fastest method, with BCSF having a negligible effect on the average query speed. On the other hand, bucketing has a tangible effect on performance, with speed negatively affected by additional layers. For short k-mers, both FIL and AMB are slower than the simple CSF by a factor equal to their number of layers.

The situation is different for larger k's where AMB is only marginally slower than a bare-bones CSF. This is because most queries are solved without accessing all layers every time, thanks to unambiguous buckets. Two layered FIL, on the other hand, gives almost constant average query times across all test, since all queries have to access both of its layers to reconstruct the exact count value. We did not perorm tests for FIL with 3 layers because it will always be slower than the two layered version.

6.5 Choosing minimizer lengths

In all reported cases, good minimizer lengths for the first layer (m_0) follow the rule: $m_0 > m_s = (\log_4(|G|) + 2)$ with $|G|$, the size in base pair of the genome. Smaller m_0, are no longer capable of partitioning k-mers in a meaningful way. Furthermore, space tends to first monotonically decrease to a minimum for increasing minimizer lengths, to increase again

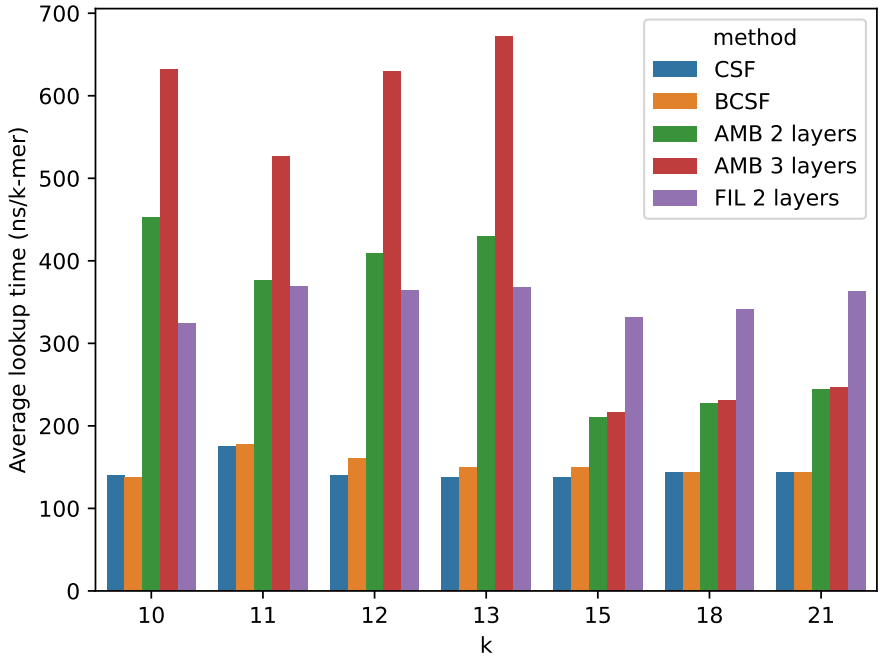


Figure 6 Average query time for AMB with 2 and 3 layers and FIL with 2 layers.

once the optimal value is passed. It is therefore possible to find the minimum by sequentially trying all possible minimizers greater than m_s and stop as soon as the compressed size starts to increase again.

Our results also show how multiple layers have a marginal effect on final compression sizes. In case of AMB, using three layers is always helpful, compared to the two-layer case. Best results are usually achieved for combinations including the best minimizer length obtained for the two-layer case.

On the other hand, FIL with three layers seems to be advantageous only for low entropy data, performing worse that its two-layer counterpart on the df dataset and for small k's.

7 Conclusions

In this work, we introduced three data structures to represent compressed k-mer count tables. Our BCSF algorithm combines Compressed Static Functions, as implemented in Sux4J software [11], with Bloom Filters. This allows for a much better compression for skewed distributions with empirical entropy smaller than 1. Note that to our knowledge, this is the first time that CSFs are used in bioinformatic applications. We also provide a method to dimension the Bloom filter in a BCSF in order to minimise the final space.

Our two other algorithms, FIL and AMB, pair BCSF with a bucketing procedure where count values are mapped into buckets according to minimizer values of respective k-mers. This locality-sensitive hashing scheme allows us to efficiently factor the space of counts, which leads to breaking the empirical entropy lower bound for large enough k's. FIL and AMB use slightly different strategies in decomposing the input table across minimizer layers.

Our last contribution is an extension of AMB to the approximate case, gaining more space at the expense of a small and user-definable absolute error on the retrieved counts.

We validated our algorithms on three different datasets, two fully assembled genomes (*E.Coli* and *C.Elegans*), and one document frequency example, for different k-mer lengths showing how BCSF, AMB and FIL behave in different situations. FIL and AMB have a clear advantage when minimizers are long enough to bucket k-mers in a meaningful way, for both skewed and high entropy data. When it is not possible to define a long-enough minimizer length, the advantage of using intermediate minimizer layers vanishes, and simple CSF and its BCSF provide a better solution.

At query time, CSF and BCSF are the fastest methods requiring about 100ns on average for a single query. For a fixed number of layers, AMB is faster than FIL in all situations when minimizers are useful. FIL becomes faster than AMB only for those cases when both algorithms achieve worse compression ratios than simple (B)CSF.

We consider this study to be the first step towards designing efficient representations for k-mer count tables occurring in data-intensive bioinformatics applications. One possible future direction is compression of RNA-Seq experiments where counts may translate expression levels of genes. Another example is metagenomics where different species may be present with different abundances which can be captured by k-mer counts. In such applications, efficient representation of k-mer counts can be particularly beneficial.

References

1 Djamal Belazzougui and Rossano Venturini. Compressed Static Functions with Applications. In *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 229–240. Society for Industrial and Applied Mathematics, 2013. doi:10.1137/1.9781611973105.17.

2 Robert S. Boyer and J. Strother Moore. MJRTY – A Fast Majority Vote Algorithm. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–117. Springer Netherlands, Dordrecht, 1991. doi:10.1007/978-94-011-3488-0_5.

3 A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, June 1997. doi:10.1109/SEQUEN.1997.666900.

4 Benny Chor, David Horn, Nick Goldman, Yaron Levy, and Tim Massingham. Genomic dna k-mer spectra: models and modalities. *Genome Biology*, 10(10):R108, October 2009. doi:10.1186/gb-2009-10-10-r108.

5 Yingnan Cong, Yao-ban Chan, and Mark A. Ragan. A novel alignment-free method for detection of lateral genetic transfer based on TF-IDF. *Scientific Reports*, 6(1):30308, 2016. doi:10.1038/srep30308.

6 M. Csűrös, L. Noé, and G. Kucherov. Reconsidering the significance of genomic word frequencies. *Trends in Genetics*, 23(11):543–546, November 2007. doi:10.1016/j.tig.2007.07.008.

7 Thomas Dencker, Chris-André Leimeister, Michael Gerth, Christoph Bleidorn, Sagi Snir, and Burkhard Morgenstern. Multi-SpaM: A Maximum-Likelihood Approach to Phylogeny Reconstruction Using Multiple Spaced-Word Matches and Quartet Trees. In Mathieu Blanchette and Aïda Ouangraoua, editors, *Comparative Genomics*, Lecture Notes in Computer Science, pages 227–241, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-030-00834-5_13.

8 Barış Ekim, Bonnie Berger, and Yaron Orenstein. A Randomized Parallel Algorithm for Efficiently Finding Near-Optimal Universal Hitting Sets. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 37–53, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45257-5_3.

9 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal Perfect Hashing via Recursive Splitting. *arXiv:1910.06416 [cs]*, November 2019. arXiv:1910.06416.

10 Huan Fan, Anthony R. Ives, Yann Surget-Groba, and Charles H. Cannon. An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data. *BMC Genomics*, 16(1):522, July 2015. doi:10.1186/s12864-015-1647-5.

11 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of ([compressed] static | minimal perfect hash) functions. *Information and Computation*, 273:104517, August 2020. doi:10.1016/j.ic.2020.104517.

12 Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

13 Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale. *bioRxiv*, page 2020.10.01.322164, November 2020. doi:10.1101/2020.10.01.322164.

14 Mikhail Karasikov, Harun Mustafa, Amir Joudaki, Sara Javadzadeh-no, Gunnar Rätsch, and André Kahles. Sparse Binary Relation Representations for Genome Graph Annotation. *Journal of Computational Biology*, 27(4):626–639, December 2019. doi:10.1089/cmb.2019.0324.

15 Parsoa Khorsand and Fereydoun Hormozdiari. Nebula: Ultra-efficient mapping-free structural variant genotyper. *bioRxiv*, page 566620, March 2019. doi:10.1101/566620.

16 Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, May 2017. doi:10.1093/bioinformatics/btx304.

17 Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, 27(5):722–736, 2017. doi:10.1101/gr.215087.116.

18 Téo Lemane, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: Efficient construction of Bloom filters for large sequencing data collections. *bioRxiv*, page 2021.02.16.429304, 2021. doi:10.1101/2021.02.16.429304.

19 Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, July 2016. doi:10.1093/bioinformatics/btw152.

20 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, May 2018. doi:10.1093/bioinformatics/bty191.

21 Antoine Limasset, Jean-François Flot, and Pierre Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *Bioinformatics*, 36(5):1374–1381, 2020. doi:10.1093/bioinformatics/btz102.

22 Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and Scalable Minimal Perfect Hashing for Massive Key Sets. In *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SEA.2017.25.

23 Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement_1):i177–i185, July 2020. doi:10.1093/bioinformatics/btaa487.

24 Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764, March 2011. doi:10.1093/bioinformatics/btr011.

25 Marmar Moussa and Ion I. Măndoiu. Single cell RNA-seq data clustering using TF-IDF based methods. *BMC Genomics*, 19(6):569, 2018. doi:10.1186/s12864-018-4922-4.

26 Harun Mustafa, André Kahles, Mikhail Karasikov, and Gunnar Rätsch. Metannot: A succinct data structure for compression of colors in dynamic de Bruijn graphs. *bioRxiv*, page 236711, March 2018. doi:10.1101/236711.

27 Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and Perfect Hashing Using Fingerprinting. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 138–149, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-07959-2_12.

28 Atif Rahman, Ingileif Hallgrímsdóttir, Michael Eisen, and Lior Pachter. Association mapping from sequencing reads using k-mers. *eLife*, 7:e32920, June 2018. doi:10.7554/eLife.32920.

29 Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, March 2013. doi:10.1093/bioinformatics/btt020.

30 Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004. doi:10.1093/bioinformatics/bth408.

31 K. Salikhov, G. Sacomoto, and G. Kucherov. Using cascading Bloom filters to improve the memory usage for de Brujin graphs. *BMC Algorithms for Molecular Biology*, 9(1):2, 2014. URL: http://www.almob.org/content/9/1/2.

32 Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 76–85, San Diego, California, 2003. Association for Computing Machinery. doi:10.1145/872757.872770.

33 Moustafa Shokrof, C. Titus Brown, and Tamer A. Mansour. MQF and buffered MQF: Quotient filters for efficient storage of k-mers with their counts and metadata. *bioRxiv*, page 2020.08.23.263061, August 2020. doi:10.1101/2020.08.23.263061.

34 Gregory E. Sims, Se-Ran Jun, Guohong A. Wu, and Sung-Hou Kim. Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions. *Proceedings of the National Academy of Sciences of the United States of America*, 106(8):2677–2682, February 2009. doi:10.1073/pnas.0813249106.

35 Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, March 2014. doi:10.1186/gb-2014-15-3-r46.

36 Huiguang Yi and Li Jin. Co-phylog: an assembly-free phylogenomic approach for closely related organisms. *Nucleic Acids Research*, 41(7):e75, 2013. doi:10.1093/nar/gkt003.

37 Ye Yu, Djamal Belazzougui, Chen Qian, and Qin Zhang. Memory-efficient and Ultra-fast Network Lookup and Forwarding using Othello Hashing. *arXiv:1608.05699 [cs]*, November 2017. arXiv:1608.05699.

38 Ye Yu, Jinpeng Liu, Xinan Liu, Yi Zhang, Eamonn Magner, Erik Lehnert, Chen Qian, and Jinze Liu. SeqOthello: querying RNA-seq experiments at scale. *Genome Biology*, 19(1):167, 2018. doi:10.1186/s13059-018-1535-9.

39 Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Lower Density Selection Schemes via Small Universal Hitting Sets with Short Remaining Path Length. In Russell Schwartz, editor, *Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 202–217, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45257-5_13.

A Multilayer FIL algorithm

Algorithm 4 FIL multi-layer construction algorithm.

Data: A mapping f of keys to (integer) values, a list L of minimizer lengths
Result: One BCSFs + Bloom filter for each layer
Sort L by increasing order;
$T = f$;
for m *in* L **do**
  Initialise an array A of buckets;
  $n = 0$;
  for (q, c) *in* T **do**
    $z = \mu_m(q)$ Insert c into $g(z)$;
    $n = n + 1$;
  end
  for b *in* A **do**
    Select representative r of bucket b by majority rule;
  end
  Compress A by using BCSF;
  Create output table O;
  $p_q = 0$;
  for (q, c) *in* T **do**
    if $g(\mu_m(q)) \neq c$ **then**
      Write q and $c - g(\mu_m(q))$ to O;
      $p_q = p_q + 1$;
    end
  end
  $\alpha = (n - p_q)/n$;
  $\epsilon = (1 - \alpha)/\alpha$;
  if $\epsilon < 1$ **then**
    Initialise an empty Bloom Filter of size $1.44 \log_2(1/\epsilon)$;
    Insert all elements of O into B;
    for (q, c) *in* T **do**
      if $g(\mu_m(q)) = c$ *and* $B(q)$ **then**
        Write q and $c - g(\mu_m(q))$ to O
      end
    end
  end
  $T = O$;
end

B Additional figures

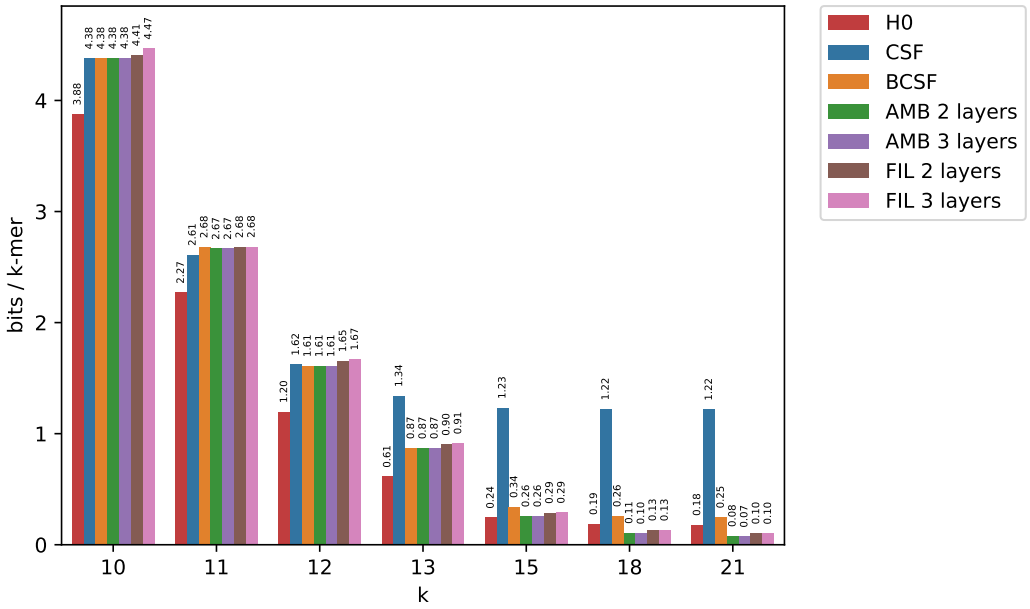


Figure 7 Space usage across all values of k, for the Sakai dataset.

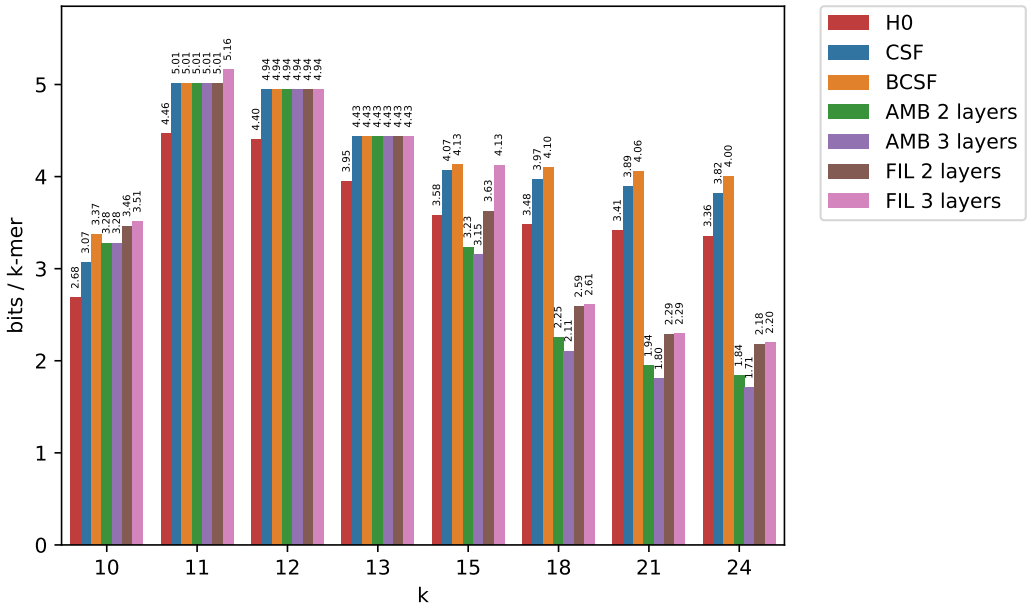


Figure 8 Space usage across all values of k, for the df dataset.

Parsimonious Clone Tree Reconciliation in Cancer

Palash Sashittal
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Simone Zaccaria
Computational Cancer Genomics Research Group, University College London Cancer Institute, London, UK
Cancer Research UK Lung Cancer Centre of Excellence, University College London Cancer Institute, London, UK

Mohammed El-Kebir[1]
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
Cancer Center at Illinois, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Abstract

Every tumor is composed of heterogeneous clones, each corresponding to a distinct subpopulation of cells that accumulated different types of somatic mutations, ranging from single-nucleotide variants (SNVs) to copy-number aberrations (CNAs). As the analysis of this intra-tumor heterogeneity has important clinical applications, several computational methods have been introduced to identify clones from DNA sequencing data. However, due to technological and methodological limitations, current analyses are restricted to identifying tumor clones only based on either SNVs or CNAs, preventing a comprehensive characterization of a tumor's clonal composition. To overcome these challenges, we formulate the identification of clones in terms of both SNVs and CNAs as a reconciliation problem while accounting for uncertainty in the input SNV and CNA proportions. We thus characterize the computational complexity of this problem and we introduce a mixed integer linear programming formulation to solve it exactly. On simulated data, we show that tumor clones can be identified reliably, especially when further taking into account the ancestral relationships that can be inferred from the input SNVs and CNAs. On 49 tumor samples from 10 prostate cancer patients, our reconciliation approach provides a higher resolution view of tumor evolution than previous studies.

2012 ACM Subject Classification Applied computing → Computational genomics

Keywords and phrases Intra-tumor heterogeneity, phylogenetics, mixed integer linear programming

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.9

Supplementary Material *Software (Source Code)*: `https://github.com/elkebir-group/paction` archived at `swh:1:dir:4a932ffa97a0c3be4118281e67ac2d86459de00f`

Funding *Simone Zaccaria*: Rosetrees Trust and CRUK Lung Cancer Centre of Excellence grant reference M917.
Mohammed El-Kebir: National Science Foundation award numbers CCF 1850502 and CCF 2046488.

Acknowledgements This work was a project in the course CS598MEB (Computational Cancer Genomics, Spring 2021) at UIUC. We thank the students in this course for their valuable feedback. We also thank Ron Zeira for providing the code to compute distances between copy number profiles.

[1] Corresponding author

1 Introduction

Cancer results from an evolutionary process where somatic mutations accumulate in the genomes of different cells. This process yields highly heterogeneous tumors composed of different *clones*, each corresponding to a distinct subpopulation of cells with the same complement of somatic mutations [27]. The resulting intra-tumor heterogeneity has been clearly linked to critically important cancer phenotypes, including cancer prognosis and the potential of developing resistance to cancer therapy [3, 24]. Therefore, important downstream applications rely on accurate reconstructions of a tumor's clonal architecture, which in turn requires the identification of the different clones, their proportions and their evolutionary history. However, the presence of different types of somatic mutations in the same clones renders these tasks particularly challenging. In particular, the following two types of somatic mutations are frequent in cancer [4, 39, 40]: (1) single nucleotide variants (SNVs), which are substitutions of individual DNA nucleotides, and (2) copy number alterations (CNAs), which are amplifications and deletions of large genomic regions.

Most cancer sequencing studies use bulk DNA sequencing technology, where one does not directly measure the co-occurrence of different mutations in the same clone because the generated DNA sequencing reads originate from unknown mixtures of millions of different cells in a bulk tumor sample. To identify distinct clones from such data, one thus needs to deconvolve the mixed sequencing data into the different clonal components [37]. Several computational methods have been introduced to perform this task. However, the majority of existing methods only focus on either SNVs [6, 29, 31, 35, 36] or CNAs [11, 25, 26, 28, 42–44], but rarely on both. Methods that attempt to identify clones in terms of both SNVs and CNAs do not not scale to the numbers of current cancer sequencing datasets (e.g., number of samples, mutations, clones, etc.) and often require heuristics to reduce the size of input instances [5, 9, 19]. As a result, current cancer evolutionary analyses [16, 18] do not apply such proposed methods but rather perform a *post hoc* analysis, manually assigning CNAs to a tree inferred from SNVs. Furthermore, we note that similar issues arise with some single-cell DNA sequencing technologies, since the different features of these technologies only allow the reliable measurement of either SNVs or CNAs [14]. For example, targeted MDA single-cell sequencing technologies are more suited for the idenification of SNVs whereas whole-exome/genome DOP-PCR single-cell technologies are more suited for the identification of CNAs, and both these technologies have been used in parallel on the same tumor sample [22].

In this study, we investigate whether tumor clonal compositions can be comprehensively reconstructed by an alternative simpler and automated approach. Leveraging the SNV and CNA clone proportions that can be independently and reliably inferred by existing methods, we introduce the Parsimonious Clone Reconciliation (PCR) and Parsimonious Clone Tree Reconciliation (PCTR) problems to infer clones in terms of both SNVs and CNAs, their proportions and, additionally for the PCTR problem, their evolutionary relationships (Figure 1). We prove that the proposed problems are NP-hard and we introduce PACTION (PArsimonious Clone Tree reconciliatION), an algorithm that solves these problems using two mixed integer linear programming formulations. Using simulations, we find that our approach reliably handles errors in input SNV and CNA proportions and scales to practical instance sizes. On 49 samples from prostate cancer patients [16], we find that our approach more comprehensively reconstructs tumor clonal architectures compared to the manual approach adopted in the previous analysis of the same data.

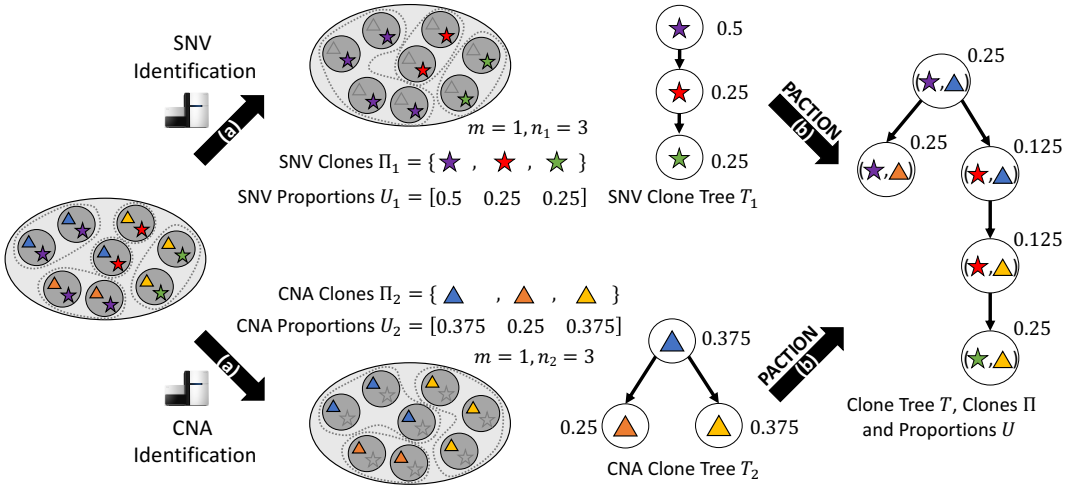


Figure 1 Overview. A tumor is composed of multiple subpopulations of cells, or clones, with distinct somatic mutations, which can be measured using DNA sequencing. (a) Due to limitations in inference algorithms and/or sequencing technologies, we are limited to characterizing tumor clones in terms of either single-nucleotide variants (SNVs, stars) or copy-number aberrations (CNAs, triangles). That is, we infer clones Π_1, proportions U_1 and a clone tree T_1 for the SNVs. Similarly, we infer clones Π_2, proportions U_2 and a clone tree T_2 for the CNAs. (b) PACTION solves the Parsimonious Clone Tree Reconciliation problem of inferring clones $\Pi \subseteq \Pi_1 \times \Pi_2$, a clone tree T and proportions U that characterize the clones of the tumor in terms of both SNVs and CNAs.

2 Problem Statements

We introduce two reconciliation problem formulations to reconstruct tumor clonal composition from inferred SNV and CNA clone proportions[2]. The first problem aims at inferring tumor clones and related proportions with both SNVs and CNAs given the clone proportions of SNVs and CNAs independently (Section 2.1). The second problem additionally considers phylogenetic trees describing the evolution of tumor clones with either different SNVs or CNAs (Section 2.2).

2.1 Parsimonious Clone Reconciliation

Suppose a tumor is composed of a set Π of $n = |\Pi|$ clones, which are characterised by unique complements of two different features (e.g., SNVs and CNAs). These clones occur in m samples at varying proportions, defined as follows.

▶ **Definition 1.** *An $m \times n$ matrix $U = [u_{p,\ell}]$ is a* proportion matrix for n clones Π *provided (i) $u_{p,\ell} \geq 0$ for all samples $p \in [m]$ and clones $\ell \in [n]$, and (ii) $\sum_{\ell=1}^{n} u_{p,\ell} = 1$ for all samples $p \in [m]$.*

Due to limitations in inference algorithms and/or sequencing technologies, we only infer clones and their proportions for one feature in isolation. These two features lead to two distinct partitions of all tumor cells: a set $\Pi_1 = [n_1]$ of clones induced by the first feature (e.g.,

[2] While reconciliation is used in species phylogenetics, particularly in the context of gene-tree species-tree reconciliation, here we will use this term to indicate the process of obtaining a comprehensive evolutionary tree of tumor clones given input trees that each focus on a distinct genomic feature.

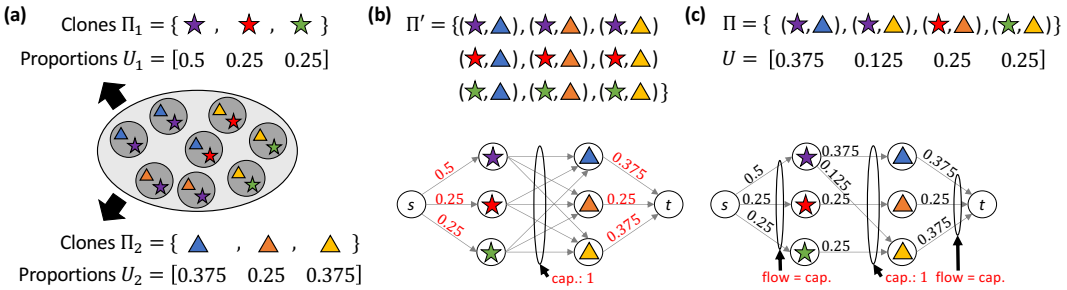


Figure 2 The Parsimonious Clone Reconciliation (PCR) problem. (a) Given clones Π_1 and Π_2 and corresponding proportions U_1 and U_2, we seek clones $\Pi \subseteq \Pi_1 \times \Pi_2$ and corresponding proportions U consistent with U_1 and U_2. (b) There always exists a consistent proportion matrix U' for the trivial solution $\Pi' = \Pi_1 \times \Pi_2$, which can be identified by solving a maximum flow problem. (c) We seek the solution Π with minimum number $|\Pi|$ of clones. Here, $|\Pi| = 4$, which is smaller than ground truth (see panel (a)). The corresponding matrix U follows from solving the illustrated maximum flow problem. However, incorporating tree constraints, as in the PCTR problem, will lead to ground truth (Figure 1).

SNVs) and a set $\Pi_2 = [n_2]$ of clones induced by the second feature (e.g., CNAs). We refer to the original clones as Π-clones and the clones induced by the first and the second features as Π_1-clones and Π_2-clones, respectively. The proportions of the Π_1-clones and Π_2-clones are given by the $m \times n_1$ proportion matrix $U_1 = [u^{(1)}_{p,i}]$ and the $m \times n_2$ proportions matrix $U_2 = [u^{(2)}_{p,j}]$, respectively. How are the proportions U_1 for Π_1-clones and the proportions U_2 for Π_2-clones related to the proportions U of the Π-clones?

To answer this question, recall that Π is a partition of all tumor cells induced by the combination of both the two features, whereas Π_1 and Π_2 are partitions induced by each feature in isolation (Figure 2a). As such, we have that the partition Π is a refinement of partitions Π_1 and Π_2. Thus, each Π-clone ℓ corresponds to a unique Π_1-clone i and a unique Π_2-clone j. In other words, we may view the set Π as a binary relation of sets Π_1 and Π_2 of clones composed of pairs $\ell = (i, j)$ of clones, i.e., $\Pi \subseteq \Pi_1 \times \Pi_2$. This relation is captured by the projection functions $\pi_1 : \Pi \to \Pi_1$ and $\pi_2 : \Pi \to \Pi_2$ such that $\pi_1((i, j)) = i$ and $\pi_2((i, j)) = j$ for all $(i, j) \in \Pi$. We relate the proportion matrix U for clones Π to the proportion matrix U_1 for clones Π_1 and the proportion matrix U_2 for clones Π_2 as follows.

▶ **Definition 2.** *Given projection functions $\pi_1 : \Pi \to \Pi_1$ and $\pi_2 : \Pi \to \Pi_2$ induced by the set $\Pi \subseteq \Pi_1 \times \Pi_2$ of clones, the proportion matrix $U = [u_{p,\ell}]$ for clones Π is* consistent *with a proportion matrix $U_1 = [u^{(1)}_{p,i}]$ for clones $\Pi_1 = [n_1]$ and proportion matrix $U_2 = [u^{(2)}_{p,j}]$ for clones $\Pi_2 = [n_2]$ provided (i) $u^{(1)}_{p,i} = \sum_{\ell:\pi_1(\ell)=i} u_{p,\ell}$ for all samples $p \in [m]$ and clones $i \in [n_1]$, and (ii) $u^{(2)}_{p,j} = \sum_{\ell:\pi_2(\ell)=j} u_{p,\ell}$ for all samples $p \in [m]$ and clones $j \in [n_2]$.*

The above definition formalizes the intuition that clones Π of the tumor are a refinement of the input clones Π_1 and Π_2, and therefore their proportions U must be consistent with the input proportions U_1 and U_2. Our goal is to recover the set $\Pi \subseteq \Pi_1 \times \Pi_2$ of clones and their proportions U from the proportion matrices U_1 and U_2 for clones Π_1 and Π_2, respectively. While there always exist trivial solutions given by the full set $\Pi' = \Pi_1 \times \Pi_2$ of $n = n_1 \cdot n_2$ clones (Figure 2b), we seek a solution Π with the smallest number n of clones under the principle of parsimony (Figure 2c).

▶ **Problem 3** (Parsimonious Clone Reconciliation (PCR))**.** *Given proportions U_1 for clones $\Pi_1 = [n_1]$ and proportions U_2 for clones $\Pi_2 = [n_2]$, find (i) the smallest set $\Pi \subseteq \Pi_1 \times \Pi_2$ of clones and (ii) proportions U for Π such that U is consistent with U_1 and U_2.*

2.2 Parsimonious Clone Tree Reconciliation

In practice, proportions U_1 and U_2 are not measured exactly but are affected by potential measurement errors. As such, accurate recovery of the original clones Π and their proportions U requires correcting U_1 and U_2. To accomplish this, we require additional information and constraints. In this work, we propose to use the evolutionary relationships among the clones Π_1 and Π_2 that can be inferred by existing methods in the form of clone trees [6–8, 23, 29, 33]. Specifically, a rooted tree T is a *clone tree* for clones Π provided the vertex set $V(T)$ equals Π. Moreover, the root vertex $r(T)$ of a clone tree T corresponds to the normal clone while each edge $(u, v) \in E(T)$ represents a mutation event that altered one of the features of clone u and led to the formation of the clone v.

Similarly to the PCR problem, we are given two clone trees, one for each feature in isolation. In the specific example of two features (e.g., SNVs and CNAs), let clone tree T_1 describe the evolution of clones Π_1 (e.g., SNVs) and clone tree T_2 describe the evolution of clones Π_2 (e.g., CNAs). These trees are inferred using standard algorithms in the field [6, 11, 25, 26, 28, 29, 31, 35, 36, 42–44]. Since all clones share a common evolutionary history the original clone tree T is a *refinement* [31, 41] of the clone trees T_1 and T_2, which is defined as follows.

▶ **Definition 4.** *Clone tree T for clones Π is a* refinement *of clone trees T_1 for clones Π_1 and clone tree T_2 for clones Π_2 provided*

- **(i)** *for each edge $(i, i') \in E(T_1)$ there exists exactly one $j \in \Pi_2$ such that $((i, j), (i', j)) \in E(T)$,*
- **(ii)** *for each edge $(j, j') \in E(T_2)$ there exists exactly one $i \in \Pi_1$ such that $((i, j), (i, j')) \in E(T)$,*
- **(iii)** *for each $((i, j), (i', j')) \in E(T)$, it holds that $(i, i') \in E(T_1)$ and $j = j'$, or $(j, j') \in E(T_2)$ and $i = i'$.*

Intuitively, the above definition states that when collapsing vertices of T corresponding to identical Π_1-clones one obtains T_1, and, similarly, T_2 is obtained by collapsing vertices of T corresponding to identical Π_2-clones.

Under a principle of parsimony and given clone trees T_1, T_2 with related proportions U_1, U_2, our goal is to find a set $\Pi \subseteq \Pi_1 \times \Pi_2$ of clones, a clone proportion matrix U, and a T_1, T_2-refined clone tree T that require the smallest correction in U_1 and U_2. This motivates the following problem statement.

▶ **Problem 5** (Parsimonious Clone Tree Reconciliation (PCTR))**.** *Given proportions U_1 and tree T_1 for clones $\Pi_1 = [n_1]$ and proportions U_2 and tree T_2 for clones $\Pi_2 = [n_2]$, find (i) the set Π of clones, (ii) clone tree T and (iii) proportions U for Π such that the clone tree T is a refinement of T_1 and T_2 and minimizes the total error $J(U, U_1, U_2)$ such that*

$$J(U, U_1, U_2) = \sum_{p=1}^{m} \sum_{i=1}^{n_1} |u_{p,i}^{(1)} - \sum_{\ell : \pi_1(\ell) = i} u_{p,\ell}| + \sum_{p=1}^{m} \sum_{j=1}^{n_2} |u_{p,j}^{(2)} - \sum_{\ell : \pi_2(\ell) = i} u_{p,\ell}|.$$

Note that $J(U, U_1, U_2) = 0$ if and only if U is consistent with U_1 and U_2. The clone trees T, T_1 and T_2 do not appear in the objective function $J(U, U_1, U_2)$ and only provides constraints to the optimization problem. Due to these constraints, unlike the previous PCR problem, PCTR does not always admit a trivial solution with $J(U, U_1, U_2) = 0$ (as we further discuss in Section 3.2).

3 Combinatorial Characterization and Computational Complexity

We investigate the combinatorial structure and computational complexity of the two proposed PCR and PCTR problems in the following two sections, respectively.

3.1 Parsimonious Clone Reconciliation

We characterize the combinatorial structure of feasible and optimal solutions (Π, U) for the PCR problem. We first observe that the PCR problem always has a trivial solution. Specifically, given a set Π_1 of $n_1 = |\Pi_1|$ clones and a set Π_2 of $n_2 = |\Pi_2|$ clones and corresponding proportions $U_1 \in [0,1]^{m \times n_1}$ and $U_2 \in [0,1]^{m \times n_2}$, a trivial feasible solution is composed of $n = n_1 n_2$ clones $\Pi = \Pi_1 \times \Pi_2$, which may have many possible corresponding proportions U (Figure 2b). For example, proportions $U = [u_{p,(i,j)}]$ can be computed greedily by considering the n clones in any arbitrary order, and assigning each clone $(i,j) \in \Pi$ a proportion of $u_{p,(i,j)} = \min(u^{(1)}_{p,i}, u^{(2)}_{p,j})$ followed by subsequently updating $u^{(1)}_{p,i} := u^{(1)}_{p,i} - u_{p,(i,j)}$ and $u^{(2)}_{p,j} := u^{(2)}_{p,j} - u_{p,(i,j)}$ for each sample $p \in [m]$. Thus, $n = n_1 n_2$ is an upper bound on the number of clones needed. Can we similarly identify a lower bound on n?

To answer this question, let the *support* $S(U)$ of an $m \times n$ proportion matrix U be defined as the number of non-zero entries in the vector $U\mathbf{1}_m$ where $\mathbf{1}_m$ is a $m \times 1$ vector with all entries equal to one. That is, the support $S(U)$ of a proportion matrix U of clones Π signifies the number of clones with non-zero proportion in at least one of the samples $p \in [m]$. Any such clone must be part of at least one clone $\ell \in \Pi$ in the solution to the PCR problem to ensure consistency of the proportion matrices. This leads to the following observation.

▶ **Observation 6.** *Given an instance (Π_1, U_1, Π_2, U_2) of the PCR problem with solution Π we have $n \geq \max(S(U_1), S(U_2))$ where $n = |\Pi|$.*

Given any set $\Pi \subseteq \Pi_1 \times \Pi_2$ of clones, deciding whether there exists a proportion matrix U that is consistent with given proportion matrix U_1 for clones Π_1 and U_2 for clones Π_2, and constructing such a matrix is equivalent to solving a maximum flow problem, which takes polynomial time [1]. Figure 2 illustrates the construction such that there exists a consistent proportion matrix if and only the value of the flow is 1. Note that for $m > 1$ samples, we need to solve a multi-commodity rather than a single-commodity flow problem. However, the PCR problem, where we simultaneously seek Π and U, is NP-hard and the hardness comes from having to identify the smallest set Π of clones.

▶ **Theorem 7.** *The PCR problem is NP-hard even for number $m = 1$ of samples.*

This follows by reduction from the 3-PARTITION problem, a known NP-complete problem [12, 13] stated as follows.

▶ **Problem 8** (3-PARTITION). *Given an integer $B \in \mathbb{N}^{>0}$, a multiset $A = \{a_1, \cdots, a_{3q}\}$ of $3q$ positive integers such that $a_i \in (B/4, B/2)$ for all $i \in [3q]$, and $\sum_{i=1}^{3q} a_i = Bq$, does there exist a partition of A into q disjoint subsets such that the sum of the integers in each subset equals B?*

Note that since each a_i occurs within the open interval $(B/4, B/2)$ and the elements in each subset of the desired partition sum to B, it holds that each subset must be composed of exactly three elements from the multiset A – hence the name of the problem.

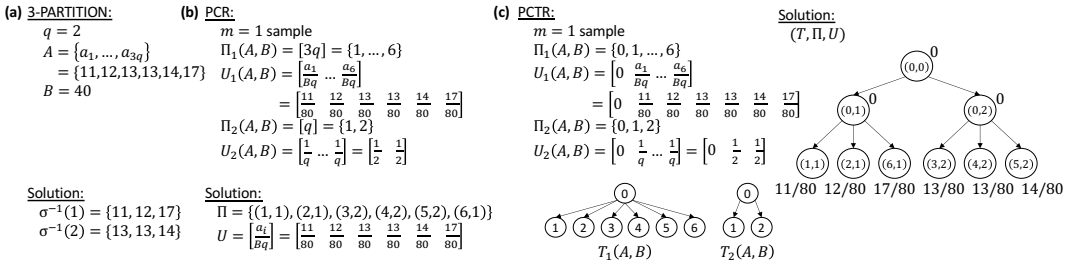


Figure 3 Reduction from 3-PARTITION. (a) Example instance of 3-PARTITION with a multiset A of 6 elements and target sum $B = 40$. (b) Corresponding PCR instance (Π_1, U_1, Π_2, U_2) and solution (Π, U). (c) Corresponding PCTR instance $(T_1, \Pi_1, U_1, T_2, \Pi_2, U_2)$ and solution (T, Π, U).

We represent the solution to an instance (A, B) of the 3-PARTITION problem as a function $\sigma : [3q] \to [q]$, which encodes the division of the elements of $A = \{a_1, \ldots, a_{3q}\}$ into q disjoint subsets. The inverse of this function specifies the subset corresponding to each $j \in [q]$ as $\sigma^{-1}(j) = \{i \in [3q] : \sigma(i) = j\}$. Note that any solution $\sigma : [3q] \to [q]$ of the 3-PARTITION problem satisfies the following constraint.

$$\sum_{i \in \sigma^{-1}(j)} a_i = B, \quad \forall j \in [q]. \tag{1}$$

Figure 3a provides an example 3-PARTITION instance and solution.

Given a 3-PARTITION problem instance (A, B), we construct an instance of the PCR problem with number $m = 1$ of samples as follows. The set $\Pi_1(A, B)$ of clones is given by the set $[3q]$. The corresponding proportions are given by the $1 \times 3q$ proportion matrix $U_1(A, B) = [u^{(1)}_{1,i}]$ where $u^{(1)}_{1,i} = a_i/Bq$ for all $i \in [3q]$. Clearly, $U_1(A, B) = [u^{(1)}_{1,i}]$ is a proportion matrix for $\Pi_1(A, B)$ as, by construction, we have that $\sum_{i=1}^{3q} u^{(1)}_{1,i} = 1$ and $u^{(1)}_{1,i} \geq 0$ for all $i \in [3q]$. The second set $\Pi_2(A, B)$ of clones is given by $[q]$. The corresponding proportions are given by the $1 \times q$ proportion matrix $U_2(A, B) = [u^{(2)}_{1,j}]$ where $u^{(2)}_{1,j} = 1/q$ for all $j \in [q]$. It is easy to verify that $U_2(A, B)$ is a proportion matrix for $\Pi_2(A, B)$. Clearly, this construction takes polynomial time. Figure 3b shows an example. Hardness follows from the following lemma whose proof is omitted due to space constraints.

▶ **Lemma 9.** *Given proportions $U_1(A, B)$ for clones $\Pi_1(A, B) = [3q]$ and proportions $U_2(A, B)$ for clones $\Pi_2(A, B) = [q]$, there exists a set Π of clones of size $n = |\Pi| \leq 3q$ with proportions U that are consistent with $U_1(A, B)$ and $U_2(A, B)$ if and only if there exists a solution to the 3-PARTITION instance (A, B).*

3.2 Parsimonious Clone Tree Reconciliation

We now characterize the combinatorial structure of feasible and optimal solutions (Π, U, T) for the PCTR problem. Let T_1 be the first input clone tree for the input set Π_1 of $n_1 = |\Pi_1|$ clones. Similarly, let T_2 be the second input clone tree for the input set Π_2 of $n_2 = |\Pi_2|$ clones. Let T be a solution clone tree that is a refinement of both T_1 and T_2. First, we observe that the clones that label the root vertices $r(T_1)$ and $r(T_2)$ of the two input trees together label the root vertex $r(T)$ of the output tree T, i.e., $r(T) = (r(T_1), r(T_2))$.

▶ **Observation 10.** *If clones Π, clone tree T and proportion matrix U form a solution to the PCTR instance $(\Pi_1, T_1, U_1, \Pi_2, T_2, U_2)$, then $(r(T_1), r(T_2)) \in \Pi$ and $r(T) = (r(T_1), r(T_2))$.*

Next, from Definition 4 it follows that in the output clone tree T it must hold that along each edge there is either a change in corresponding Π_1-clones or Π_2-clones but not both.

▶ **Observation 11.** *For each* $(i,j) \in V(T) \setminus \{r(T)\}$ *it holds that either* $((i',j),(i,j)) \in E(T)$ *or* $((i,j'),(i,j)) \in E(T)$ *where* $(i',i) \in E(T_1)$ *and* $(j',j) \in E(T_2)$.

Combining these observations, we get that the number of vertices/clones in T equals $n = n_1 + n_2 - 1$.

▶ **Observation 12.** *The number of clones* $V(T)$ *equals* $n = n_1 + n_2 - 1$.

We note that T is a multi-state perfect phylogeny with two characters, i.e. each character state labels at most one edge of T, whose two sets of states correspond to Π_1 and Π_2. Moreover, T_1 and T_2 impose an ordering of two sets of states to which T must adhere – i.e., the two characters are cladistic [10]. The problem of deciding whether there exists an error-free solution of PCTR with $J(U,U_1,U_2) = 0$ is equivalent to a special case of the Cladistic Multi-state Perfect Phylogeny Deconvolution problem [9]. Details and precise definitions of these concepts are omitted due to space constraints. Although the tree constraints alter the solution space of PCTR problem compared to the PCR problem (see Figure 1 and Figure 2c), PCTR remains NP-hard, as we will show in the following.

▶ **Theorem 13.** *The PCTR problem is* NP*-hard even for number* $m = 1$ *of samples.*

For a given instance (A,B) of the 3-PARTITION problem, we construct an instance of the PCTR problem as follows. The first set $\Pi_1(A,B)$ of clones equals $\{0\} \cup [3q]$ with corresponding $1 \times (3q+1)$ proportion matrix $U_1(A,B) = [u^{(1)}_{1,i}]$ where $u^{(1)}_{1,i} = a_i/(Bq)$ for all $i \in [3q]$, and $u^{(1)}_{1,0} = 0$. The second set $\Pi_2(A,B)$ of clones equals $\{0\} \cup [q]$ with corresponding $1 \times (q+1)$ proportion matrix $U_2(A,B) = [u^{(2)}_{1,j}]$ where $u^{(2)}_{1,j} = 1/q$ for all $j \in [q]$, and $u^{(2)}_{1,0} = 0$. The clone tree $T_1(A,B)$ is a star phylogeny rooted at Π_1-clone $i = 0$ with outgoing edges to each of the remaining Π_1-clones. Similarly, clone tree $T_2(A,B)$ is also a *star* phylogeny rooted at Π_2-clone $j = 0$ with outgoing edges to each of the remaining Π_2-clones. It is easy to verify that $U_1(A,B)$ and $U_2(A,B)$ are proportion matrices for $\Pi_1(A,B)$ and $\Pi_2(A,B)$, respectively. Clearly, this construction takes polynomial time. Figure 3c shows an example. The hardness follows from the following lemma whose proof is omitted due to space constraints.

▶ **Lemma 14.** *Given proportions* $U_1(A,B)$ *and clone tree* T_1 *for clones* $\Pi_1(A,B) = \{0\} \cup [3q]$ *and proportions* $U_2(A,B)$ *and clone tree* T_2 *for clones* $\Pi_2(A,B) = \{0\} \cup [q]$*, there exists a set* Π *of clones of size* $n = |\Pi| = 4q+1$*, clone tree* T *and proportion matrix* U *such that* T *is a refinement of* T_1 *and* T_2 *and* $J(U,U_1,U_2) = 0$ *if and only if there exists a solution of the 3-PARTITION instance* (A,B).

4 Methods

We introduce two mixed integer linear programming (MILP) formulations to solve the PCR (Section 4.1) and the PCTR problems (Section 4.2). We implement these two formulations within the algorithm PACTION (PArsimonious Clone Tree reconciliatION), which uses the MILP-solver Gurobi version 9.1. PACTION is available at `https://github.com/elkebir-group/paction`.

4.1 Parsimonious Clone Reconciliation

To solve the PCR problem, we introduce an MILP formulation composed of $\mathcal{O}(n_1 n_2 m)$ variables (including $O(n_1 n_2)$ binary variables) and $\mathcal{O}(n_1 n_2 m)$ constraints. We introduce binary variables $x_{i,j} \in \{0,1\}$ for each Π_1-clone $i \in [n_1]$ and Π_2-clone $j \in [n_2]$ that indicate if clone (i,j) belongs to Π. As such, the corresponding proportion of clone (i,j) in sample $p \in [m]$ is denoted by the continuous variable $u_{p,i,j} \in [0,1]$. In the following we define the constraints on these variables by first describing the constraints for consistency and next those for encoding the objective function.

Consistency constraints. This first set of constraints ensure that proportion matrix U is consistent with proportion matrices U_1 and U_2. We begin by forcing $u_{p,i,j}$ to 0 if (i,j) is not a clone in the solution Π.

$$u_{p,i,j} \leq x_{i,j} \qquad \forall p \in [m], i \in [n_1], j \in [n_2].$$

These above constraints allow us to model consistency of the solution U with input proportions $U_1 = [u^{(1)}_{p,i}]$ and $U_2 = [u^{(2)}_{p,j}]$ as follows.

$$\sum_{j=1}^{n_2} u_{p,i,j} = u^{(1)}_{p,i} \qquad \forall p \in [m], i \in [n_1],$$

$$\sum_{i=1}^{n_1} u_{p,i,j} = u^{(2)}_{p,j} \qquad \forall p \in [m], j \in [n_2].$$

Note that these two sets of constraints imply that $\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} u_{p,i,j} = 1$ for all $p \in [m]$.

Objective function. We minimize the total number of clones in the set Π by minimizing the following objective function.

$$\min \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} x_{i,j}.$$

4.2 Parsimonious Clone Tree Reconciliation

To solve the PCTR problem, we introduce an MILP formulation composed of $\mathcal{O}(n_1 n_2 m)$ variables (including $O(n_1 n_2)$ binary variables) and $\mathcal{O}(n_1 n_2 m)$ constraints. Similarly to the PCR MILP, we introduce binary variables $x_{i,j} \in \{0,1\}$ for $i \in [n_1]$ and $j \in [n_2]$ that indicate if clone (i,j) belongs to Π. As such, the corresponding proportion of clone (i,j) in sample $p \in [m]$ is denoted by the continuous variable $u_{p,i,j} \in [0,1]$. We introduce constraints to model the error $J(U, U_1, U_2)$ used in the objective function, as well constraints to enforce that U is a proportion matrix, and finally constraints to enforce that T is a refinement of T_1 and T_2.

Correction constraints. Unlike the PCR problem, the proportion matrix U need not be consistent with proportion matrices U_1 and U_2. We introduce continuous variables $c^{(1)}_{p,i} \in [0,1]$ for $p \in [m], i \in [n_1]$ and $c^{(2)}_{p,j} \in [0,1]$ for $p \in [m], j \in [n_2]$ to model the entry-wise absolute differences, i.e., $c^{(1)}_{p,i} = |\sum_{j=1}^{n_2} u_{p,i,j} - u^{(1)}_{p,i}|$ and $c^{(2)}_{p,j} = |\sum_{j=1}^{n_2} u_{p,i,j} - u^{(2)}_{p,j}|$. We do so with the following constraints.

$$c^{(1)}_{p,i} \geq \sum_{j=1}^{n_2} u_{p,i,j} - u^{(1)}_{p,i} \qquad \forall p \in [m], i \in [n_1],$$

$$c^{(1)}_{p,i} \geq u^{(1)}_{p,i} - \sum_{j=1}^{n_2} u_{p,i,j} \qquad \forall p \in [m], i \in [n_1],$$

$$c^{(2)}_{p,j} \geq \sum_{i=1}^{n_1} u_{p,i,j} - u^{(2)}_{p,j} \qquad \forall p \in [m], j \in [n_2],$$

$$c^{(2)}_{p,j} \geq u^{(2)}_{p,j} - \sum_{i=1}^{n_1} u_{p,i,j} \qquad \forall p \in [m], j \in [n_2].$$

Proportion matrix constraints. To model that our output matrix U is a proportion matrix, we begin by ensuring that $u_{p,i,j} = 0$ with $x_{i,j} = 0$, i.e., the proportion of clone (i,j) is zero when it is not part of the solution Π with the following constraints.

$$u_{p.i,j} \leq x_{i,j} \qquad \forall p \in [m], i \in [n_1], j \in [n_2].$$

Next, we ensure that matrix U is a valid proportion matrix by enforcing that the proportions of the clones in each sample sum to 1.

$$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} u_{p,i,j} = 1 \qquad \forall p \in [m].$$

Refinement constraints. We introduce constraints that ensure that the clone tree T is a refinement of the clone trees T_1 and T_2. Following condition (iii) in Definition 4, we require that for each clone $(i,j) \neq (r(T_1), r(T_2))$ there only two possible parents, i.e., either (i',j) or (i,j') where $(i',i) \in E(T_1)$ and $(j',j) \in E(T_2)$. We model the first case with continuous variables $z^{(1)}_{(i,i'),j} \in [0,1]$ and the second case with continuous variables $z^{(2)}_{i,(j,j')}$. More specifically, we model the products $z^{(1)}_{(i,i'),j} = x_{i,j}x_{i',j}$ and $z^{(2)}_{i,(j,j')} = x_{i,j}x_{i,j'}$ with the following constraints.

$$z^{(1)}_{(i,i'),j} \leq x_{i,j} \qquad \forall (i,i') \in E(T_1), j \in [n_2],$$

$$z^{(1)}_{(i,i'),j} \leq x_{i',j} \qquad \forall (i,i') \in E(T_1), j \in [n_2],$$

$$z^{(1)}_{(i,i'),j} \geq x_{i,j} + x_{i',j} - 1 \qquad \forall (i,i') \in E(T_1), j \in [n_2].$$

$$z^{(2)}_{i,(j,j')} \leq x_{i,j} \qquad \forall i \in [n_1], (j,j') \in E(T_2),$$

$$z^{(2)}_{i,(j,j')} \leq x_{i,j'} \qquad \forall i \in [n_1], (j,j') \in E(T_2),$$

$$z^{(2)}_{i,(j,j')} \geq x_{i,j} + x_{i,j'} - 1 \qquad \forall i \in [n_1], (j,j') \in E(T_2).$$

We now enforce conditions (i) and (ii) in Definition 4 as follows.

$$\sum_{j=1}^{n_2} z^{(1)}_{(i,i'),j} = 1 \qquad \forall (i,i') \in E(T_1),$$

$$\sum_{i=1}^{n_1} z^{(2)}_{i,(j,j')} = 1 \qquad \forall (j,j') \in E(T_2).$$

Objective function. Our goal is to minimize the difference between projections of proportion matrix U with U_1 and U_2. To that end, we minimize the following objective function

$$\min \sum_{p=1}^{m} \sum_{i=1}^{n_1} c_{p,i}^{(1)} + \sum_{p=1}^{m} \sum_{j=1}^{n_2} c_{p,j}^{(2)}.$$

We provide the full MILP for reference in Appendix A.

5 Results

5.1 Simulations

We perform simulations to investigate the performance of PACTION when solving the PCR and PCTR problems under different simulation regimes.

Setup. Given numbers n_1, n_2 of clones, number m of samples and noise parameter $h \in [0, 1]$, we use a three-step procedure to simulate a set Π of $n = n_1 + n_2$ clones whose SNV and CNA evolution is described by a clone tree T and with clone proportions U on m samples. From T and U, we obtain input trees T_1 and T_2 as well as input proportion matrices U_1 and U_2 subject to additional noise h. We detail the three steps in the following.

First, we use an approach based on growing random networks [21] to simulate T: starting from the root vertex (representing the normal clone $(1, 1)$) T's topology is built by iteratively adding descendant vertices, choosing each parent uniformly at random. Specifically, we label each edge with a single event from either the first set $\{2, \ldots, n_1\}$ or second set $\{2, \ldots, n_2\}$ of features. Thus, the overall clones Π are obtained by labeling all vertices with a depth-first traversal. Second, we obtain the clone trees T_1 and T_2 by collapsing vertices of T corresponding to identical Π_1-clones and collapsing vertices of T corresponding to identical Π_2-clones, respectively. Third, the proportions U of the Π-clones in each sample are simulated by using a Dirichlet distribution with all concentration parameters equal to 1, similarly to previous methods [6, 23]. Proportions U_1 and U_2 are thus obtained following the consistency condition (Definition 2). Furthermore, we introduce noise in these two proportion matrices by mixing in a second draw from the same Dirichlet distribution using the parameter $h \in [0, 1]$ – a value of $h = 0$ indicates the absence of noise. Details are in Appendix B.

We ran PACTION in both PCR and PCTR mode on 360 simulated instances that we obtained by generating 10 instances for each combination of varying parameters. Matching numbers observed in recent cancer genomics studies [16, 18, 44], we varied the numbers $n_1 \in \{3, 5, 8\}$ and $n_2 \in \{3, 5, 8\}$ of clones, the number $m \in \{1, 2, 5\}$ of samples and noise level $h \in \{0, 0.05, 0.1, 0.15\}$. Note that both proportions U_1, U_2 and the simulated trees T_1, T_2 are taken in input in PCTR mode, while only proportions U_1, U_2 are considered in PCR mode.

Results. We measure the performance of PACTION based on recall, which is the fraction of ground truth clones that are predicted by our method, i.e., the *clone recall* equals $|\Pi \cap \Pi^*|/|\Pi^*|$ where Π is the set of clones inferred by PACTION and Π^* are the ground truth clones. As expected, PACTION in PCTR mode leverages additional information from the clone trees T_1 and T_2 and thus resulted in higher recall compared to PCR mode (Figure 4a). Interestingly, recall increased with increasing number m of samples, as each additional samples provides additional constraints regarding consistency of the output clone proportions. Breaking down the clone recall by noise level h, we found that performance decreased with increasing noise levels in both PCR mode (Figure 4b) as well as PCTR mode (Figure 4c). However, we

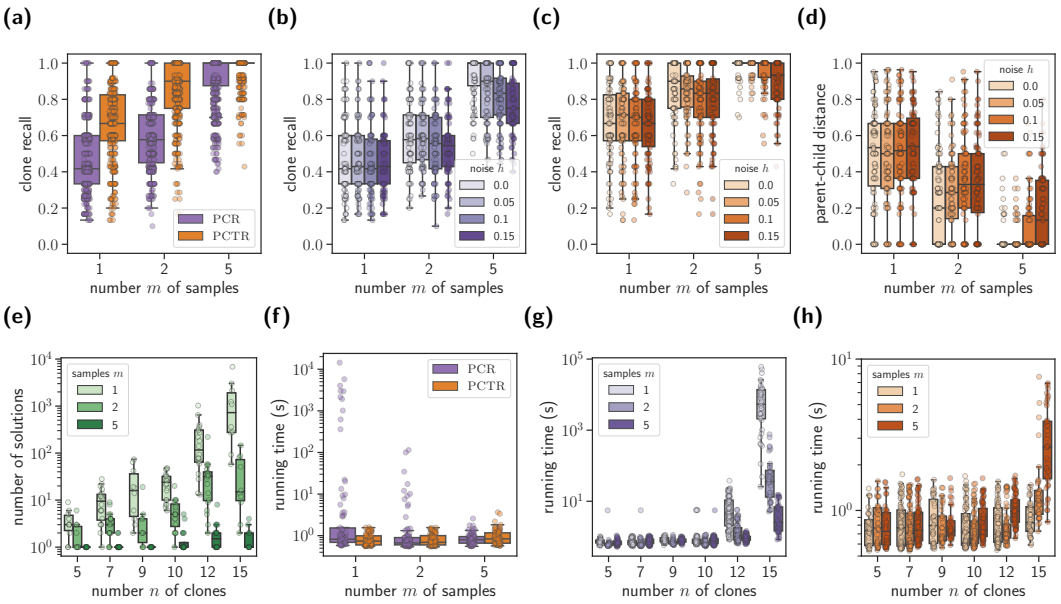


Figure 4 Simulations show that PACTION quickly and accurately reconstructs comprehensive clonal architectures. (a) Clone recall of PACTION in the PCR and PCTR mode for simulation instances with increasing number m of samples. Clone recall of PACTION in the (b) PCR mode and (c) PCTR mode for different noise levels h and number m of samples. (d) Parent-child distance between the clone tree in the ground truth and the solution of PACTION in the PCTR mode for simulation instances with increasing number m of samples. (e) Number of solutions to the error-free version of the PCTR problem (with additional constraint of $J(U, U_1, U_2) = 0$) by SPRUCE [9] for increasing number n of clones. (f) Running time of PACTION in the PCR and PCTR modes for simulation instances with increasing number m of samples. Running time of PACTION in the (g) PCR mode and the (h) PCTR mode for simulation instances with increasing number n of clones and number m of samples.

found that the PCTR solver better handles increasing noise levels h, with a medial clone recall of 1 for noise level $h = 0$ as well as $h = 0.05$ when number m of samples is 5 (Figure 4c and Figure S1).

Next, we investigated how well PACTION in PCTR mode infers ground truth clone trees T^*. To that end, we computed the parent-child distance [15] between the predicted clone tree T and the clone tree T^* in the ground truth. Specifically, the *parent-child distance* equals the ratio between the size $|E(T) \triangle E(T^*)|$ of the symmetric difference of the edge sets by the size $|E(T) \cup E(T^*)|$ of the union of edge sets. We observed that the clone tree distance is inversely correlated with the clone recall and when the clone recall is 1, the predicted clone tree matches the ground truth perfectly (Figure 4d). Indeed, we observed that performance increases with increasing number m of samples, e.g., for $m = 5$ samples the median parent-child distance is 0 for noise levels $h \in \{0, 0.05, 0.1\}$ indicating that in the majority of these instances PACTION perfectly inferred ground truth trees. The reason why performance drops for decreasing number of samples is because the number of solutions increases with decreasing number of samples (Figure 4e). We used the correspondence between the PCTR problem (subject to the constraint that $J(U, U_1, U_2) = 0$, i.e., the proportions are error-free) and the perfect phylogeny mixture problem solved by SPRUCE [9] to enumerate all solutions for $h = 0$ instances. For instances with a large number of optimal solutions, the PCTR problem and consequently the MILP lacks additional constraints to disambiguate between solutions, thus sometimes reporting solutions that do not match the ground truth.

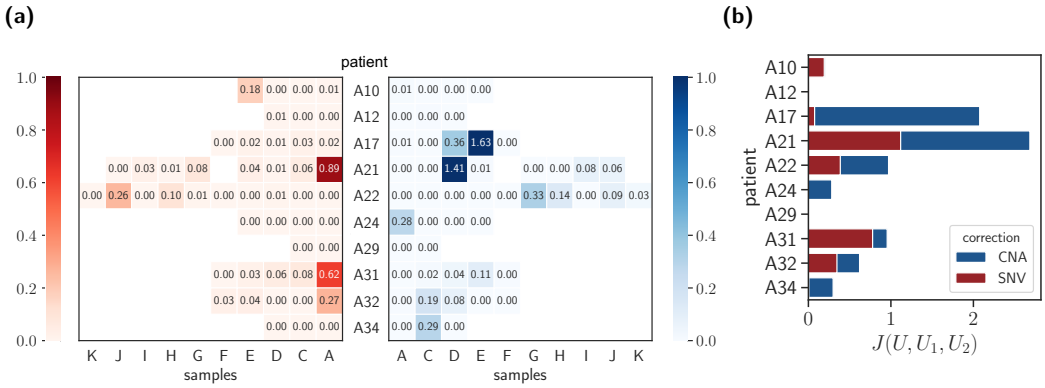


Figure 5 Overview of PACTION results on samples from 10 metastatic prostate cancer patients [16]. (a) The corrections made by PACTION to the SNV and CNA clone proportions in the samples from each of the 10 patients. (b) The total correction made to clone proportions $J(U, U_1, U_2)$ in samples from each patient.

Finally, we investigated the running times of PACTION in PCR and PCTR modes. Overall, the running times in PCR mode (median of 0.79 s and mean of 385.52 s) were larger than PCTR mode (median of 0.77 s and mean of 0.95 s), likely due to the tree constraints providing more guidance for the MILP solver (Table S1). Interestingly, while running time decreased with increasing number m of samples in PCR mode, the opposite is true in PCTR mode. The reason is that in PCTR mode the MILP is often solved in the first iteration prior to branching, where the running time of solving the linear programming relaxation will depend on the size of the formulation, which in turn depends on m. However, in PCR mode, the solver requires branching, and here additional constraints due to more samples will provide stronger bounds that will lead to more pruning and reduction in overall running time.

In summary, our simulations demonstrate that PACTION is able to quickly and accurately reconstruct ground truth clonal architectures under varying noise levels h, especially when the number m is large and when run in PCTR mode.

5.2 Metastatic prostate cancer

In this study, we analyze whole-genome sequencing data from 49 tumor samples from 10 metastatic prostate cancer patients [16]. In a previous analysis of this data, Gundem et al. [16] identified SNV clones and reconstructed the SNV clone tree for each of the 10 patients. To further investigate the role of CNAs on tumor evolution, the authors annotated the SNV clone trees with CNA events in a *post hoc* analysis by manually comparing and matching frequencies of SNVs and CNAs. However, this approach does not allow us to identify tumor clones that are only distinguished by different CNAs and have the same SNVs. Therefore, there is no information about CNA-only driven tumor clones nor information about the ordering of the CNA events and the SNV events on the same edge of the tree. Such information is crucial to understand cancer progression [38] and is the subject of numerous studies [17, 20, 34]. Therefore, we investigated whether we can use PACTION to provide a more comprehensive analysis of these tumor clonal compositions by jointly considering SNVs and CNAs.

We applied PACTION to previously inferred SNV and CNA clone proportions. First, we used the SNV clone proportions as well as the SNV clone tree T_1 inferred for each patient by Gundem et al. [16]. Note that each edge of the SNV tree represents a cluster of SNV

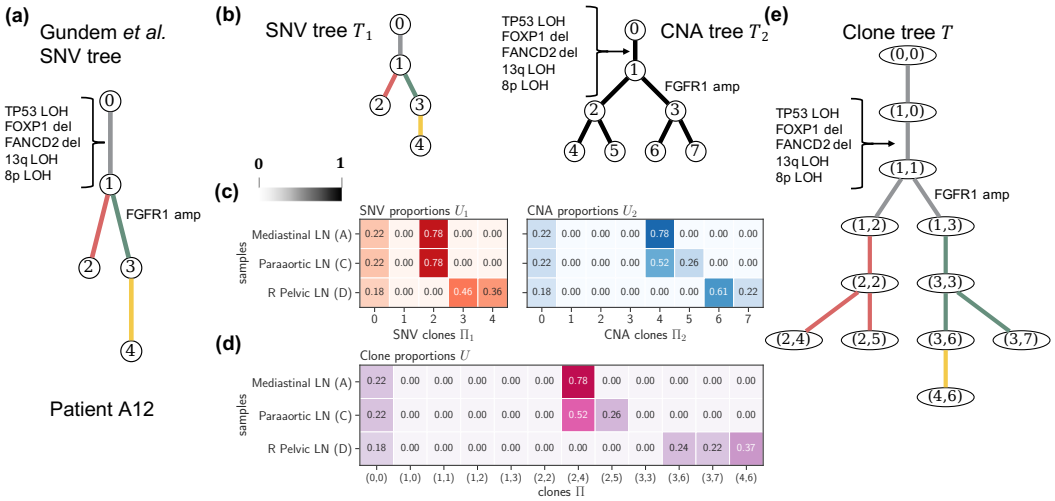


Figure 6 PACTION results for patient A12. (a) The SNV clone tree reported by Gundem et al. [16] where the authors manually annotated edges with CNA events. (b) SNV clone tree T_1 and CNA clone tree T_2 describing the evolution of the SNV clones Π_1 and CNA clones Π_2 in the tumor samples of patient A12, respectively. (c) Proportions U_1 of SNV clones Π_1 and proportions U_2 of CNA clones Π_2 in the four samples of patient A12. (d) Proportions U of tumor clones Π in the four samples of patient A12 inferred by PACTION. (e) Reconciled clone tree T inferred by PACTION. amp: amplification, del: deletion, LOH: loss of heterozygosity.

mutations. As such, we computed the SNV clone proportions U_1 using the published cancer cell fractions of SNVs (details in Appendix C). Second, we used the CNA clones obtained from a previous copy-number analysis [44] of the same patients. Since this previous analysis does not provide CNA clone trees, we enumerated all possible binary trees [2] with the CNA clones as the leaves and independently ran PACTION in PCTR mode with each tree as input. We then selected the CNA clone tree with the smallest correction $J(U, U_1, U_2)$, which for each patient was unique. Overall, we ultimately obtained SNV trees with $n_1 \in \{5, \ldots, 16\}$ clones and CNA trees with $n_2 \in \{4, \ldots, 8\}$ clones across $m \in \{2, \ldots, 10\}$ samples (Table S2).

In all patients but A29, we found that one cannot reconcile independently-inferred SNV and CNA clone trees without additional corrections to the clone proportions. Importantly, this observation highlights that the clone proportions inferred by existing methods are generally characterized by errors (Figure 5a). As previously demonstrated in our simulation study, PACTION, however, reliably handles the presence of noise, enabling the inference of the complete clonal composition and tumor evolution with limited corrections for all patients. Specifically, the corrections applied by PACTION were limited to only a few samples per patient, potentially indicating sample-specific errors in previous analysis or samples with higher levels of noise. Importantly, we also observed that corrections were uniformly needed for both SNV and CNA clone proportions (Figure 5). This important observation highlights that both features are generally characterized by errors and, therefore, one cannot simply leave one feature fixed and use it to reconcile the other feature, as done previously [16].

Notably, we found that the reconciled clone trees inferred by PACTION reveal additional branching events that were previously missed. As an example, in patient A12, Gundem et al. [16] inferred an SNV clone tree with five clones and annotated this tree with five clonal CNA events, including loss-of-heterozygosity (LOH) of gene TP53 and chromosomes 8p and 13q, as well as deletions of genes FOXP1 and FANCD2 (gray edge in Figure 6a). The

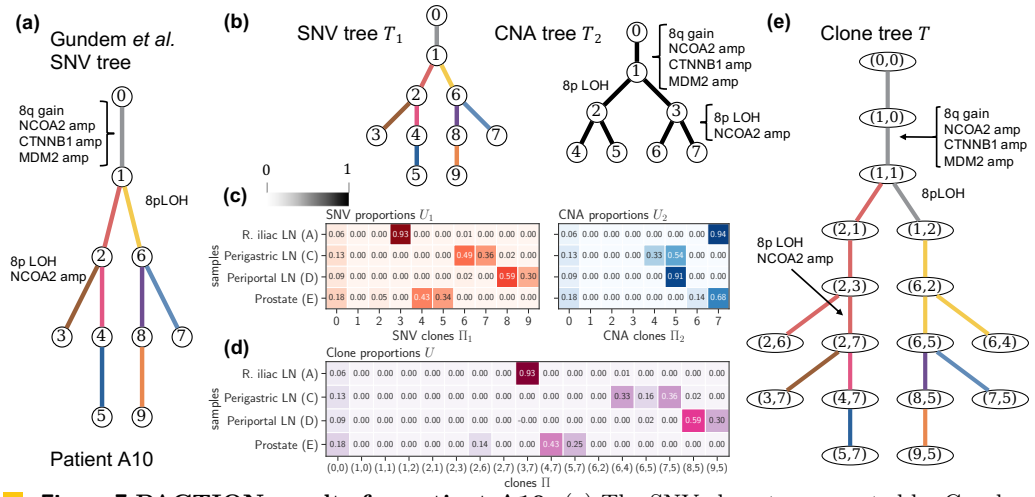


Figure 7 PACTION results for patient A10. (a) The SNV clone tree reported by Gundem et al. [16] where the authors manually annotated edges with CNA events. (b) SNV clone tree T_1 and CNA clone tree T_2 describing the evolution of the SNV clones Π_1 and CNA clones Π_2 in the tumor samples of patient A12, respectively. (c) Proportions U_1 of SNV clones Π_1 and proportions U_2 of CNA clones Π_2 in the four samples of patient A10. (d) Proportions U of tumor clones Π in the four samples of patient A10 inferred by PACTION. (e) Reconciled clone tree T inferred by PACTION. amp: amplification, LOH: loss of heterozygosity.

tree also contains a single subclonal CNA event, amplification of gene FGFR1 (green edge in Figure 6a). When using PACTION to analyze the previously-inferred SNV and CNA clone proportions, we reconstructed a reconciled clone tree with higher resolution. In fact, PACTION reconstructed a more refined clone tree with 12 clones while only applying modest corrections to the input clone proportions (Figure 5a). Similarly to the published tree, PACTION's inferred clone tree contains a trunk with the same four clonal CNA events. However, PACTION's tree contains additional branching events that are absent in the published SNV tree. Specifically, we observed that two SNV clones in the published tree (i.e., 2 and 3) were split into multiple clones in PACTION's refined tree (i.e., (2, 2), (2, 4), and (2, 5) for SNV clone 2, and (3, 3), (3, 6), and (3, 7) for SNV clone 3). Importantly, a subset of these refined clones are present at large proportions in the sequenced samples (Figure 6d), thus showing that PACTION enables a more fine-grained analysis of current sequencing data.

Finally, we found that the more refined clone trees inferred by PACTION also reveal novel insights about the relative temporal ordering of SNVs and CNAs. This phenomenon is particularly interesting in patient A10 (Figure 7a), for which PACTION inferred a clone tree with 17 clones and relatively high corrections to the previous SNV clone proportions (Figure 7b-d). PACTION's tree recapitulates the same four clonal CNAs identified in the previous tree, including gain of chromosome 8q and amplifications of genes NCOA2, CTNNB1 and MDM2 (gray edge in Figure 7a). Importantly, PACTION's tree also recapitulates subclonal CNA events as in the previous tree but further revealed that these CNA events precede the SNV events placed on the same edges in the published SNV clone tree (Figure 7e). More specifically, PACTION revealed that LOH of chromosome 8p and amplification of gene NCOA2 occur on the edge from clone $(2, 3)$ to $(2, 7)$ which precedes the SNV cluster represented by the edge from clone $(2, 7)$ to $(3, 7)$. Similarly, PATION revealed that LOH of chromosome 8p occurs on the edge from clone $(1, 1)$ to $(1, 2)$ which precedes the SNV cluster represented by the edge from clone $(1, 2)$ to $(6, 2)$.

In summary, we demonstrated on metastatic prostate cancer patients that PACTION is able to resolve the temporal ordering of mutations and reveal branching events that are either unclear or hidden when the SNV tree or the CNA tree are considered in isolation.

6 Discussion

In this paper, we introduced PACTION, a new algorithm that infers comprehensive tumor clonal compositions by reconciling the clones proportions of both SNVs and CNAs that are inferred by existing methods. Our algorithm can additionally leverage SNV and CNA clone trees reconstructed by existing methods to obtain a refined tumor clone tree and correct potential errors in the input proportions. We formulated two problems, the PCR problem to infer the clones and their proportions, and the PCTR problem to additionally infer tumor clone trees with both SNVs and CNAs. We showed that both problems are NP-hard and can be solved exactly by PACTION using two mixed inter linear programming formulations. We demonstrated the performance of PACTION on simulations, showing that our method accurately reconciles clone trees, reliably handles errors in clone proportions, and scales to practical input sizes. Finally, we applied our method to whole-genome sequencing data from 10 metastatic prostate cancer patients [16], obtaining a higher resolution view of tumor evolution than previously reported.

In addition to the contributions of this study, we foresee four major avenues for future research. First, building upon the established relationship of the error-free PCTR and the cladistic multi-state perfect phylogeny deconvolution problems, we can adapt the existing method SPRUCE [9] to enumerate all possible solution of the PCTR problem in the presence of errors in the input proportions. Second, PACTION can be extended to account for uncertainty in the input clone trees and quantify its effect on the solution space. One way of incorporating the uncertainty in the input clone trees, is to consider a set of possible clone trees for each feature instead of a single input tree, choosing the best tree that leads to the most parsimonious solution. Moreover, we plan to adapt the PCR and PCTR to incorporate probabilistic models that account for uncertainty in the estimated clone proportions. Third, the PCR and PCTR problems can be generalized to reconcile more than two features. For instance, in addition to SNVs and CNAs, tumor cells may be partitioned into clones based on RNA expression or DNA methylation profiles. Finally, a likelihood-based objective function could be used to incorporate a joint evolutionary model for SNVs and CNAs [32].

References

1 Ravindra K Ahuja, Thomas L Magnanti, James B Orlin, and K Weihe. Network flows: theory, algorithms and applications. *ZOR-methods and models of operations research*, 41(3):252–254, 1995.

2 Johnathan Barnett, Hannah Correia, Peter Johnson, Michael Laughlin, and Kathryn Wilson. Darwin meets graph theory on a strange planet: Counting full n-ary trees with labeled leafs. *Alabama Journal of Mathematics*, 2010.

3 Rebecca A Burrell, Nicholas McGranahan, Jiri Bartek, and Charles Swanton. The causes and consequences of genetic heterogeneity in cancer evolution. *Nature*, 501(7467):338–345, 2013.

4 Giovanni Ciriello, Martin L Miller, Bülent Arman Aksoy, Yasin Senbabaoglu, Nikolaus Schultz, and Chris Sander. Emerging landscape of oncogenic signatures across human cancers. *Nature genetics*, 45(10):1127–1133, 2013.

5 Amit G Deshwar, Shankar Vembu, Christina K Yung, Gun Ho Jang, Lincoln Stein, and Quaid Morris. Phylowgs: reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. *Genome biology*, 16(1):1–20, 2015.

6 Mohammed El-Kebir, Layla Oesper, Hannah Acheson-Field, and Benjamin J Raphael. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):i62–i70, 2015.

7 Mohammed El-Kebir, Benjamin J Raphael, Ron Shamir, Roded Sharan, Simone Zaccaria, Meirav Zehavi, and Ron Zeira. Copy-number evolution problems: complexity and algorithms. In *International Workshop on Algorithms in Bioinformatics*, pages 137–149. Springer, 2016.

8 Mohammed El-Kebir, Benjamin J Raphael, Ron Shamir, Roded Sharan, Simone Zaccaria, Meirav Zehavi, and Ron Zeira. Complexity and algorithms for copy-number evolution problems. *Algorithms for Molecular Biology*, 12(1):1–11, 2017.

9 Mohammed El-Kebir, Gryte Satas, Layla Oesper, and Benjamin J. Raphael. Inferring the Mutational History of a Tumor Using Multi-state Perfect Phylogeny Mixtures. *Cell Systems*, 3(1):43–53, 2016. doi:10.1016/j.cels.2016.07.004.

10 D Fernández-Baca. The perfect phylogeny problem. In D Z Zu and X Cheng, editors, *Steiner Trees in Industries*. Kluwer Acedemic Publishers, 2000.

11 Andrej Fischer, Ignacio Vázquez-García, Christopher JR Illingworth, and Ville Mustonen. High-definition reconstruction of clonal composition in cancer. *Cell reports*, 7(5):1740–1752, 2014.

12 Michael R Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

13 Michael R. Garey and David S. Johnson. Computers and intractability. a guide to the theory of np-completeness, 1983.

14 Charles Gawad, Winston Koh, and Stephen R Quake. Single-cell genome sequencing: current state of the science. *Nature Reviews Genetics*, 17(3):175, 2016.

15 Kiya Govek, Camden Sikes, and Layla Oesper. A consensus approach to infer tumor evolutionary histories. In *Proceedings of the 2018 Acm international conference on bioinformatics, computational biology, and health informatics*, pages 63–72, 2018.

16 Gunes Gundem, Peter Van Loo, Barbara Kremeyer, Ludmil B Alexandrov, Jose MC Tubio, Elli Papaemmanuil, Daniel S Brewer, Heini ML Kallio, Gunilla Högnäs, Matti Annala, et al. The evolutionary history of lethal metastatic prostate cancer. *Nature*, 520(7547):353–357, 2015.

17 Jun Guo, Hanliang Guo, and Zhanyi Wang. Inferring the temporal order of cancer gene mutations in individual tumor samples. *PLoS One*, 9(2):e89244, 2014.

18 Mariam Jamal-Hanjani, Gareth A Wilson, Nicholas McGranahan, Nicolai J Birkbak, Thomas BK Watkins, Selvaraju Veeriah, Seema Shafi, Diana H Johnson, Richard Mitter, Rachel Rosenthal, et al. Tracking the evolution of non–small-cell lung cancer. *New England Journal of Medicine*, 376(22):2109–2121, 2017.

19 Yuchao Jiang, Yu Qiu, Andy J Minn, and Nancy R Zhang. Assessing intratumor heterogeneity and tracking longitudinal and spatial clonal evolutionary history by next-generation sequencing. *Proceedings of the National Academy of Sciences*, 113(37):E5528–E5537, 2016.

20 Sahand Khakabimamaghani, Dujian Ding, Oliver Snow, and Martin Ester. Uncovering the subtype-specific temporal order of cancer pathway dysregulation. *PLoS computational biology*, 15(11):e1007451, 2019.

21 Paul L Krapivsky and Sidney Redner. Organization of growing random networks. *Physical Review E*, 63(6):066123, 2001.

22 Marco L Leung, Alexander Davis, Ruli Gao, Anna Casasent, Yong Wang, Emi Sei, Eduardo Vilar, Dipen Maru, Scott Kopetz, and Nicholas E Navin. Single-cell dna sequencing reveals a late-dissemination model in metastatic colorectal cancer. *Genome research*, 27(8):1287–1299, 2017.

23 Salem Malikic, Andrew W McPherson, Nilgun Donmez, and Cenk S Sahinalp. Clonality inference in multiple tumor samples using phylogeny. *Bioinformatics*, 31(9):1349–1356, 2015.

24 Nicholas McGranahan and Charles Swanton. Biological and therapeutic impact of intratumor heterogeneity in cancer evolution. *Cancer cell*, 27(1):15–26, 2015.

25 Andrew W McPherson, Andrew Roth, Gavin Ha, Cedric Chauve, Adi Steif, Camila PE de Souza, Peter Eirew, Alexandre Bouchard-Côté, Sam Aparicio, S Cenk Sahinalp, et al. Remixt: clone-specific genomic structure estimation in cancer. *Genome biology*, 18(1):1–14, 2017.

26 Faiyaz Notta, Michelle Chan-Seng-Yue, Mathieu Lemire, Yilong Li, Gavin W Wilson, Ashton A Connor, Robert E Denroche, Sheng-Ben Liang, Andrew MK Brown, Jaeseung C Kim, et al. A renewed model of pancreatic cancer evolution based on genomic rearrangement patterns. *Nature*, 538(7625):378–382, 2016.

27 Peter C Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976.

28 Layla Oesper, Ahmad Mahmoody, and Benjamin J Raphael. Theta: inferring intra-tumor heterogeneity from high-throughput dna sequencing data. *Genome biology*, 14(7):1–21, 2013.

29 Victoria Popic, Raheleh Salari, Iman Hajirasouliha, Dorna Kashef-Haghighi, Robert B West, and Serafim Batzoglou. Fast and scalable inference of multi-sample cancer lineages. *Genome biology*, 16(1):1–17, 2015.

30 Dikshant Pradhan and Mohammed El-Kebir. On the non-uniqueness of solutions to the perfect phylogeny mixture problem. In *RECOMB International conference on Comparative Genomics*, pages 277–293. Springer, 2018.

31 Gryte Satas and Benjamin J Raphael. Tumor phylogeny inference using tree-constrained importance sampling. *Bioinformatics*, 33(14):i152–i160, 2017.

32 Gryte Satas, Simone Zaccaria, Geoffrey Mon, and Benjamin J Raphael. Scarlet: Single-cell tumor phylogeny inference with copy-number constrained mutation losses. *Cell Systems*, 10(4):323–332, 2020.

33 Roland F Schwarz, Anne Trinh, Botond Sipos, James D Brenton, Nick Goldman, and Florian Markowetz. Phylogenetic quantification of intra-tumour heterogeneity. *PLoS Comput Biol*, 10(4):e1003535, 2014.

34 Kathleen Sprouffske, John W Pepper, and Carlo C Maley. Accurate reconstruction of the temporal order of mutations in neoplastic progression. *Cancer prevention research*, 4(7):1135–1144, 2011.

35 Francesco Strino, Fabio Parisi, Mariann Micsinai, and Yuval Kluger. Trap: a tree approach for fingerprinting subclonal tumor composition. *Nucleic acids research*, 41(17):e165–e165, 2013.

36 Linda K Sundermann, Jeff Wintersinger, Gunnar Rätsch, Jens Stoye, and Quaid Morris. Reconstructing tumor evolutionary histories and clone trees in polynomial-time with submarine. *PLoS computational biology*, 17(1):e1008400, 2021.

37 Maxime Tarabichi, Adriana Salcedo, Amit G Deshwar, Máire Ni Leathlobhair, Jeff Wintersinger, David C Wedge, Peter Van Loo, Quaid D Morris, and Paul C Boutros. A practical guide to cancer subclonal reconstruction from dna sequencing. *Nature methods*, 18(2):144–155, 2021.

38 Hamid Teimouri and Anatoly B Kolomeisky. Temporal order of mutations influences cancer initiation dynamics. *bioRxiv*, 2021.

39 ICGC The, TCGA Pan-Cancer Analysis of Whole, Genomes Consortium, et al. Pan-cancer analysis of whole genomes. *Nature*, 578(7793):82, 2020.

40 Thomas BK Watkins, Emilia L Lim, Marina Petkovic, Sergi Elizalde, Nicolai J Birkbak, Gareth A Wilson, David A Moore, Eva Grönroos, Andrew Rowan, Sally M Dewhurst, et al. Pervasive chromosomal instability and karyotype order in tumour evolution. *Nature*, 587(7832):126–132, 2020.

41 Taoyang Wu, Vincent Moulton, and Mike Steel. Refining phylogenetic trees given additional data: An algorithm based on parsimony. *IEEE/ACM transactions on computational biology and bioinformatics*, 6(1):118–125, 2008.

42 Simone Zaccaria, Mohammed El-Kebir, Gunnar W Klau, and Benjamin J Raphael. The copy-number tree mixture deconvolution problem and applications to multi-sample bulk sequencing tumor data. In *International Conference on Research in Computational Molecular Biology*, pages 318–335. Springer, 2017.

43 Simone Zaccaria, Mohammed El-Kebir, Gunnar W Klau, and Benjamin J Raphael. Phylogenetic copy-number factorization of multiple tumor samples. *Journal of Computational Biology*, 25(7):689–708, 2018.

44 Simone Zaccaria and Benjamin J Raphael. Accurate quantification of copy-number aberrations and whole-genome duplications in multi-sample tumor sequencing data. *Nature communications*, 11(1):1–13, 2020.

A MILP formulation for the PCTR problem

$$\min \sum_{p=1}^{m}\sum_{i=1}^{n_1} c^{(1)}_{p,i} + \sum_{p=1}^{m}\sum_{j=1}^{n_2} c^{(2)}_{p,j} \tag{2}$$

$$\text{s.t. } c^{(1)}_{p,i} \geq \sum_{j=1}^{n_2} u_{p,i,j} - u^{(1)}_{p,i} \qquad \forall p \in [m], i \in [n_1], \tag{3}$$

$$c^{(1)}_{p,i} \geq u^{(1)}_{p,i} - \sum_{j=1}^{n_2} u_{p,i,j} \qquad \forall p \in [m], i \in [n_1], \tag{4}$$

$$c^{(2)}_{p,j} \geq \sum_{i=1}^{n_1} u_{p,i,j} - u^{(2)}_{p,j} \qquad \forall p \in [m], j \in [n_2], \tag{5}$$

$$c^{(2)}_{p,j} \geq u^{(2)}_{p,j} - \sum_{i=1}^{n_1} u_{p,i,j} \qquad \forall p \in [m], j \in [n_2], \tag{6}$$

$$u_{p.i,j} \leq x_{i,j} \qquad \forall p \in [m], i \in [n_1], j \in [n_2], \tag{7}$$

$$\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} u_{p,i,j} = 1 \qquad \forall p \in [m], \tag{8}$$

$$z^{(1)}_{(i,i'),j} \leq x_{i,j} \qquad \forall (i,i') \in E(T_1), j \in [n_2], \tag{9}$$

$$z^{(1)}_{(i,i'),j} \leq x_{i',j} \qquad \forall (i,i') \in E(T_1), j \in [n_2], \tag{10}$$

$$z^{(1)}_{(i,i'),j} \geq x_{i,j} + x_{i',j} - 1 \qquad \forall (i,i') \in E(T_1), j \in [n_2]. \tag{11}$$

$$z^{(2)}_{i,(j,j')} \leq x_{i,j} \qquad \forall i \in [n_1], (j,j') \in E(T_2), \tag{12}$$

$$z^{(2)}_{i,(j,j')} \leq x_{i,j'} \qquad \forall i \in [n_1], (j,j') \in E(T_2), \tag{13}$$

$$z^{(2)}_{i,(j,j')} \geq x_{i,j} + x_{i,j'} - 1 \qquad \forall i \in [n_1], (j,j') \in E(T_2), \tag{14}$$

$$\sum_{j=1}^{n_2} z^{(1)}_{(i,i'),j} = 1 \qquad \forall (i,i') \in E(T_1), \tag{15}$$

$$\sum_{i=1}^{n_1} z^{(2)}_{i,(j,j')} = 1 \qquad \forall (j,j') \in E(T_2), \tag{16}$$

$$x_{i,j} \in \{0,1\}, \qquad \forall i \in [n_1], j \in [n_2], \tag{17}$$

$$u^{(1)}_{p,i} \in [0,1], \qquad \forall p \in [m], i \in [n_1], \tag{18}$$

$$u^{(2)}_{p,j} \in [0,1], \qquad \forall p \in [m], j \in [n_2], \tag{19}$$

$$c^{(1)}_{p,i} \in [0,1], \qquad \forall p \in [m], i \in [n_1], \tag{20}$$

$$c^{(2)}_{p,j} \in [0,1], \qquad \forall p \in [m], j \in [n_2], \tag{21}$$

$$z^{(1)}_{(i,i'),j} \geq 0 \qquad \forall (i,i') \in E(T_1), j \in [n_2], \tag{22}$$

$$z^{(2)}_{i,(j,j')} \geq 0 \qquad \forall i \in [n_1], (j,j') \in E(T_2). \tag{23}$$

B Simulation Details

We perturb the proportion matrices U_1 and U_2 by introducing noise following a user-defined level $h \in [0,1]$. For each sample $p \in [m]$, let $\mathbf{u}_p^{(1)} = [u_{p,i}^{(1)}]$ for $i \in [n_1]$ and $\mathbf{u}_p^{(2)} = [u_{p,j}^{(2)}]$ for $j \in [n_2]$. The perturbed proportions $\bar{\mathbf{u}}_p^{(1)}$ and $\bar{\mathbf{u}}_p^{(2)}$ are drawn from the following distributions

$$\bar{\mathbf{u}}_p^{(1)} \sim (1-h)\mathbf{u}_p^{(1)} + h\text{Dir}(\mathbf{1}_{n_1}), \quad \forall p \in [m],$$
$$\bar{\mathbf{u}}_p^{(2)} \sim (1-h)\mathbf{u}_p^{(2)} + h\text{Dir}(\mathbf{1}_{n_2}), \quad \forall p \in [m].$$

The resulting proportion matrices are $\bar{U}_1 = [\bar{u}_{p,i}^{(1)}]$ for $p \in [m], i \in [n_1]$ and $\bar{U}_2 = [\bar{u}_{p,j}^{(2)}]$ for $p \in [m], j \in [n_2]$. Note that when noise level $h = 0$, we have $\bar{U}_1 = U_1$ and $\bar{U}_2 = U_2$. Also, for any $h \in [0,1]$, the matrices $\bar{U}_1$ and $\bar{U}_2$ satisfy the conditions laid out in the definition of proportion matrices (Definition 2).

C Computation of SNV Clone Proportions

Each edge of the SNV clone tree T_1 reported by Gundem et al. [16] represents a set of mutations, also known as mutation clusters. As such, for a SNV clone tree T_1 with n_1 vertices, there are $n_1 - 1$ mutation clusters. The authors have provided the cancer cell fraction (CCF) for each of the mutation clusters in each sample of the ten patients. They used pigeonhole principle (PPH) to construct the SNV clone tree manually. For a given patient, let $F \in [0,1]^{m \times (n_1 - 1)}$ be the CCF matrix such that $F = [f_{p,k}]$ and $f_{p,k}$ is the CCF of mutation cluster $k \in [n_1 - 1]$ in sample $p \in [m]$. The SNV clone tree T_1, excluding the root vertex which represent the normal cell, is used to construct a perfect phylogeny matrix B [30]. We use the perfect phylogeny matrix B and the CCF matrix F to get the proportion U' of SNV clones, excluding the normal clone, in each sample of the ten patients by solving the following linear program

$$\begin{aligned} &\min |F - BU'|_1, \\ &\text{s.t. } 0 \leq u_{p,i} \leq 1, \quad \forall p \in [m], i \in [n_1 - 1], \\ &\quad \sum_{i=1}^{n_1 - 1} u_{p,i} = 1, \quad \forall p \in [m], \end{aligned}$$

where $|\cdot|_1$ is the entry-wise L_1 norm. Finally, we correct the proportion matrix U' for the purity of the tumor samples (also known as tumor cellularity), which is the proportion of cancer cells in the tumor. We use the proportion of normal cells in each sample, inferred by HATCHet [44], to compute the purity of the tumor samples. Let $\boldsymbol{\gamma} \in [0,1]^{m \times 1}$ be a vector such that $\gamma_{p,1}$ is the purity of sample $p \in [m]$ inferred using HATCHet. The proportion matrix $U \in [0,1]^{m \times n_1}$ of the SNV clones is given by

$$U = \begin{bmatrix} \text{Diag}(\boldsymbol{\gamma})U' & \mathbf{1}_m - \boldsymbol{\gamma} \end{bmatrix}$$

where $\mathbf{1}_m$ is a $m \times 1$ vector with all entries equal to 1 and $\text{Diag}(\gamma)$ is a $m \times m$ diagonal matrix with the diagonal elements given by the entries of the vector $\boldsymbol{\gamma}$. It is easy to see that the proportion matrix U satisfies the conditions for being a proportion matrix (see Definition 1).

D Supplementary Figures and Tables

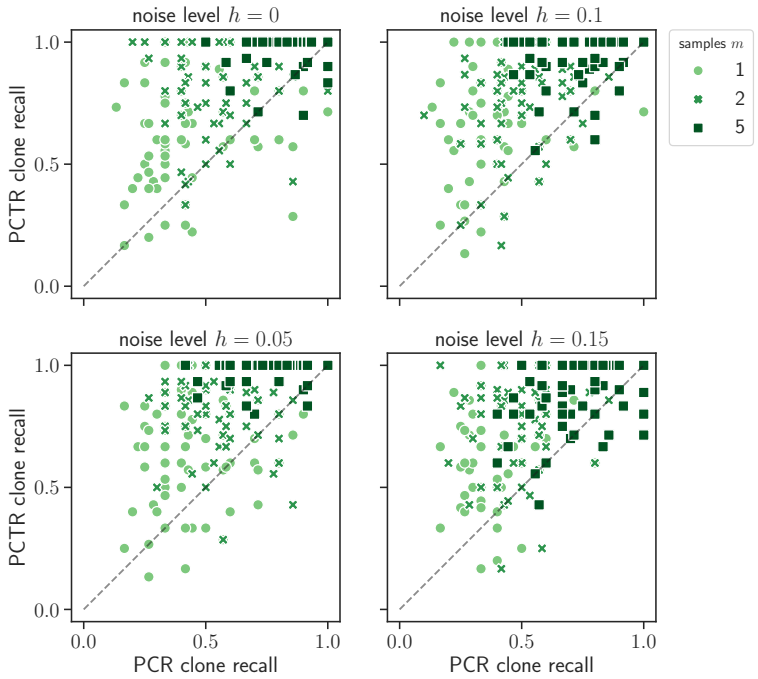


Figure S1 Clone recall for the two modes of PACTION on the simulated instances. We show the clone recall of PACTION with the PCR and the PCTR mode on the simulated instances for varying noise levels h and number m of samples. For majority of simulated instances, PACTION in the PCTR mode has a higher recall compared to the PCR mode.

Table S1 Median running time of PACTION in PCT and PCTR modes for simulation instances with varying number of samples m.

number of samples m	PCR runtime (s)	PCTR runtime (s)
1	0.84820	0.74365
2	0.6949	0.7379
5	0.81985	0.84460

Table S2 Statistics of the metastatic prostate cancer data [16]. Number m of samples, number n_1 of SNV clones and number n_2 of CNA clones for the 10 patients from Gundem et al. [16]. The CNA clones were identified using HATCHet [44].

patient	number m of samples	number n_1 of SNV clones	number n_2 of CNA clones
A10	4	10	8
A12	3	5	8
A17	5	11	6
A21	8	15	6
A22	10	16	4
A24	4	10	4
A29	2	6	4
A31	5	11	6
A32	5	13	6
A34	3	14	6

An Efficient Linear Mixed Model Framework for Meta-Analytic Association Studies Across Multiple Contexts

Brandon Jew[1]
Bioinformatics Interdepartmental Program, University of California, Los Angeles, CA, USA

Jiajin Li[1]
Department of Human Genetics, University of California, Los Angeles, CA, USA

Sriram Sankararaman
Department of Human Genetics, University of California, Los Angeles, CA, USA
Department of Computer Science, University of California, Los Angeles, CA, USA
Department of Computational Medicine, University of California, Los Angeles, CA, USA

Jae Hoon Sul[2] ✉
Department of Psychiatry and Biobehavioral Sciences,
University of California, Los Angeles, CA, USA

Abstract

Linear mixed models (LMMs) can be applied in the meta-analyses of responses from individuals across multiple contexts, increasing power to detect associations while accounting for confounding effects arising from within-individual variation. However, traditional approaches to fitting these models can be computationally intractable. Here, we describe an efficient and exact method for fitting a multiple-context linear mixed model. Whereas existing exact methods may be cubic in their time complexity with respect to the number of individuals, our approach for **m**ultiple-**c**ontext LMMs (mcLMM) is linear. These improvements allow for large-scale analyses requiring computing time and memory magnitudes of order less than existing methods. As examples, we apply our approach to identify expression quantitative trait loci from large-scale gene expression data measured across multiple tissues as well as joint analyses of multiple phenotypes in genome-wide association studies at biobank scale.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Computational genomics

Keywords and phrases Meta-analysis, Linear mixed models, multiple-context genetic association

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.10

Supplementary Material mcLMM is available as an R package:
Software (Source Code): `https://github.com/brandonjew/mcLMM`
archived at `swh:1:dir:c7953ff0d490aca333db79d5275f5a339e6bf2ec`

Funding *Brandon Jew*: National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650604.
Sriram Sankararaman: NIH R35GM125055, Alfred P. Sloan Research Fellowship, NSF III-1705121, CAREER 1943497.
Jae Hoon Sul: National Institute of Environmental Health Sciences (NIEHS) [K01 ES028064]; the National Science Foundation grant [#1705197]; the National Institute of Neurological Disorders and Stroke (NINDS) [R01 NS102371]; and NINDS [R03 HL150604].

[1] Equal contribution
[2] Corresponding author

1 Introduction

Over the last decade, the scale of genomic datasets has steadily increased. These datasets have grown to the size of hundreds of thousands of individuals [3] with millions soon to come [21]. Similarly, datasets for transcriptomics and epigenomics are growing to thousands of samples [1, 5, 14]. These studies provide valuable insight into the relationship between our genome and complex phenotypes [23].

Identifying these associations requires statistical models that can account for biases in study design that can negatively influence results through false positives or decreased power. Linear mixed models (LMMs) have been a popular choice for controlling these biases in genomic studies, utilizing variance components to account for issues such as population stratification [8]. These models can also be used to analyze studies with repeated measurements from individuals, such as replicates or measurements across different contexts. Meta-Tissue [20] is a method that applies this model in the context of identifying expression quantitative trait loci (eQTLs) across multiple tissues. In this framework, gene expression is measured in several tissues from the same individuals and the LMM is utilized to test the association between these values and genotypes. A meta-analytic approach is used to combined effects across multiple tissues to increase the power of detecting eQTLs. This approach has also been applied to increase power in genome-wide association studies (GWAS) by testing the association between genotypes and multiple related phenotypes [7].

However, these approaches are computationally intensive. Existing approaches for fitting these models are cubic in time complexity with respect to the number of samples across all contexts [8, 26]. Here, we present an ultra-fast LMM framework specifically for multiple-context studies. Our method, mcLMM, is linear in complexity with respect to the number of individuals and allows for statistical tests in a manner of hours rather than days or years with existing approaches. To illustrate the computational efficiency of mcLMM, we compare the runtime and memory usage of our method with EMMA and GEMMA [8, 26], two popular approaches for fitting these models. We further apply mcLMM to identify a large number of eQTLs in the Genotype-Tissue Expression (GTEx) dataset [5] and compare our results from METASOFT [6], which performs the meta-analysis of the mcLMM output, to a recent meta-analytic approach known as `mash` [22]. Finally, to demonstrate the practicality of mcLMM on modern datasets, we perform a multiple-phenotype GWAS combining over a million observations sampled from hundreds of thousands of individuals in the UK Biobank [3] within hours.

2 Methods

2.1 Linear Mixed Model

For multiple-context experiments with n individuals, t contexts, and c covariates, we fit the following linear mixed model

$$\mathbf{y} = X\beta + \mathbf{u} + \mathbf{e} \tag{1}$$

where $\mathbf{u} \sim N(0, \sigma_g^2 K)$, $\mathbf{e} \sim N(0, \sigma_e^2 I)$, $\mathbf{y} \in R^{nt}$ is a vectorized representation of the responses, $X \in R^{nt \times tc}$ is the matrix of covariates, $\beta \in R^{tc}$ is the vector of estimated coefficients, $K \in R^{nt \times nt}$ is a binary matrix where $K_{i,j} = 1$ indicates that sample i and sample j in Y come from the same individual, and $I \in R^{nt \times nt}$ is an identity matrix. X is structured such that both an intercept and the covariate effects are fit within each context. For sake of simplicity, dimensions of nt assume that there is no missing data; however, this is not

a requirement for the model. We note that this definition of K models within-individual variability as a random-effect, while within-context or across-individual variability is not included.

The full and restricted log-likelihood functions for this model are

$$l_F(\mathbf{y};\beta,\sigma_g,\delta) = \frac{1}{2}\left[-N\log\left(2\pi\sigma_g^2\right) - \log(|H|) - \frac{1}{\sigma_g^2}(\mathbf{y}-X\beta)^T H^{-1}(\mathbf{y}-X\beta)\right] \tag{2}$$

$$l_R(\mathbf{y};\beta,\sigma_g,\delta) = l_F(\mathbf{y};\beta,\sigma_g,\sigma_e) + \frac{1}{2}\left[tc\log(2\pi\sigma_g^2) + \log\left(|X^TX|\right) - \log\left(|X^TH^{-1}X|\right)\right] \tag{3}$$

where N is the total number of measurements made across the individuals and contexts, $\delta = \frac{\sigma_e^2}{\sigma_g^2}$, and $H = K + \delta I$ [24]. These likelihood functions are maximized with the generalized least squares estimator $\hat{\beta} = (X^TH^{-1}X)^{-1}X^TH^{-1}\mathbf{y}$ and $\hat{\sigma_g^2} = \frac{R}{N}$ in the full log-likelihood and $\hat{\sigma_g^2} = \frac{R}{N-tc}$ in the restricted log-likelihood, where $R = (\mathbf{y}-X\hat{\beta})^TH^{-1}(\mathbf{y}-X\hat{\beta})$. Our goal is to maximize these likelihood functions to estimate the optimal $\hat{\delta}$.

2.2 Likelihood refactoring in the general case

The EMMA algorithm optimizes these likelihoods for δ by refactoring them in terms of constants calculated from eigendecompositions of H and SHS, where $S = I - X(X^TX)^{-1}X^T$, that allow linear complexity optimization iterations with respect to the number of individuals [8]. The GEMMA algorithm further increases efficiency by replacing the SHS eigendecomposition with a matrix-vector multiplication [26]. Both approaches require the eigendecomposition of at least one N by N matrix which is typically cubic in complexity. Here, we show that our specific definition of K as a binary indicator matrix allows us to refactor these likelihood functions without any eigendecomposition steps. It should be noted that EMMA and GEMMA can fit this model for any positive semidefinite K, while mcLMM is restricted to the definition described above.

We note that previous work has shown similar speedups when the matrix K is low rank and has a block structure as described here [10]. This work, FaST-LMM, shows that the likelihood functions can be computed in linear time with respect to the number of individuals after singular value decomposition of a matrix with complexity that is also linear with respect to the number of individuals. We improve upon these methods by recognizing that the eigenvalues of the K matrix are known beforehand, which allows for further efficiency in fitting this model. Furthermore, the FaST-LMM model assumes that all individuals within each context share additional covariance while mcLMM assumes that all contexts observed within an individual share additional covariance.

First, note that $H = K+\delta I$ is a block diagonal matrix. Specifically, each block corresponds to an individual i with t_i contexts measured, where t_i is less than or equal to t depending on the number of contexts observed for individual i. Each block is equal to $[\mathbf{1}_{t_i} + \delta I_{t_i}] \in R^{t_i \times t_i}$, where $\mathbf{1}_{t_i}$ is a t_i by t_i matrix composed entirely of 1. These properties of H make its eigendecomposition and inverse directly known.

The eigenvalues of a block diagonal matrix are equal to the union of the eigenvalues of each block. Moreover, the eigenvalues of $[\mathbf{1}_{t_i} + \delta I_{t_i}]$ are $t_i + \delta$ with multiplicity 1 and δ with multiplicity $t_i - 1$. Therefore, H has eigenvalues δ with multiplicity $N - n$ and $t_i + \delta$ for each t_i. This provides our first refactoring

$$\log\left(|H|\right) = (N-n)\log\left(\delta\right) + \sum_{i=1}^{n}\log\left(t_i+\delta\right) \tag{4}$$

The inverse of a block diagonal matrix can also be computed by inverting each block individually. Moreover, using the Sherman-Morrison formula [16], the inverse of $[\mathbf{1}_{t_i} + \delta I_{t_i}]$ is known

$$(\mathbf{1}_{t_i} + \delta I_{t_i})^{-1} = -\frac{1}{t+\delta}\mathbf{1}_{t_i} + \frac{1}{\delta} I_{t_i} \tag{5}$$

Given each entry of H^{-1}, we can show algebraically that

$$X^T H^{-1} X = \frac{1}{\delta}(E - D) \tag{6}$$

$$E_{i,j} = \begin{cases} \sum\limits_{\text{ind} \in f(i)} x_{\text{ind},g(i)} x_{\text{ind},g(j)} & \text{if } f(i) = f(j) \\ 0 & \text{if } f(i) \neq f(j) \end{cases} \tag{7}$$

$$D_{i,j} = \sum_{g \in \text{groups}} \frac{1}{t_g + \delta} \sum_{\text{ind} \in f(i), f(j), g} x_{\text{ind},g(i)} x_{\text{ind},g(j)} \tag{8}$$

where $f(i) = i\%t$ (modulo operator) provides the context of a given 0-indexed column of X, $g(i) = i//t$ (integer division) provides the covariate of a given index. A group g defines the set of individuals that share the same number of measured contexts t_g. The expression "ind $\in f(i), f(j), g$" indicates the set of all individuals that have t_g measured contexts that include context i and j.

Note that with all values independent of δ precomputed, specifically the sum of covariate interactions within the sets of individuals indicated above, E is constant with respect to δ and each entry of the symmetric matrix D can be calculated in linear time with respect to the number of groups, which is less than or equal to the number of contexts t. For a given δ, we can compute $X^T H^{-1} X$ in $O(t(tc)^2)$ time complexity. Both the restricted and full log-likelihoods require the calculation of $(X^T H^{-1} X)^{-1}$. The restricted log-likelihood requires the additional calculation of $\log(|X^T H^{-1} X|)$. To calculate both of these terms, we perform a Cholesky decomposition of $X^T H^{-1} X = LL^*$, where * indicates the conjugate transpose. Given this decomposition, we can compute

$$\log(|X^T H^{-1} X|) = \sum_{i=1}^{tc} 2\log(L_{i,i}) \tag{9}$$

$$(X^T H^{-1} X)^{-1} = (L^*)^{-1} L^{-1} \tag{10}$$

These operations can be done in $O((tc)^3)$ time complexity.

Let $P(X)$ denote a projection matrix and $M(X) = (I - P(X))$. Note that both $P(X)$ and $M(X)$ are idempotent. The term remaining term in the likelihood functions, R, can be reformulated as follows

$$\begin{aligned} \mathbf{y} - X\hat{\beta} &= \mathbf{y} - X(X^T H^{-1} X)^{-1} X^T H^{-1} \mathbf{y} \\ &= (I - X(X^T H^{-1} X)^{-1} X^T H^{-1})\mathbf{y} \\ &= (I - P(X))\mathbf{y} \\ &= M(X)\mathbf{y} \end{aligned} \tag{11}$$

$$\begin{aligned} M(X)^T H^{-1} &= (I - X(X^T H^{-1} X)^{-1} X^T H^{-1})^T H^{-1} \\ &= (I - H^{-1} X (X^T H^{-1} X)^{-1} X^T) H^{-1} \\ &= H^{-1} - H^{-1} X (X^T H^{-1} X)^{-1} X^T H^{-1} \\ &= H^{-1} (I - X (X^T H^{-1} X)^{-1} X^T H^{-1}) \\ &= H^{-1} M(X) \end{aligned} \tag{12}$$

$$\begin{aligned} R &= (\mathbf{y} - X\hat{\beta})^T H^{-1} (\mathbf{y} - X\hat{\beta}) \\ &= \mathbf{y}^T M(X)^T H^{-1} M(X) \mathbf{y} \\ &= \mathbf{y}^T H^{-1} M(X) M(X) \mathbf{y} \\ &= \mathbf{y}^T H^{-1} M(X) \mathbf{y} \\ &= (\mathbf{y}^T H^{-1} \mathbf{y}) - (\mathbf{y}^T H^{-1} X (X^T H^{-1} X)^{-1} X^T H^{-1} \mathbf{y}) \\ &= a - \mathbf{b}^T (X^T H^{-1} X)^{-1} \mathbf{b} \\ &= a - \mathbf{b}^T (L^*)^{-1} L^{-1} \mathbf{b} \end{aligned} \tag{13}$$

The scalar a and vector $\mathbf{b}$ are a function of δ and can be algebraically formulated as

$$a = \frac{1}{\delta} \left(\left(\sum_{i=1}^{N} \mathbf{y}_i^2 \right) - \left(\sum_{g \in \text{groups}} \frac{1}{t_g + \delta} \sum_{\text{ind} \in g} (\sum \mathbf{y}_{\text{ind}})^2 \right) \right) \tag{14}$$

$$\mathbf{b}_i = \frac{1}{\delta} \left(\left(\sum_{\text{ind} \in \text{context(i)}} x_{\text{ind},g(i)} \mathbf{y}_{\text{ind},f(i)} \right) - \left(\sum_{g \in \text{groups}} \frac{1}{t_g + \delta} \sum_{\text{ind} \in f(i),g} x_{\text{ind},g(i)} (\sum \mathbf{y}_{\text{ind}}) \right) \right) \tag{15}$$

where $\sum \mathbf{y}_{\text{ind}}$ indicates the sum of responses across all contexts for an individual. With values independent of δ pre-calculated, a and $\mathbf{b}$ can be calculated in linear time with respect to the number of groups.

Note that Equations 16 and 17 remove terms that are independent of δ since they are not required for finding its optimal value, indicated by the $\approx$ symbol. We can reformulate the entire likelihood functions as follows

$$\begin{aligned} l_F(\mathbf{y}; \beta, \sigma_g, \delta) &= \frac{1}{2} \left[-N \log(2\pi\sigma_g^2) - \log(|H|) - \frac{1}{\sigma_g^2} (\mathbf{y} - X\beta)^T H^{-1} (\mathbf{y} - X\beta) \right] \\ &= \frac{1}{2} \left[-N \log(2\pi \frac{R}{N}) - \log(|H|) - N \right] \\ &= \frac{1}{2} \left[-N \log(2\pi \frac{R}{N}) - \left((N-n) \log(\delta) + \sum_{i=1}^{n} \log(t_i + \delta) \right) - N \right] \\ &\approx -N \log(a - \mathbf{b}^T (L^*)^{-1} L^{-1} \mathbf{b}) - \left((N-n) \log(\delta) + \sum_{i=1}^{n} \log(t_i + \delta) \right) \end{aligned} \tag{16}$$

$$\begin{aligned} l_R(\mathbf{y}; \beta, \sigma_g, \delta) &= l_F(\mathbf{y}; \beta, \sigma_g, \sigma_e) + \frac{1}{2} \left[tc \log(2\pi\sigma_g^2) + \log(|X^T X|) - \log(|X^T H^{-1} X|) \right] \\ &\approx (tc - N) \log(a - \mathbf{b}^T (L^*)^{-1} L^{-1} \mathbf{b}) \\ &- \left((N-n) \log(\delta) + \sum_{i=1}^{n} \log(t_i + \delta) \right) - \sum_{i=1}^{tc} 2 \log(L_{i,i}) \end{aligned} \tag{17}$$

These likelihoods are maximized for $\hat{\delta}$ using the optimize function in R. Each likelihood evaluation has a time complexity of $O((tc)^3 + n)$.

2.3 Likelihood refactoring with no missing data

When there is no missing data, the likelihood functions can be further simplified. Note that in this case, $N = nt$ and all $t_i = t$. Hence,

$$\begin{aligned}\log(|H|) &= (N-n)\log(\delta) + \sum_{i=1}^{n}\log(t_i+\delta) \\ &= (nt-n)log(\delta) + n\log(t+\delta)\end{aligned} \tag{18}$$

If the input terms $\mathbf{y}$, X, and K are permuted resulting in samples being sorted in order of context, and the covariates in X are sorted in order of context, we can decompose H and X into

$$H = (\mathbf{1}_t + \delta I_t) \otimes I_n \tag{19}$$

$$X = I_t \otimes X_{\text{dense}} \tag{20}$$

where $\otimes$ indicates the Kronecker product and $X_{\text{dense}} \in R^{n\times c}$ is a typical representation of the covariates for each individual without multiple contexts (i.e. samples as rows and covariates as columns). Utilizing the properties of Kronecker products, we can perform the following reformulation

$$\begin{aligned}(X^T H^{-1} X)^{-1} &= ((I_t \otimes X_{\text{dense}}^T)((\mathbf{1}_t+\delta I_t)\otimes I_n)^{-1}(I_t \otimes X_{\text{dense}}))^{-1} \\ &= ((\mathbf{1}_t+\delta I_t)^{-1} \otimes X_{\text{dense}}^T X_{\text{dense}})^{-1} \\ &= (\mathbf{1}_t+\delta I_t) \otimes (X_{\text{dense}}^T X_{\text{dense}})^{-1}\end{aligned} \tag{21}$$

$$\begin{aligned}\log\left(|(X^T H^{-1} X)^{-1}|\right) &= \log(|(\mathbf{1}_t+\delta I_t)\otimes(X_{\text{dense}}^T X_{\text{dense}})^{-1}|) \\ &= \log(|(\mathbf{1}_t+\delta I_t)|^c |(X_{\text{dense}}^T X_{\text{dense}})^{-1}|^t) \\ &= c\log(|(\mathbf{1}_t+\delta I_t)|) + t\log(|(X_{\text{dense}}^T X_{\text{dense}})^{-1}|) \\ &= c\log\left(\frac{1}{(t+\delta)\delta^{t-1}}\right) + t\log(|(X_{\text{dense}}^T X_{\text{dense}})^{-1}|) \\ &= c(-\log(t+\delta) - (t-1)\log(\delta)) + t\log(|(X_{\text{dense}}^T X_{\text{dense}})^{-1}|)\end{aligned} \tag{22}$$

Note that the remaining determinant in Equation 22 will not need to be calculated since it is independent of δ. Next, we show that $\hat{\beta}$ is independent of δ.

$$\begin{aligned}\hat{\beta} &= (X^T H^{-1} X)^{-1} X^T H^{-1}\mathbf{y} \\ &= \left((\mathbf{1}_t+\delta I_t)\otimes(X_{\text{dense}}^T X_{\text{dense}})^{-1}\right) X^T H^{-1}\mathbf{y} \\ &= \left((\mathbf{1}_t+\delta I_t)\otimes(X_{\text{dense}}^T X_{\text{dense}})^{-1}\right)(I_t \otimes X_{\text{dense}}^T)((\mathbf{1}_t+\delta I_t)^{-1}\otimes I_n)\mathbf{y} \\ &= \left((\mathbf{1}_t+\delta I_t)\otimes(X_{\text{dense}}^T X_{\text{dense}})^{-1} X_{\text{dense}}^T\right)((\mathbf{1}_t+\delta I_t)^{-1}\otimes I_n)\mathbf{y} \\ &= \left((\mathbf{1}_t+\delta I_t)(\mathbf{1}_t+\delta I_t)^{-1}\otimes(X_{\text{dense}}^T X_{\text{dense}})^{-1} X_{\text{dense}}^T\right)\mathbf{y} \\ &= \left(I_t \otimes (X_{\text{dense}}^T X_{\text{dense}})^{-1} X_{\text{dense}}^T\right)\mathbf{y}\end{aligned} \tag{23}$$

This form of $\hat{\beta}$ shows that the optimal coefficients are equivalent to fitting separate ordinary least squares (OLS) models for each context. We compute $\hat{\beta}$ by concatenating these OLS estimates. Given this term, we can also compute the residuals of this model $\mathbf{s} = (\mathbf{y} - X\hat{\beta})$ and reformulate R as follows.

$$\begin{aligned} R &= (\mathbf{y} - X\hat{\beta})^T H^{-1} (\mathbf{y} - X\hat{\beta}) \\ &= \mathbf{s}^T H^{-1} \mathbf{s} \\ &= \sum_{i=1}^{nt} \mathbf{s}_i \sum_{j=1}^{nt} \mathbf{s}_j H^{-1}_{j,i} \\ &= \frac{1}{\delta} \left(\sum_{i=1}^{nt} \mathbf{s}_i^2 \right) + \frac{1}{\delta(t+\delta)} \left(- \sum_{i=1}^{n} \left(\sum \mathbf{s}_{\text{ind}(i)} \right)^2 \right) \end{aligned} \tag{24}$$

The term $\sum \mathbf{s}_{\text{ind}(i)}$ denotes the sum of residuals for an individual across all contexts. Let $u = \sum_{i=1}^{nt} \mathbf{s}_i^2$ and $v = -\sum_{i=1}^{n} \left(\sum \mathbf{s}_{\text{ind}(i)} \right)^2$.

$$R = \frac{1}{\delta} u + \frac{1}{\delta(t+\delta)} v \tag{25}$$

Now we can reformulate the log-likelihoods, omitting terms that do not depend on δ.

$$\begin{aligned} l_F(\delta) &= -nt \log(R) - \log(|H|) \\ &= -nt \log \left(\frac{1}{\delta} u + \frac{1}{\delta(t+\delta)} v \right) - (nt - n) \log(\delta) - n \log(t+\delta) \\ &= -nt \log \left(u + \frac{1}{t+\delta} v \right) + n \log \left(\frac{\delta}{t+\delta} \right) \end{aligned} \tag{26}$$

$$\begin{aligned} l_R(\delta) &= (tc - nt) \log(R) - \log(|H|) - \log(|(X^T H^{-1} X)^{-1}|) \\ &= (tc - nt) \log \left(u + \frac{1}{t+\delta} v \right) + (c - n) \log \left(\frac{t+\delta}{\delta} \right) \end{aligned} \tag{27}$$

Both functions are differentiable with respect to δ. Moreover, both derivatives have the same root

$$\hat{\delta} = \frac{-tu - v}{u + v} \tag{28}$$

The scalar values u and v can be calculated by performing a separate OLS regression for each context, which can be completed in $O(t(nc^2 + c^3))$ time for a naive OLS implementation. Compared to the methods described above, this approach requires no iterative optimization and the estimate is optimal. Our implementation has a time complexity of $O(c^3 + nc^2 + tcn)$.

2.4 Resource requirement simulation comparison

We installed EMMA v1.1.2 and manually built GEMMA from its GitHub source (genetics-statistics/GEMMA.git, commit 9c5dfbc). We edited the source code of GEMMA to prevent the automatic addition of intercept term in the design matrix (commented out lines 1946 to 1954 of src/param.cpp).

Data were simulated using the mcLMM package. Sample sizes of 100, 200, 300, 400, and 500 were simulated with 50 contexts. Context sizes of 4, 8, 16, 32, and 64 were simulated with 500 samples. Data were simulated with $\sigma_e^2 = 0.2$ and $\sigma_g^2 = 0.4$ and a sampling rate of 0.5. Memory usage of each method was measured using the peakRAM R package (v 1.0.2).

2.5 False positive rate simulation

We simulated gene expression levels in multiple tissues for individuals where there were no eQTLs. In other words, gene expression levels were not affected by any SNPs. We considered 10,000 genes and 100 SNPs resulting in one million gene-SNP pairs. We simulated 1,000 individuals. We also examined false positive rates with 500 and 800 individuals. We generated 49 such datasets where the number of tissues varied from 2 to 50. To simulate the genotypes for each subject, we randomly generated two haplotypes (vectors consisting of 0 and 1) assuming a minor allele frequency (MAF) of 30%. To simulate gene expression levels from multiple tissues among the same individuals, we sampled gene expression from the following multivariate normal distribution:

$$\mathbf{y} \sim N(0, \sigma_g^2 \mathbf{K} + \sigma_e^2 \mathbf{I}) \tag{29}$$

where $\mathbf{y}$ is an $N \times T$ vector representing the gene expression levels of N individuals in T tissues and $\mathbf{K}$ is an $NT \times NT$ matrix corresponding the correlation between the subjects across the tissues. $K_{i,j} = 1$ when i and j are from two tissues of the same individuals, $K_{i,j} = 0$ otherwise. Here, we let $\sigma_g = \sigma_e = 0.5$. We used a custom R function (included with the mcLMM package) to simulate data from this distribution, which is based on sampling with a smaller covariance matrix for each block of measurements from an individual.

After generating the simulation datasets, we first ran mcLMM to obtain the estimated effect sizes and their standard errors, as well as the correlation matrices. The results from mcLMM were used as the input of METASOFT for meta-analysis to evaluate the significance. False positive rate was calculated as the proportion of gene-SNP pairs with p-values smaller than the significance level ($\alpha = 0.05$).

2.6 True positive simulations

We developed the true positive simulation framework based on a previous study describing `mash` [22]. We simulated effects for 20,000 gene-SNP pairs in 44 tissues, 400 of which have non-null effects (true positives) and 19,600 of which have null effects. Let β_{jr} denote the effects of the gene-SNP pair j in context/tissue r and β_j is a vector of effects across various tissues, including null effects and non-null effects. We simulated the gene expression levels for 1,000 individuals as:

$$\mathbf{y} = \beta_{jr}^T X + \mathbf{e} \tag{30}$$

where X denotes the genotypes of the individuals that were simulated as described in the false positive rate simulation. $\mathbf{e} \sim N(0, \sigma_g^2 \mathbf{K} + \sigma_e^2 \mathbf{I})$, which is similar to the simulation in the false positive rate simulation. For β_j, we defined two types of non-null effects and simulated them in different ways:

- Shared, structured effects: non-null effects are shared in all tissues and the sharing is structured. The non-null effects are similar in effect sizes and directions (up-regulation or down-regulation) across all tissues, and this similarity would be stronger among some subsets of tissues. For 19,600 null effects, we set $\beta_j = 0$. For 400 non-null effects, we assumed that each β_j independently followed a multivariate normal distribution with mean 0 and variance ωU_k, where k is an index number randomly sample from $1, \ldots, 8$. $\omega = |\omega'|, \omega' \sim N(0, 1)$ represents a scaling factor to help capture the full range of effects. U_k are 44×44 data-driven covariance matrices learned from the GTEx dataset, which are provided in [22].

- Shared, unstructured effects: non-null effects are shared in all tissues but the sharing is unstructured or independent across different tissues. For 19,600 null effects, we set $\beta_j = 0$. For 400 non-null effects, we sampled β_j from a multivariate normal distribution with mean of 0 and variance of $0.01I$, where I is a 44×44 identity matrix.

After simulating the gene expression levels $\mathbf{y}$, we first ran mcLMM on the simulated datasets to acquire the estimated effect sizes and their standard errors, as well as the correlation matrices. We then applied METASOFT for meta-analysis with mcLMM outputs to evaluate the significance. For `mash`, we first performed simple linear regression to get the estimates of the effects and their standard errors in each tissue separately. These estimates and standard errors were used as the inputs for `mash`, which returned the measure of significance for each effect, the local false sign rate (lfsr). Finally, we employed the "pROC" R package [15] to calculate the receiver operating characteristic (ROC) curve and area under the ROC curve with the significance measures (p-values for mcLMM and METASOFT, lfsr for `mash`) and the correct labels of null effects and non-null effects.

2.7 Analysis of the GTEx dataset

The Genotype-Tissue Expression (GTEx) v8 dataset [5] was used in this study. We downloaded the gene expression data, the summary statistics of single-tissue cis-eQTL data using a 1 MB window around each gene, and the covariates in the eQTL analysis from GTEx portal (`https://gtexportal.org/home/datasets`). The subject-level genotypes were acquired from dbGaP accession number phs000424.v8.p2. The GTEx v8 dataset includes 49 tissues from 838 donors. We selected 15,627 genes that were expressed in all 49 tissues. We only included SNPs with minor allele frequency (MAF) greater than 1% and missing rate lower than 5%. We applied `mash` and mcLMM plus METASOFT to the GTEx v8 dataset in our analysis.

Since `mash` requires observation of the correlation structure among non-significant tests and data-driven covariance matrices before fitting its model, we prepared its input by selecting the top SNP with the smallest p-value and 49 random SNPs (or all other SNPs if there were fewer than 49 SNPs left in a gene) in every gene from the eQTL analysis in the GTEx v8 dataset. There were 560,475 gene-SNP pairs in total. `mash` uses the estimated effect sizes and standard errors of these gene-SNP pairs to learn the correlation structure of different conditions/tissues. We used the top significant SNPs to set up the data-driven covariances. We then fit `mash` to the random set of gene-SNP pairs with the canonical and data-driven covariances. With the fitted `mash` model, we computed the posterior summaries including local false sign rate (lfsr) [18] for the selected gene-SNP pairs to estimate the significance. We defined significant gene-SNP pairs as those with lfsr < 0.05 in any tissues.

We applied mcLMM to the same set of gene-SNP pairs. We regressed out unwanted confounding factors in gene expression levels for each tissue with a linear model using covariates provided by GTEx. Covariates of each sample included top 5 genotyping principal components, PEER factors [17] (15 factors for tissues with fewer than 150 samples, 30 factors for those with 150-250 samples, 45 factors for those with 250-350 samples, and 60 factors for those with more than 350 samples), sequencing platform, and sex. We ran mcLMM with the genotypes and processed gene expression levels of all 838 individuals across 49 GTEx tissues for each gene-SNP pair. Missing values in gene expression were included in the mcLMM input. The effect sizes, standard errors, and correlation matrices estimated by mcLMM were meta-analyzed with METASOFT to evaluate the significance under both the fixed effects (FE) and random effects (RE2) models. The resulting p-values were converted to q-values [19] to control false discovery rates. A gene-SNP pair was considered significant if its false discovery rate (FDR) was smaller than 5%.

2.8 Analysis of the UK Biobank dataset

This work was conducted using the UK Biobank Resource under application 33127. Samples were filtered for Caucasian individuals (Data-Field 22006)). Hard imputed genotype data from the UK Biobank were LD pruned using a window size of 50, step size of 1, and correlation threshold of 0.2. SNPs were further filtered for minor allele frequency of at least 0.01 and a Hardy-Weinberg equilibrium p-value greater than 1e-7 using Plink 2 [4]. Samples were filtered for unrelated individuals with KING using a cutoff value of 0.125 [11]. Genotype data were split by chromosome and converted to bigsnpr format (v 1.4.4) for memory efficiency [12].

The following data fields were retrieved: age at recruitment (Data-Field 31), sex (Data-Field 21022), BMI (Data-Field 23104), body fat percentage (Data-Field 23099), 10 genetic principal components (Data-Field 22009), HDL Cholesterol (Data-Field 30760), LDL Direct (Data-Field 30780), Apolipoprotein A (Data-Field 30630), Apolipoprotein B (Data-Field 30640), and Triglycerides (Data-Field 30870). Continuous phenotypes were visually inspected and triglycerides were log-transformed due to skewness. Data were filtered for complete observations. All fields were scaled to unit variance and centered at 0.

HDL cholesterol, LDL cholesterol, Apolipoprotein A, Apolipoprotein B, and triglycerides were combined as response variables in the LMM and age, sex, BMI, body fat percentage, and the top 10 genetic principal components were used as additional covariates in the model. Each SNP was marginally fit with mcLMM. The coefficients output by this model for each phenotype were meta-analyzed to calculate FE p-values using METASOFT as packaged with Meta-Tissue v 0.5. The top GWAS hits for five different chromosomes (one per chromosome) were validated using the NHGRI-EBI GWAS catalog [2] and compared to studies for LDL and HDL cholesterol (GCST008035 and GCST008037).

3 Results

3.1 mcLMM is computationally efficient

To demonstrate the efficiency of mcLMM compared to existing approaches, we applied our method to simulated data of varying sample sizes and number of contexts. For these simulations, we simulated a sampling rate of 0.5, which indicates that only half of all possible individual-context pairs of observations are expected to be sampled.

We first applied our method to simulations with a fixed number of 50 contexts and varied the sample size from 100 to 500. From these experiments, we observed that mcLMM requires computational time orders of magnitude less than EMMA and GEMMA. Similarly, when we fixed the number of samples at 500 and varied the context sizes from 4 to 64, we observed dramatically reduced runtimes for mcLMM.

In these experiments, mcLMM also significantly reduces the memory footprint compared to EMMA and GEMMA, since we avoid creating any nt by nt matrices. In these simulations, existing approaches quickly grow memory requirements, with usages that grow to dozens of gigabytes for modestly sized datasets in the thousands of samples. mcLMM allows large-scale studies to be performed on relatively little computational resources (Figure 1).

In cases where there is no missing data, mcLMM allows for further speedups. We ran similar simulations to compare mcLMM with no missing data (optimal model) and mcLMM with missing data (iterative model). We observed a dramatic speedup, with sample sizes of 500,000 individuals across 10 contexts completed in under 10 seconds for the optimal model compared to around 15 minutes for the iterative model.

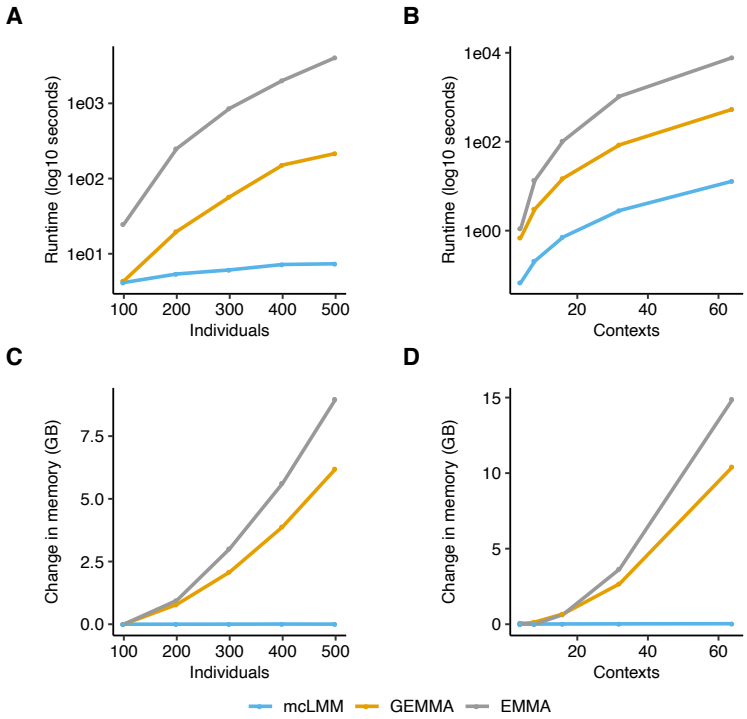


Figure 1 Resource requirements of mcLMM, GEMMA, and EMMA across various simulated individual and context sizes with missing values (sampling rate of 0.5). For varying individuals, contexts were fixed at 50. For varying contexts, individuals were fixed at 500. (A-B) Runtime with log10(seconds) on the y-axis and number of individuals or contexts simulated on the x-axis. (C-D) Memory usage (GB) on the y-axis and number of individuals or contexts simulated on the x-axis.

3.2 mcLMM enables powerful meta analyses to detect eQTLs

We utilized mcLMM to reduce the computational resource requirements of the Meta-Tissue pipeline, which fits a multiple-context LMM and combines the resulting effect sizes using METASOFT [20]. While powerful, the existing approach utilizes EMMA to fit the LMM. For a recent release from the GTEx consortium [5], each pair of genes and single nucleotide polymorphisms (SNPs) required over two hours to run. Across hundreds of thousands of gene-SNP pairs, this method would require years of computational runtime to complete. Utilizing mcLMM, we were able to complete this analysis in 3 days parallelized over each chromosome.

We compared our approach to a method known as `mash` [22]. This approach utilizes effect sizes estimated within each context independently and employs a Bayesian approach to combine their results for meta-analysis. In order to estimate the power of these methods, we performed simulations as described in the methods. In null simulations, we observed well-controlled false positive rates at $\alpha = 0.05$ for mcLMM coupled with METASOFT. In our simulation with true positives, we observed an increased area under the receiver operating characteristic (AUROC) for mcLMM coupled with the random effects (RE2) METASOFT model compared to `mash` (Figure 2).

Next, we compared the number of significant associations identified in the GTEx dataset. The `mash` approach utilized gene-SNP effect sizes estimated by the GTEx consortium within each tissue independently. Concordant with our simulations, we observed that the Meta-

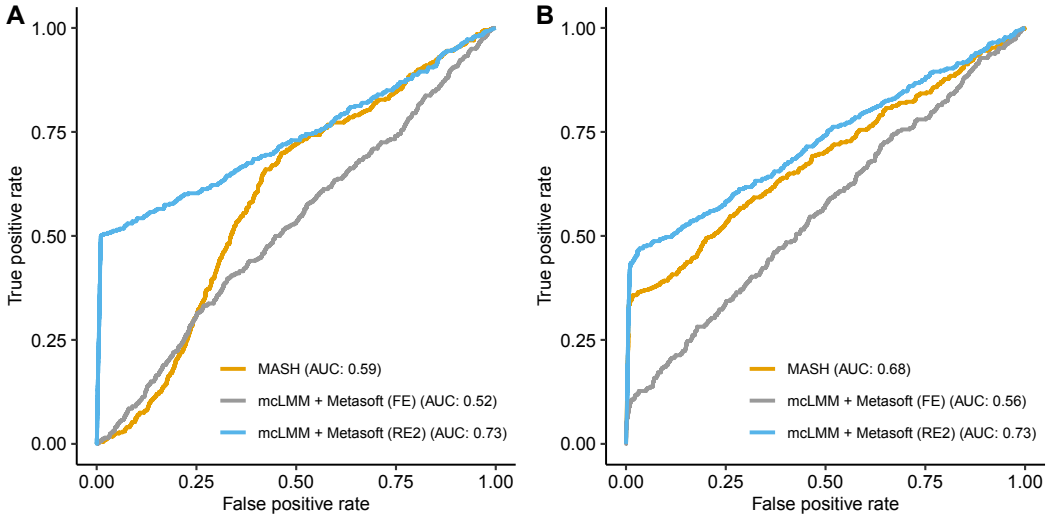


Figure 2 AUROC curves of mcLMM+METASOFT (fixed effects and random effects models) and `mash` in simulated data, assuming the effects of gene-SNP pairs are (A) shared and unstructured, and (B) shared and structured.

Tissue approach, utilizing mcLMM for vast speedup, identified more significant eQTLs than `mash` (Figure 3). These associations allow researchers to better understand the link between genetic variation and complex phenotypes through possible mediation of gene expression.

3.3 mcLMM scales to millions of samples across related phenotypes

As a practical application of the efficiency of mcLMM, we performed a multiple phenotype GWAS in the UK Biobank. A multiple phenotype GWAS associates SNPs with several related phenotypes in order to increase the effective sample size for greater power, under the assumption that the phenotypes are significantly correlated. For our analysis, we combined HDL and LDL cholesterol, Apolipoprotein A and B, and triglyceride levels across 323,266 unrelated caucasian individuals in the UK Biobank. In total, 1,616,330 observations of these related phenotypes were fit as responses in the LMM.

The mcLMM approach completed this analysis over 211,642 SNPs with an additional 14 covariates, parallelized over each chromosome, within a day. Each chromosome was analyzed on a single core machine with 32 GB of memory, with each test taking around 2 seconds to complete. We identified several significant loci, a subset of which replicate previous findings for specific phenotypes included in the model, such as HDL cholesterol [25] (Figure 4). Existing approaches, namely EMMA and GEMMA, require orders of magnitude more memory to begin this analyses and could not be run on the available computational resources.

4 Discussion

We presented mcLMM, an efficient method for fitting LMMs used for multiple-context association studies. Our method provides exact results and scales linearly in time and memory with respect to sample size, while existing methods are cubic. This efficiency allows mcLMM to process hundreds of thousands of samples over several contexts within a day on minimal computational resources, as we showed in simulation and in the UK Biobank. The

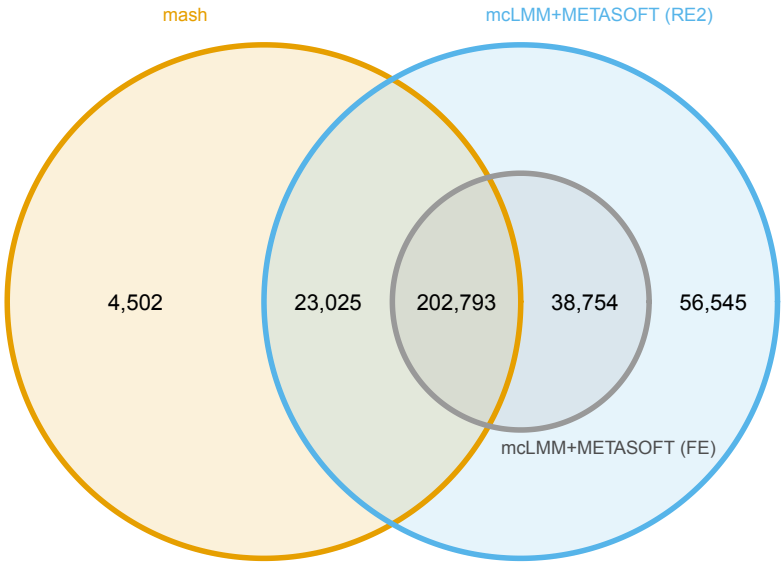


Figure 3 Venn diagram of significant eQTLs identified by meta-analysis methods in the GTEx dataset. We compared mcLMM using the fixed effects (FE) and random effects (RE2) models in METASOFT to `mash`. Note that areas are not proportional to the number of eQTLs in each region. mcLMM+METASOFT (RE2) identified a total of 321,117 significant associations that contained 225,818 eQTLs identified by `mash`.

association parameters learned by mcLMM can further be utilized with the METASOFT framework to provide powerful meta-analysis of the associations, as we showed in the GTEx dataset.

Previous approaches have derived related speedups for LMMs when the matrix K is low rank, such as in the case when multiple samples are genetically identical or clustered in genome wide association studies as described in FaST-LMM [10]. In this approach, the authors show that the likelihood function can be evaluated in linear time with respect to the number of individuals after singular value decomposition of a matrix that is also linear with respect to the number of individuals. Other work has similarly used block structures and Kronecker refactorizations in studies with structured designs, such as multi-trait GWAS, to significantly speed up these approaches as well [9, 13].

Our approach builds upon these findings and we optimize the method specifically for the low rank matrix with known eigenvalues described in the model, thus avoiding any spectral or singular value decompositions. Furthermore, when there is no missing data, our method can compute the optimal model parameters with a closed form solution requiring no iterative optimization of likelihood functions. We also note that mcLMM models covariance across contexts within an individual while the FaST-LMM approach, described above, models covariance across individuals within each context. This specific model fit by mcLMM arises in multiple-context association studies, such as the approach employed by Meta Tissue [20] for

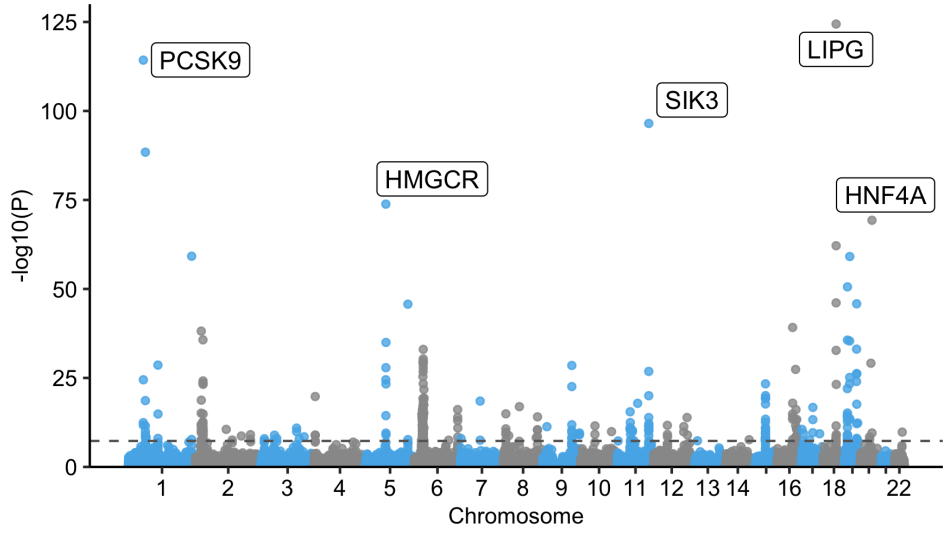


Figure 4 Multiple phenotype GWAS results from UK Biobank. Five phenotypes (LDL cholesterol, HDL cholesterol, Apolipoprotein A, Apolipoprotein B, and triglyceride levels) were used as responses in the mcLMM framework. The model was fit with 1,616,330 observations from 323,266 unrelated Caucasian individuals. In total, 211,642 SNPs were tested with an additional 14 covariates. Each test required around 2 seconds to run on a 32GB machine and was parallelized over each chromosome. The -log10 of the p-values are plot on the y-axis and genomic positions on the x-axis. The horizontal dashed line indicates the genome wide significance level at $p = 0.05/1e6$. The top hit for 5 different chromosomes is annotated with the gene containing the SNP. These genes have been previously identified as associated with a subset of these phenotypes.

identifying eQTLs across tissues utilizing the cubic EMMA algorithm. Applied within this framework for eQTL and multi-trait genome wide association studies, our method provides exact results and scales to hundreds of thousands of samples with minimal computational resources.

References

1 François Aguet, Andrew A. Brown, Stephane E. Castel, Joe R. Davis, Yuan He, Brian Jo, Pejman Mohammadi, YoSon Park, Princy Parsana, Ayellet V. Segrè, Benjamin J. Strober, Zachary Zappala, Beryl B. Cummings, Ellen T. Gelfand, Kane Hadley, Katherine H. Huang, Monkol Lek, Xiao Li, Jared L. Nedzel, Duyen Y. Nguyen, Michael S. Noble, Timothy J. Sullivan, Taru Tukiainen, Daniel G. MacArthur, Gad Getz, Anjene Addington, Ping Guan, Susan Koester, A. Roger Little, Nicole C. Lockhart, Helen M. Moore, Abhi Rao, Jeffery P. Struewing, Simona Volpi, Lori E. Brigham, Richard Hasz, Marcus Hunter, Christopher Johns, Mark Johnson, Gene Kopen, William F. Leinweber, John T. Lonsdale, Alisa McDonald, Bernadette Mestichelli, Kevin Myer, Bryan Roe, Michael Salvatore, Saboor Shad, Jeffrey A. Thomas, Gary Walters, Michael Washington, Joseph Wheeler, Jason Bridge, Barbara A. Foster, Bryan M. Gillard, Ellen Karasik, Rachna Kumar, Mark Miklos, Michael T. Moser, Scott D. Jewell, Robert G. Montroy, Daniel C. Rohrer, Dana Valley, Deborah C. Mash, David A. Davis, Leslie Sobin, Mary E. Barcus, Philip A. Branton, Nathan S. Abell, Brunilda Balliu, Olivier Delaneau, Laure Frésard, Eric R. Gamazon, Diego Garrido-Martín, Ariel D. H. Gewirtz, Genna

Gliner, Michael J. Gloudemans, Buhm Han, Amy Z. He, Farhad Hormozdiari, Xin Li, Boxiang Liu, Eun Yong Kang, Ian C. McDowell, Halit Ongen, John J. Palowitch, Christine B. Peterson, Gerald Quon, Stephan Ripke, Ashis Saha, Andrey A. Shabalin, Tyler C. Shimko, Jae Hoon Sul, Nicole A. Teran, Emily K. Tsang, Hailei Zhang, Yi-Hui Zhou, Carlos D. Bustamante, Nancy J. Cox, Roderic Guigó, Manolis Kellis, Mark I. McCarthy, Donald F. Conrad, Eleazar Eskin, Gen Li, Andrew B. Nobel, Chiara Sabatti, Barbara E. Stranger, Xiaoquan Wen, Fred A. Wright, Kristin G. Ardlie, Emmanouil T. Dermitzakis, Tuuli Lappalainen, Robert E. Handsaker, Seva Kashin, Konrad J. Karczewski, Duyen T. Nguyen, Casandra A. Trowbridge, Ruth Barshir, Omer Basha, Alexis Battle, Gireesh K. Bogu, Andrew Brown, Christopher D. Brown, Lin S. Chen, Colby Chiang, Farhan N. Damani, Barbara E. Engelhardt, Pedro G. Ferreira, Ariel D.H. Gewirtz, Roderic Guigo, Ira M. Hall, Cedric Howald, Hae Kyung Im, Eun Yong Kang, Yungil Kim, Sarah Kim-Hellmuth, Serghei Mangul, Jean Monlong, Stephen B. Montgomery, Manuel Muñoz-Aguirre, Anne W. Ndungu, Dan L. Nicolae, Meritxell Oliva, Nikolaos Panousis, Panagiotis Papasaikas, Anthony J. Payne, Jie Quan, Ferran Reverter, Michael Sammeth, Alexandra J. Scott, Reza Sodaei, Matthew Stephens, Sarah Urbut, Martijn van de Bunt, Gao Wang, Hualin S. Xi, Esti Yeger-Lotem, Judith B. Zaugg, Joshua M. Akey, Daniel Bates, Joanne Chan, Melina Claussnitzer, Kathryn Demanelis, Morgan Diegel, Jennifer A. Doherty, Andrew P. Feinberg, Marian S. Fernando, Jessica Halow, Kasper D. Hansen, Eric Haugen, Peter F. Hickey, Lei Hou, Farzana Jasmine, Ruiqi Jian, Lihua Jiang, Audra Johnson, Rajinder Kaul, Muhammad G. Kibriya, Kristen Lee, Jin Billy Li, Qin Li, Jessica Lin, Shin Lin, Sandra Linder, Caroline Linke, Yaping Liu, Matthew T. Maurano, Benoit Molinie, Jemma Nelson, Fidencio J. Neri, Yongjin Park, Brandon L. Pierce, Nicola J. Rinaldi, Lindsay F. Rizzardi, Richard Sandstrom, Andrew Skol, Kevin S. Smith, Michael P. Snyder, John Stamatoyannopoulos, Hua Tang, Li Wang, Meng Wang, Nicholas Van Wittenberghe, Fan Wu, Rui Zhang, Concepcion R. Nierras, Latarsha J. Carithers, Jimmie B. Vaught, Sarah E. Gould, Nicole C. Lockart, Casey Martin, Anjene M. Addington, Susan E. Koester, GTEx Consortium, Lead analysts:, Data Analysis & Coordinating Center (LDACC): Laboratory, NIH program management:, Biospecimen collection:, Pathology:, eQTL manuscript working group:, Data Analysis & Coordinating Center (LDACC)-Analysis Working Group Laboratory, Statistical Methods groups-Analysis Working Group, Enhancing GTEx (eGTEx) groups, NIH Common Fund, NIH/NCI, NIH/NHGRI, NIH/NIMH, NIH/NIDA, and Biospecimen Collection Source Site-NDRI. Genetic effects on gene expression across human tissues. *Nature*, 550(7675):204–213, October 2017. doi:10.1038/nature24277.

2 Annalisa Buniello, Jacqueline A L MacArthur, Maria Cerezo, Laura W Harris, James Hayhurst, Cinzia Malangone, Aoife McMahon, Joannella Morales, Edward Mountjoy, Elliot Sollis, Daniel Suveges, Olga Vrousgou, Patricia L Whetzel, Ridwan Amode, Jose A Guillen, Harpreet S Riat, Stephen J Trevanion, Peggy Hall, Heather Junkins, Paul Flicek, Tony Burdett, Lucia A Hindorff, Fiona Cunningham, and Helen Parkinson. The NHGRI-EBI GWAS Catalog of published genome-wide association studies, targeted arrays and summary statistics 2019. *Nucleic Acids Research*, 47(D1):D1005–D1012, November 2018. doi:10.1093/nar/gky1120.

3 Clare Bycroft, Colin Freeman, Desislava Petkova, Gavin Band, Lloyd T. Elliott, Kevin Sharp, Allan Motyer, Damjan Vukcevic, Olivier Delaneau, Jared O'Connell, Adrian Cortes, Samantha Welsh, Alan Young, Mark Effingham, Gil McVean, Stephen Leslie, Naomi Allen, Peter Donnelly, and Jonathan Marchini. The uk biobank resource with deep phenotyping and genomic data. *Nature*, 562(7726):203–209, October 2018. doi:10.1038/s41586-018-0579-z.

4 Christopher C Chang, Carson C Chow, Laurent CAM Tellier, Shashaank Vattikuti, Shaun M Purcell, and James J Lee. Second-generation PLINK: rising to the challenge of larger and richer datasets. *GigaScience*, 4(1), February 2015. s13742-015-0047-8. doi:10.1186/s13742-015-0047-8.

5 GTEx Consortium. The GTEx consortium atlas of genetic regulatory effects across human tissues. *Science*, 369(6509):1318–1330, 2020.

6 Buhm Han and Eleazar Eskin. Random-effects model aimed at discovering associations in meta-analysis of genome-wide association studies. *The American Journal of Human Genetics*, 88(5):586–598, May 2011. doi:10.1016/j.ajhg.2011.04.014.

7 Jong Wha J Joo, Eun Yong Kang, Elin Org, Nick Furlotte, Brian Parks, Farhad Hormozdiari, Aldons J Lusis, and Eleazar Eskin. Efficient and Accurate Multiple-Phenotype Regression Method for High Dimensional Data Considering Population Structure. *Genetics*, 204(4):1379–1390, December 2016. doi:10.1534/genetics.116.189712.

8 Hyun Min Kang, Noah A. Zaitlen, Claire M. Wade, Andrew Kirby, David Heckerman, Mark J. Daly, and Eleazar Eskin. Efficient control of population structure in model organism association mapping. *Genetics*, 178(3):1709–1723, 2008. doi:10.1534/genetics.107.080101.

9 Arthur Korte, Bjarni J. Vilhjálmsson, Vincent Segura, Alexander Platt, Quan Long, and Magnus Nordborg. A mixed-model approach for genome-wide association studies of correlated traits in structured populations. *Nature Genetics*, 44(9):1066–1071, September 2012. doi:10.1038/ng.2376.

10 Christoph Lippert, Jennifer Listgarten, Ying Liu, Carl M. Kadie, Robert I. Davidson, and David Heckerman. Fast linear mixed models for genome-wide association studies. *Nature Methods*, 8(10):833–835, October 2011. doi:10.1038/nmeth.1681.

11 Ani Manichaikul, Josyf C. Mychaleckyj, Stephen S. Rich, Kathy Daly, Michèle Sale, and Wei-Min Chen. Robust relationship inference in genome-wide association studies. *Bioinformatics*, 26(22):2867–2873, October 2010. doi:10.1093/bioinformatics/btq559.

12 Florian Privé, Hugues Aschard, Andrey Ziyatdinov, and Michael G B Blum. Efficient analysis of large-scale genome-wide data with two R packages: bigstatsr and bigsnpr. *Bioinformatics*, 34(16):2781–2787, March 2018. doi:10.1093/bioinformatics/bty185.

13 Barbara Rakitsch, Christoph Lippert, Karsten Borgwardt, and Oliver Stegle. It is all in the noise: Efficient multi-task gaussian process inference with structured residuals. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/59c33016884a62116be975a9bb8257e3-Paper.pdf.

14 Vardhman K. Rakyan, Thomas A. Down, David J. Balding, and Stephan Beck. Epigenome-wide association studies for common human diseases. *Nature reviews. Genetics*, 12(8):529–541, July 2011. 21747404[pmid]. doi:10.1038/nrg3000.

15 Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez, and Markus Müller. pROC: an open-source package for R and s+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12:77, 2011.

16 Jack Sherman and Winifred J. Morrison. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950. doi:10.1214/aoms/1177729893.

17 Oliver Stegle, Leopold Parts, Matias Piipari, John Winn, and Richard Durbin. Using probabilistic estimation of expression residuals (PEER) to obtain increased power and interpretability of gene expression analyses. *Nat. Protoc.*, 7(3):500–507, 2012.

18 Matthew Stephens. False discovery rates: a new deal. *Biostatistics*, 18(2):275–294, 2017.

19 John D. Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):479–498, 2002. doi:10.1111/1467-9868.00346.

20 Jae Hoon Sul, Buhm Han, Chun Ye, Ted Choi, and Eleazar Eskin. Effectively identifying eqtls from multiple tissues by combining mixed model and meta-analytic approaches. *PLOS Genetics*, 9(6):1–13, June 2013. doi:10.1371/journal.pgen.1003491.

21 The All of Us Research Program Investigators. The "all of us" research program. *New England Journal of Medicine*, 381(7):668–676, 2019. PMID: 31412182. doi:10.1056/NEJMsr1809937.

22 Sarah M. Urbut, Gao Wang, Peter Carbonetto, and Matthew Stephens. Flexible statistical methods for estimating and testing effects in genomic studies with multiple conditions. *Nature Genetics*, 51(1):187–195, January 2019. doi:10.1038/s41588-018-0268-8.

23 Peter M. Visscher, Matthew A. Brown, Mark I. McCarthy, and Jian Yang. Five years of gwas discovery. *American journal of human genetics*, 90(1):7–24, January 2012. 22243964[pmid]. doi:10.1016/j.ajhg.2011.11.029.

24 S. J. Welham and R. Thompson. Likelihood ratio tests for fixed model terms using residual maximum likelihood. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(3):701–714, 1997. doi:10.1111/1467-9868.00092.

25 Genevieve L. Wojcik, Mariaelisa Graff, Katherine K. Nishimura, Ran Tao, Jeffrey Haessler, Christopher R. Gignoux, Heather M. Highland, Yesha M. Patel, Elena P. Sorokin, Christy L. Avery, Gillian M. Belbin, Stephanie A. Bien, Iona Cheng, Sinead Cullina, Chani J. Hodonsky, Yao Hu, Laura M. Huckins, Janina Jeff, Anne E. Justice, Jonathan M. Kocarnik, Unhee Lim, Bridget M. Lin, Yingchang Lu, Sarah C. Nelson, Sung-Shim L. Park, Hannah Poisner, Michael H. Preuss, Melissa A. Richard, Claudia Schurmann, Veronica W. Setiawan, Alexandra Sockell, Karan Vahi, Marie Verbanck, Abhishek Vishnu, Ryan W. Walker, Kristin L. Young, Niha Zubair, Victor Acuña-Alonso, Jose Luis Ambite, Kathleen C. Barnes, Eric Boerwinkle, Erwin P. Bottinger, Carlos D. Bustamante, Christian Caberto, Samuel Canizales-Quinteros, Matthew P. Conomos, Ewa Deelman, Ron Do, Kimberly Doheny, Lindsay Fernández-Rhodes, Myriam Fornage, Benyam Hailu, Gerardo Heiss, Brenna M. Henn, Lucia A. Hindorff, Rebecca D. Jackson, Cecelia A. Laurie, Cathy C. Laurie, Yuqing Li, Dan-Yu Lin, Andres Moreno-Estrada, Girish Nadkarni, Paul J. Norman, Loreall C. Pooler, Alexander P. Reiner, Jane Romm, Chiara Sabatti, Karla Sandoval, Xin Sheng, Eli A. Stahl, Daniel O. Stram, Timothy A. Thornton, Christina L. Wassel, Lynne R. Wilkens, Cheryl A. Winkler, Sachi Yoneyama, Steven Buyske, Christopher A. Haiman, Charles Kooperberg, Loic Le Marchand, Ruth J. F. Loos, Tara C. Matise, Kari E. North, Ulrike Peters, Eimear E. Kenny, and Christopher S. Carlson. Genetic analyses of diverse populations improves discovery for complex traits. *Nature*, 570(7762):514–518, June 2019. doi:10.1038/s41586-019-1310-4.

26 Xiang Zhou and Matthew Stephens. Genome-wide efficient mixed-model analysis for association studies. *Nature Genetics*, 44(7):821–4, 2012. doi:10.1038/ng.2310.

LRBinner: Binning Long Reads in Metagenomics Datasets

Anuradha Wickramarachchi
School of Computing, Australian National University, Canberra, Australia

Yu Lin[1]
School of Computing, Australian National University, Canberra, Australia

Abstract

Advancements in metagenomics sequencing allow the study of microbial communities directly from their environments. Metagenomics binning is a key step in the species characterisation of microbial communities. Next-generation sequencing reads are usually assembled into contigs for metagenomics binning mainly due to the limited information within short reads. Third-generation sequencing provides much longer reads that have lengths similar to the contigs assembled from short reads. However, existing contig-binning tools cannot be directly applied on long reads due to the absence of coverage information and the presence of high error rates. The few existing long-read binning tools either use only composition or use composition and coverage information separately. This may ignore bins that correspond to low-abundance species or erroneously split bins that correspond to species with non-uniform coverages. Here we present a reference-free binning approach, LRBinner, that combines composition and coverage information of complete long-read datasets. LRBinner also uses a distance-histogram-based clustering algorithm to extract clusters with varying sizes. The experimental results on both simulated and real datasets show that LRBinner achieves the best binning accuracy against the baselines. Moreover, we show that binning reads using LRBinner prior to assembly reduces computational resources for assembly while attaining satisfactory assembly qualities.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Computational genomics

Keywords and phrases Metagenomics binning, long reads, machine learning, clustering

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.11

Supplementary Material *Software (Source Code)*: `https://www.github.com/anuradhawick/LRBinner`; archived at `swh:1:dir:a2988a1c26fc1323580378ff00135030f1ee61a4`

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments. Furthermore, this research was undertaken with the assistance of resources and services from the National Computational Infrastructure (NCI Australia), an NCRIS enabled capability supported by the Australian Government.

1 Introduction

Metagenomics binning is an important area of study in metagenomics analysis. Broadly, metagenomics enables the study of microbial genetic material directly from the source environment [3]. This eliminates the necessity of lab culturing thus revealing the microbial content of an environment as it is without culturing biases. Metagenomics *binning* is one key problem in metagenomics study that facilitates the clustering of sequences into different taxonomic groups. Mainly there are two approaches to address this problem; (1) reference-based binning and (2) reference-free binning. Reference-based binning tools (*e.g.*, Kraken [27], Centrifuge [8] and Kaiju [15]) bin sequences based on similarity by comparing with a database of known reference genomes and thus face challenges when the reference database

[1] To whom correspondence should be addressed.

is unavailable or incomplete. At present, reference-free binning tools have been gaining popularity over reference-based binning tools, especially in discovering novel or rare species in complex metagenomics datasets. While Next-Generation Sequencing (NGS) technologies produce short reads, existing reference-free binning tools typically rely on longer *contigs* that are assembled from short reads and contain richer information for binning. Reference-free binning tools (*e.g.*, MetaBAT [6, 7], MaxBin [29, 28], BMC3C [30], BusyBeeWeb [12, 11], SolidBin [24] and VAMB [18], *etc.*) bin contigs based on their composition and coverage information, without using any reference database. For example, a recent work VAMB [18] introduced the use of deep variational auto-encoders to perform reference-free unsupervised binning of contigs incorporating both the composition and coverage information. VAMB then uses an iterative medoid clustering algorithm which extracts clusters (bins) in a local search fashion. Thanks to the accurate composition and coverage information of contigs, reference-free approaches show promising results in binning contigs from metagenomics assemblies.

With the advent of Third Generation Sequencing (TGS) technologies such as Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), reads obtained are much longer than NGS reads (>10kbp). Therefore, more information becomes available in the reads themselves to support direct reads binning. However, contig-binning tools cannot be directly applied to bin long reads (by treating them as contigs) because there is no coverage information available for each long read. Moreover, while certain contig-binning tools make use of single-copy marker genes to estimate the number of bins in the sample, the high error rates of long reads and the varying coverages of different species make it infeasible to derive accurate estimations. Furthermore, datasets containing raw long reads are much larger in size compared to typical datasets containing contigs, and hence, demand more scalable reference-free binning tools.

Recently, a long-read binning tool MetaBCC-LR [26] was introduced to bin error-prone long reads. While MetaBCC-LR shows very promising results in binning long reads, it still suffers from accuracy and scalability issues, especially in complex metagenomics datasets. Firstly, MetaBCC-LR uses the composition and coverage information of long reads in a separate manner. This can result in the ignorance of bins for species with low abundance and incorrect bin split for species with non-uniform composition or coverage. Secondly, due to its scalability issue, MetaBCC-LR has to employ a sampling strategy to work on a subset of reads for large datasets, which affects its overall binning accuracy. In addition, binning of long read datasets requires novel algorithms to detect clusters of vastly varying sizes (hundreds to millions of reads per species), which is different from the contig-binning scenarios (few hundreds of contigs per species [14]). Therefore, it is persistently demanding better approaches to bin massive long-read datasets accurately and efficiently. The requirement is further supported by the advent of PacBio HiFi technology [25] which produces accurate and massive long-read datasets in metagenomics studies.

In this paper, we present LRBinner to bin TGS long reads without using any reference databases. LRBinner combines the composition and coverage features and eliminates the need to sub-sample large datasets. More specifically, LRBinner uses a variational auto-encoder to obtain lower dimensional representations by incorporating both composition and coverage information of the complete dataset. LRBinner further uses a distance-histogram-based clustering algorithm that can capture confident clusters of varying sizes. LRBinner finally assigns unclustered reads to obtained clusters using their statistical profiles. The experimental results of LRBinner compared against other baselines show that LRBinner achieves better binning results on both simulated and real datasets. Moreover, we show that binning long reads by LRBinner prior to assembly helps to improve genome fraction of assemblies while reducing the memory consumption for metagenomics assembly.

2 Methods

LRBinner consists of three main steps; (1) learning lower dimensional latent representations of composition and coverage, (2) clustering the latent representations and (3) obtaining complete clusters. The complete workflow for LRBinner is demonstrated in Figure 1. In the following sections, we will explain these three steps in details.

2.1 Step 1

LRBinner uses two typical binning features of metagenomic sequences, composition and coverage. The composition and coverage of each long read is represented as trimer frequency vectors and k-mer coverage histograms [26], respectively.

2.1.1 Computing Composition Vectors

Previous studies show that different species demonstrate unique genomic patterns [1, 4] and thus can be used in composition-based metagenomics binning. Oligonucleotide frequency vectors are one such genomic representation that can be used in metagenomics binning. Small k-mer sizes (k varying from 3-7) have been used in the past to discriminate assembled contigs of different origins [2, 7, 11, 20, 28] and 3-mers have been used in metagenomics binning of error-prone long reads [26] which shows that trinucleotide frequency vectors provide stable binning despite the noise level exist in TGS reads. Therefore in LRBinner, we utilise k=3 by default which results in trinucleotide composition vectors. For each long read, we count the frequencies of all 64 3-mers in this read and merge the reverse complements to form a vector of 32 dimensions. The resulting vector is then normalised by the total number of 3-mers observed in the read. We refer to this trimer frequency vector as V^T.

2.1.2 Computing Coverage Vectors

While an all-vs-all alignment of long reads may provide coverage information for each long read, it is usually too time-consuming to perform such alignments on large, long-read datasets (requires indexing hundreds of thousands of reads, index searching of read minimizers and pairwise-alignment of reads to filter false positives). Given a sufficiently large k, the frequency of a k-mer is defined as the number of occurrences of this k-mer in the entire dataset. Long reads from high-abundance species tend to contain k-mers with higher frequencies compared to long reads from low-abundance species. Hence, a k-mer frequency vector can be computed for each long read to represent coverage information without performing alignments [26] to represent read coverage. In order to obtain such coverage histograms, we first compute the k-mer counts of all long reads in the entire dataset by DSK [21] (the default value of k=15). The counts are then indexed in memory by encoding each nucleotide in 2 bits as per the encoding (*i.e.*, A=00, C=01, T=10 and G=11) [21]. The resulting index is in the form $(k_i, coverage(k_i))$ (as *key*, *value* pairs), where $coverage(k_i)$ is the number of occurrences of the k-mer k_i in the entire dataset. Now for each k-mer k_i of a read, we obtain the frequency from the index. These frequencies are then used to build a normalised histogram, V^C. We chose a preset bin width (*bin_width*) for the histogram and obtain a vector of *bins* dimensions. By default we set *bin_width*=10 and *bins*=32. All the k-mers with counts exceeding the histogram limits are added into the last index of the histogram. We also normalise the histogram by the total number of k-mers observed in the read.

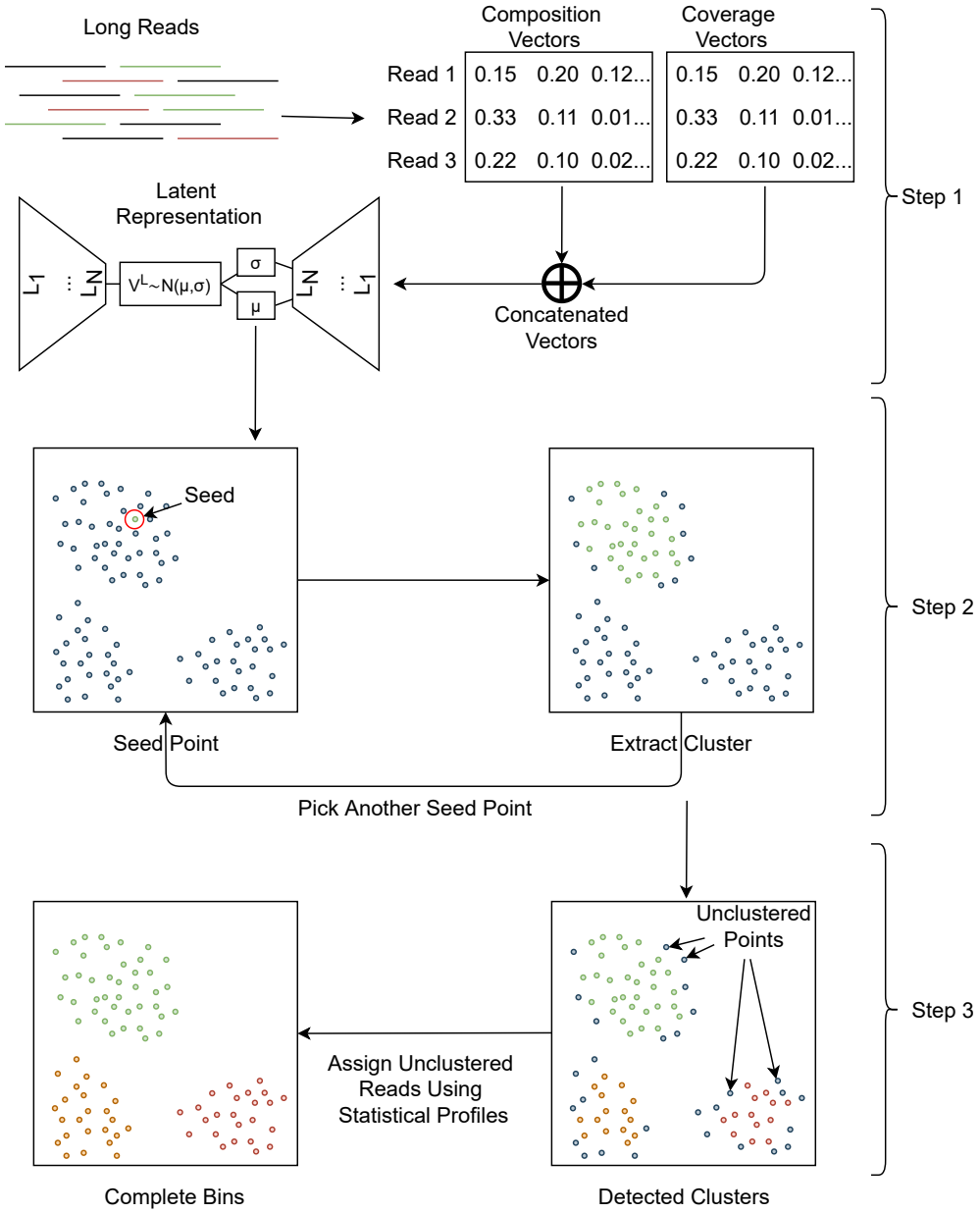


Figure 1 Overall workflow of LRBinner. (Step 1) The feature vectors of composition and coverage information are computed from long reads. The feature vectors are fed into a variational auto-encoder to obtain low-dimensional latent representations. (Step 2) Sample a seed point (read) in the latent space and derive a confident cluster (bin) that contains this seed point. Step 2 is iterated until there is no seed point. (Step 3) The unclustered points are assigned to the clusters using a statistical model. Note that the 2-dimensional representation of points is only for the illustration purpose.

2.1.3 Obtaining Latent Representations

For each long read, its coverage (V^C) and composition (V^T) vectors are concatenated to form a single vector V of 64 dimensions. We use a variational auto-encoder to obtain lower dimensional latent representations. The key motivation for using a variational auto-encoder is to combine coverage and composition features effectively. Previous work shows that using TSNE on concatenated coverage and composition reduced the effectiveness [26]. This is mainly because TSNE do not attempt to learn meaningful weights for each feature, but rather consider neighbourhoods using spatial distances. However, the variational auto-encoder learns lower dimensional representations by meaningfully weighting features through a deep neural network such that original data can be reconstructed from the decoding layers.

Our implementation of the variational auto-encoder consists of two hidden-layers in the encoder and decoder. Each layer uses batch normalisation and dropout with p=0.1 during the training phase. For each input vector V, the auto-encoder learns a latent representation V_i^L, where $V_i^L \sim \mathcal{N}(\mu_i, \sigma_i)$. The latent representation consists of 8 dimensions. Each layer in the encoder and decoder contains 128 neurons. Similar to previous studies [18], we use *LeakyRELU* (leaky rectified linear unit function) for μ and *softplus* function for σ layers. Note that μ and σ represents neural network layers intended to learn the lower dimensional means and standard deviations of each read's distribution. We use the weighted sum of reconstruction error E (equation 1) and Kullback–Leibler divergence [10, 18] D_{KL} (equation 2) as the loss function. E_{cov} and E_{com} represent reconstruction errors of coverage and composition respectively. Equation 3 demonstrates the complete loss function used.

$$E = \sum (V_{in} - V_{out})^2 \tag{1}$$

$$D_{KL}(latent|prior) = -\sum \frac{1}{2}(1 + ln(\sigma) - \mu^2 - \sigma) \tag{2}$$

$$Total\ Loss = w_{cov}E_{cov} + w_{com}E_{com} + w_{kld}D_{KL} \tag{3}$$

Here we set w_{cov}=0.1, w_{com}=1 and w_{kld}=1/500 as determined empirically using simulated data. The decoder output was obtained through LeakyRELU activation in order to reconstruct the scaled positive inputs. We train the auto-encoder with read batches of size 10,240 for 200 epochs. Finally, we obtain the predicted latent means of the input data from the encoder for clustering. Each point in the latent mean corresponds to the relevant read in the original input.

2.2 Step 2

In this step, we perform clustering of the latent means learnt by the variational auto-encoder. The complete clustering algorithm of LRBinner is illustrated in Figure 2. Similar to previous studies [18], we use the cosine distance as the distance measure for clustering. Note that cosine distance between point a and b in latent space V^L is defined as $d(a,b)=\frac{V_a^L \cdot V_b^L}{||V_a^L|| ||V_b^L||}$. Given a point a, a distance histogram H_a can be generated by computing the pairwise distances between a and all other points and setting the bin width as Δ (Δ=0.005 in our experiments). We define *peak* as the index of the first maximal of the distance histogram H_a. Similarly, the *valley* is defined as the index of the first minimal after the *peak* in the distance histogram H_a. Refer to the top right figure in Figure 2 for an example of the *peak* and *valley* in a distance histogram.

As shown in VAMB [18], a point with smaller valley-to-peak ratio $H[valley]/H[peak]$ is more likely to be the medoid of a cluster, where $H[valley]$ and $H[peak]$ are the number of points corresponding to the *valley* and *peak* in the distance histogram H, respectively. Therefore, VAMB randomly samples points, searches within a distance of 0.05 (up to 25 neighbouring points) and moves to another point if $H[valley]/H[peak]$ can be further reduced. This step is iterated until a local minimal point of $H[valley]/H[peak]$ is inferred as a proper cluster medoid and then the corresponding cluster is extracted by removing points within a distance $\Delta \times valley$ of the distance histogram. However, clusters of long reads are orders of magnitude larger than clusters of contigs, thus mere local search of a cluster medoid may be inefficient. Furthermore, while most contig clusters consist of hundreds of points per species[14], the long-read clusters vary in size drastically (from hundreds of points to millions of points), which demand for a more flexible search strategy rather than sampling points within a fixed radius and up to a fixed number of neighbours. Hence, we design the following strategy to dynamically extract clusters of varying sizes. Our algorithm consists of two steps; (1) from a seed point to a candidate cluster and (2) from a candidate cluster to a confident cluster.

2.2.1 From a Seed Point to a Candidate Cluster

A point s is called a *seed point* if its valley-to-peak ratio $H_s[valley]/H_s[peak] < 0.5$ in its distance histogram H_s. Initially, LRBinner randomly picks a seed point s from the entire dataset and obtains its distance histogram H_s. Note that a distance histogram demonstrates a *candidate cluster*. This *candidate cluster* consists of the points within the distance $\Delta \times valley$ in H_s, referred to as *candidate cluster points*. Compared to the seed point, some candidate cluster points may have lower valley-to-peak ratio that result in more confident clusters. However, the number of candidate cluster points may vary significantly depending on the size of the ground-truth clusters. In the next section, we will show how to use sampling strategies to find a confident cluster from a candidate cluster.

2.2.2 From a Candidate Cluster to a Confident Cluster

Given a *candidate cluster*, we sample 10% of candidate cluster points (up to 1,000 points) to compare their corresponding distance histograms. For each point p in *candidate cluster points*, we compute the valley-to-peak ratio $H_p[valley]/H_p[peak]$ in its corresponding distance histogram H_p. We chose a point x from the sample with the minimum $H[valley]/H[peak]$ value and extract a *confident cluster* which consists of points within a distance $\Delta \times valley$ of the distance histogram H_x. In contrast with the iterative medoid search in VAMB [18], this approach takes advantage of the rough estimation of the *candidate cluster* from a seed point and thus allows us to dynamically and efficiently discover clusters with varying sizes. This process is iterated until no further *candidate clusters* or *seed* points are observed. Please refer to Section 5 for detailed information. The resulting clusters are depicted as detected clusters in Figure 1. Note that few reads still remain unclustered due to the noise present in composition and coverage vectors of error-prone long reads and we will show how to assign them to existing bins in the next section.

2.3 Step 3

2.3.1 Obtaining Final Bins

Once all the clusters have been yielded, the points that are sparsely located are left aside. However, such points could have the potential to improve the downstream assembly processes.

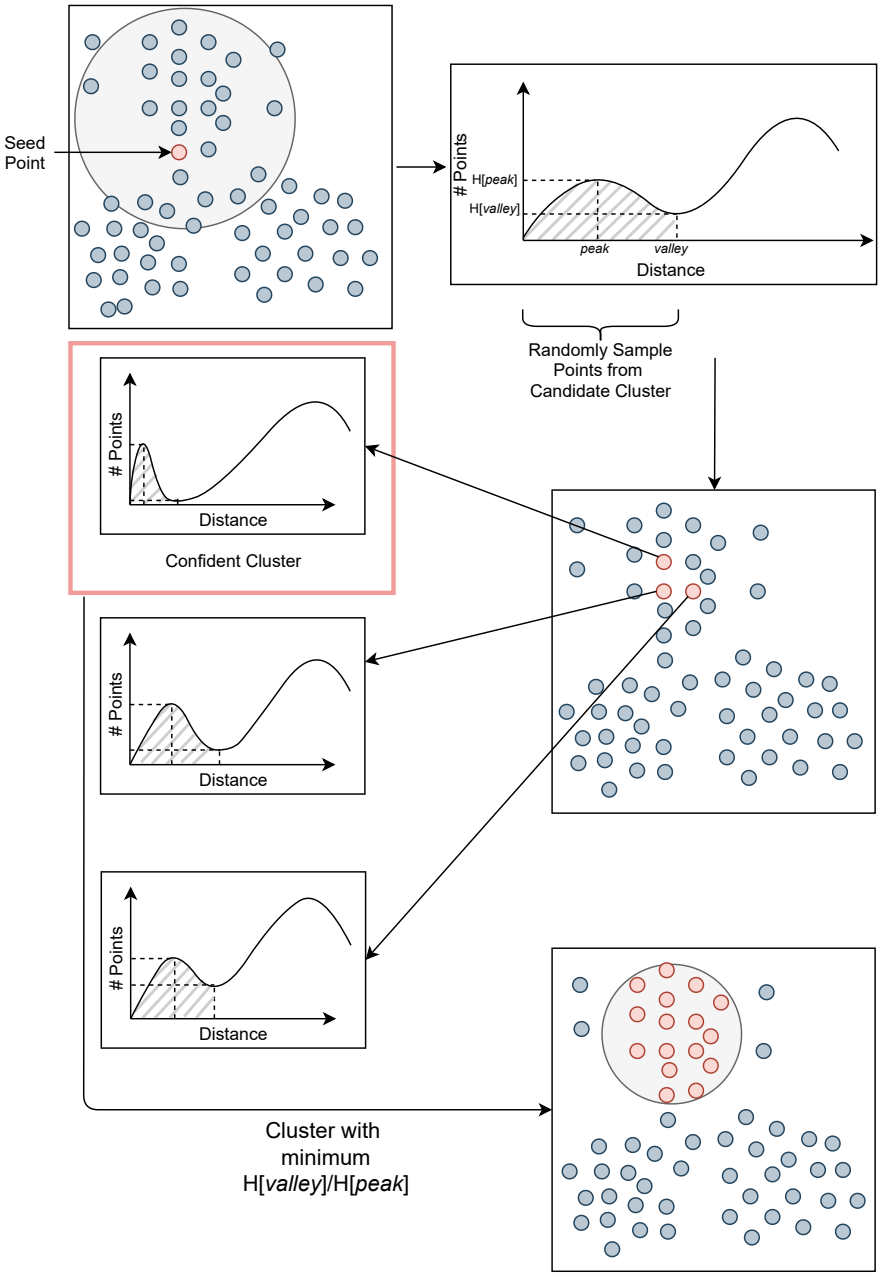


Figure 2 Illustration of the clustering algorithm. First select a seed point, generate its distance histogram and derive a *candidate cluster*. Sample from the *candidate cluster points* and choose a point with the minimum valley-to-peak ratio. Extract points before the *valley* to form a *confident cluster*. Note that the 2D representation of points is only for the illustration purposes.

Hence, we assign such points to the detected clusters using a statistical model similar to MetaBCC-LR [26]. For each cluster C_k the mean μ_k^C, μ_k^T and standard deviation σ_k^C, σ_k^T is computed using the coverage and composition vectors; V^C and V^T respectively.

$$PDF(\bar{v}, \bar{\mu}, \bar{\sigma}) = \prod_{j}^{|\bar{v}|} \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}} \tag{4}$$

Finally the unclustered reads are assigned to the cluster C_i using a maximum likelihood computed using equation 4. The assignment of reads is performed such that equation 5 is maximised. V_l^C and V_l^T are the coverage histogram and trimer frequency vectors of the unclustered read l.

$$C_i = \underset{i}{\operatorname{argmax}} \left\{ PDF(V_l^C, \mu_i^C, \sigma_i^C) \times PDF(V_l^T, \mu_i^T, \sigma_i^T) \right\} \tag{5}$$

3 Experimental Setup

3.1 Datasets

We evaluated LRBinner using several simulated and real datasets containing long reads. Detailed information about the datasets and constituent species are tabulated under Tables 3 and 4 of Appendix.

3.1.1 Simulated Datasets

We simulated two datasets using SimLoRD [23] to evaluate the performance of our method. The first dataset consists of 8 species and the second dataset consists of 20 species. These datasets are named as **Sim-8** and **Sim-20** respectively. We set the average read length to be 5,000bp with default error model of SimLoRD (insertion probability=0.11, deletion probability=0.04 and substitution probability=0.01).

3.1.2 Real Datasets

In order to evaluate LRBinner, we use several real datasets with known ground-truth references. To determine the origins of the reads in these datasets, the reads were mapped to the respective reference species using Minimap2[13]. The information about the datasets are as follows.

- Reads from ATCC MSA-1003 Mock Microbial Community with PacBio CCS reads from NCBI BioProject number *PRJNA546278* (**MSA-1003**). For the evaluation we used the top 10 species which have more than 1% abundance.
- PacBio-HiFi reads obtained from NCBI BioProject number *PRJNA680590*. There are 3 read samples (NCBI BioSample number *SAMN16885726*) and each sample consists of 21 strains for 17 species as follows;
  - **SRX9569057**: Standard input library
  - **SRX9569058**: Low input library
  - **SRX9569059**: Ultra low input library (PCR amplified sample)

Detailed information about the simulated datasets is available in Section A of Appendix.

3.2 Tools for Benchmarking

There is a limited number of tools that support binning of long reads. Remind that most contig-binning tools cannot be directly applied to bin long reads (even for highly accurate PacBio HiFi reads) because there is no coverage information available for each long read. Hence, in our evaluation we use BusyBeeWeb [11] and MetaBCC-LR [26] which supports error prone long-reads as input. However, BusyBeeWeb only supports up to 200MB of FASTA formatted data. Hence, in our evaluation we have to provide BusyBeeWeb with a sub-sampled set of reads and evaluated the binning precision and recall on this sub-sampled set.

3.3 Evaluation Criteria

In our evaluation we report precision (equation 6), recall (equation 7) and F1-score (equation 8) of binning. We transform the binning result to a matrix a of size $K \times S$, where K denotes the number of bins and S denotes the number of species. Note that a_{ks} denotes the number of reads assigned to bin k with ground truth species s. In order to evaluate the quality of binning, we used AMBER [16] to obtain the completeness (defined as $\frac{true\ positives_b}{true\ positives_b + false\ negatives_b}$ for each bin b) and contamination (defined as $1 - \frac{true\ positives_b}{true\ positives_b + false\ positives_b}$ for each bin b). Please note that we only compare AMBER results of MetaBCC-LR and LRBinner as BusyBeeWeb does not bin the entire datasets due to limited input size. Furthermore, we assemble the reads before and after binning using LRBinner. Metagenomics assemblies were performed using wtdbg2 [22] and metaFlye [9]. We compare genome fractions, CPU-time and memory usage in assembly evaluation. We used MetaQUAST [17] to obtain the genome fraction (average percentage of bases aligned per reference genome) for the qualitative evaluation of assembled contigs.

$$Precision = \frac{\sum_k max_s\{a_{ks}\}}{\sum_k \sum_s a_{ks}} \tag{6}$$

$$Recall = \frac{\sum_s max_k\{a_{ks}\}}{\sum_k \sum_s a_{ks}} \tag{7}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{8}$$

4 Results and Discussion

We first compare precision, recall, F1 score and the estimated number of bins for binning performance. We further present the completeness and contamination results of bins produced by different binners. We finally evaluate assembly results using genome fraction and recorded the resource utilisation for the chosen assembly tools.

4.1 Binning Results

We benchmarked the binning performance for BusyBeeWeb, MetaBCC-LR and LRBinner. Table 1 demonstrates the binning results in terms of precision, recall, F1-score and the number of inferred bins. While BusyBeeWeb, MetaBCC-LR and LRBinner perform in a comparable fashion on simulated datasets, LRBinner achieves the best estimation on the number of bins with respect to the ground truth. As BusyBeeWeb has a limitation of input

Table 1 Comparison of binning results of BusyBeeWeb, MetaBCC-LR and LRBinner.

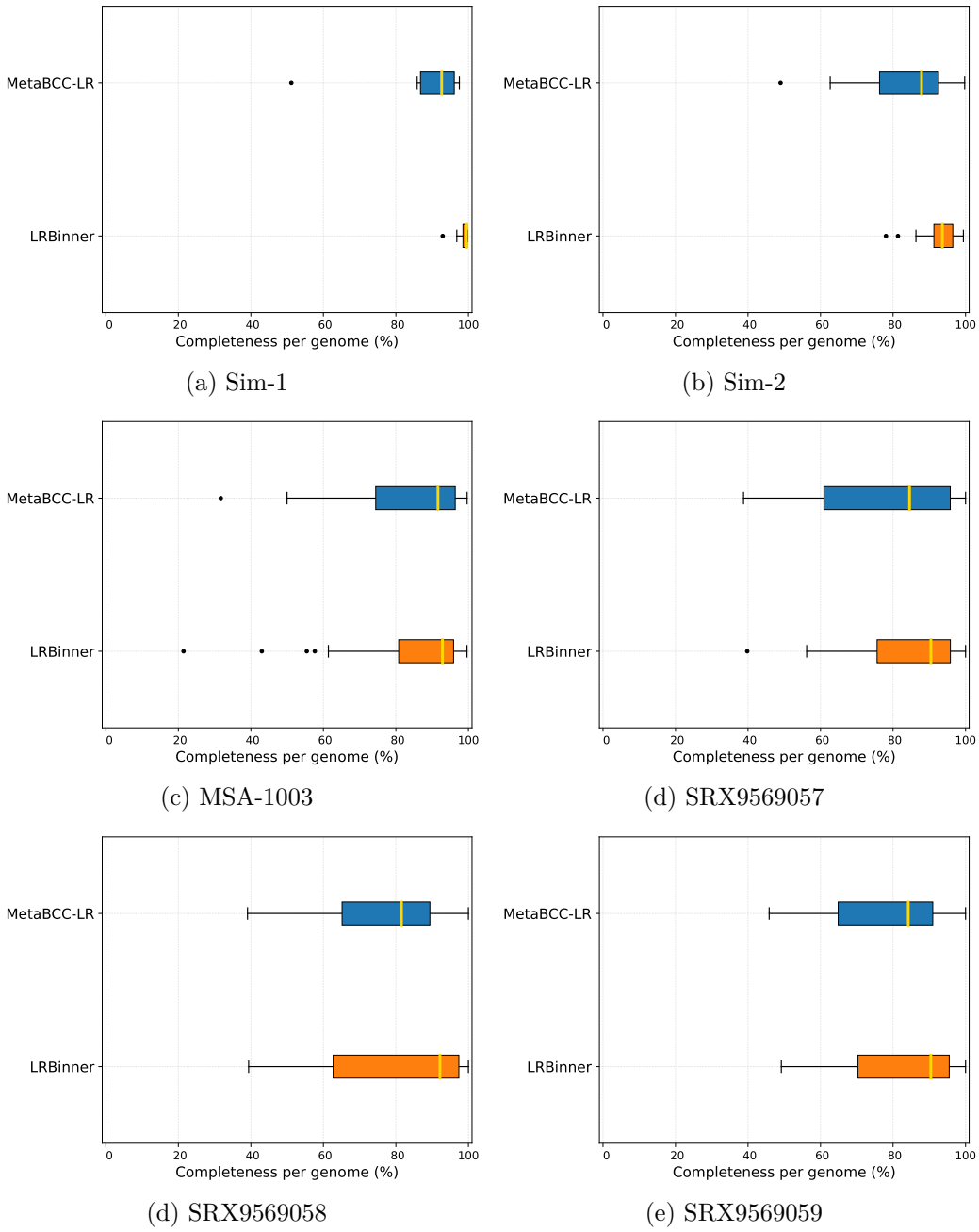
Dataset	Actual No. of Bins	Evaluation Criteria	BusyBeeWeb	MetaBCC-LR	LRBinner
Sim-8	8	Precision	90.41%	90.78%	**99.14%**
		Recall	**99.80%**	96.18%	99.14%
		F1 score	94.87%	93.40 %	**99.14%**
		Bins Detected	50	13	**8**
Sim-20	20	Precision	**95.88%**	82.97%	90.53%
		Recall	**97.99%**	81.95%	88.23%
		F1 score	**96.92%**	82.46%	89.36%
		Bins Detected	30	29	**18**
MSA-1003	10	Precision	68.30%	93.69%	**95.30%**
		Recall	81.96%	95.50%	**95.99%**
		F1 score	74.51%	94.59%	**95.64%**
		Bins Detected	87	14	**10**
SRX9569057	17	Precision	48.63%	**80.94**	80.47%
		Recall	72.68%	85.82	**90.68%**
		F1 score	58.27%	83.31	**85.27%**
		Bins Detected	111	23	**16**
SRX9569058	17	Precision	23.01%	70.18%	**73.72%**
		Recall	32.64%	86.63%	**91.03%**
		F1 score	26.99%	77.54%	**81.46%**
		Bins Detected	117	37	**22**
SRX9569059	17	Precision	65.70%	66.69%	**79.70%**
		Recall	**95.36%**	73.76%	91.25%
		F1 score	77.80%	70.05%	**85.08%**
		Bins Detected	124	**16**	20

data size (200Mb), its binning accuracy deteriorates on the real large datasets due to its limited access to the complete dataset. Note that LRBinner improves binning results for all the real datasets as indicated by the higher F1 scores.

Figure 3 illustrates the completeness of bins produced by MetaBCC-LR and LRBinner. Note that BusyBeeWeb is not included in this comparison as it cannot handle the entire dataset in most cases. LRBinner has been able to produce bins with better average completeness over MetaBCC-LR. Figure 4 also illustrates the contamination levels of bins produced by MetaBCC-LR and LRBinner. From the plots it is evident that LRBinner produces bins with lower contamination in all datasets except for **SRX9569059**. Note that the dataset **SRX9569059** has been generated from a PCR amplified sample leading to a significant deviation from the original sample abundances in contrast with **SRX9569057** and **SRX9569058** datasets. For example, in **SRX9569059**, the abundance of *Faecalibacterium prausnitzii* drops from $\sim 16\%$ to $\sim 8\%$ whereas the abundance of *Fusobacterium nucleatum* surges from $\sim 4\%$ to $\sim 7\%$, which may result in contamination of long reads in binning results.

4.2 Assembly Results

We assembled the reads binned by LRBinner to evaluate the potential assembly quality changes. For the assembly, we chose the two state-of-the-art long-read assemblers wtdbg2 [22] and metaFlye [9]. Table 2, demonstrates that binning long reads prior to assembly by LRBinner improves the genome fraction for all wtdbg2 assemblies (up to 40%) and maintains comparable genome fractions for metaFlye assemblies. This is not surprising as metaFlye is



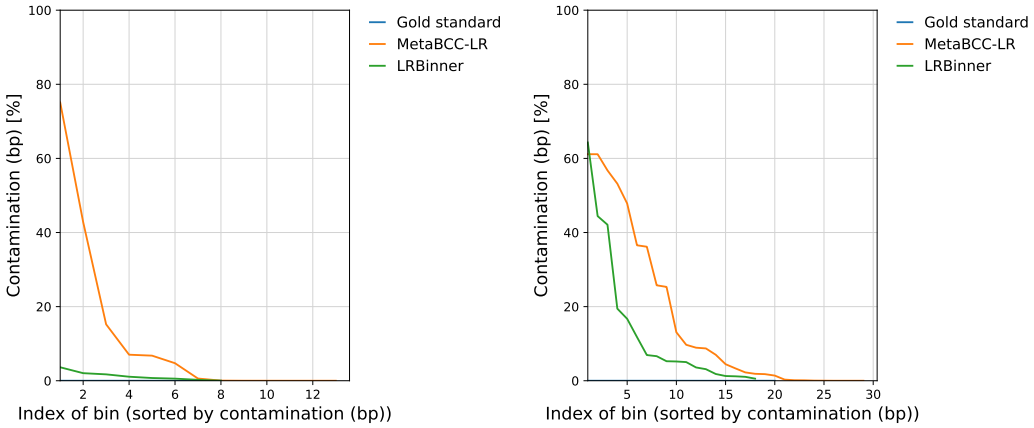
(a) Sim-1 (b) Sim-2

(c) MSA-1003 (d) SRX9569057

(d) SRX9569058 (e) SRX9569059

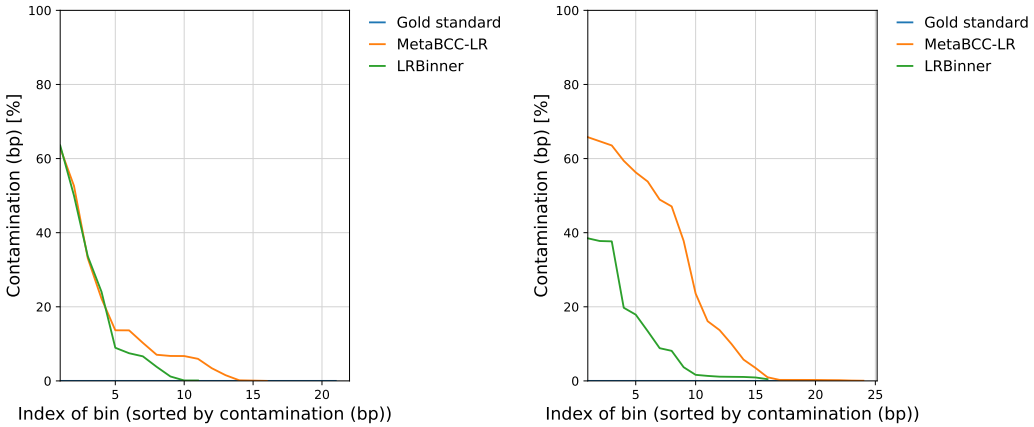
Figure 3 Comparison of bin completeness between MetaBCC-LR and LRBinner.

a metagenomics specialised assembler in contrast with wtdbg2. For example, in the datasets SRX9569057, SRX9569058 and SRX9569059, binning via LRBinner enabled wtdbg2 to recover low-abundance species which were ignored in the assembly of the entire raw dataset, *e.g., Methanobrevibacter smithii* (from 0 to 96%), *Saccharomyces cerevisiae* (from 0 to 75%) and *Candida albican* (from 0 to 70%). This is because LRBinner allows wtdbg2 to estimate more appropriate parameters in each bin rather than applying the same parameters across the entire dataset.



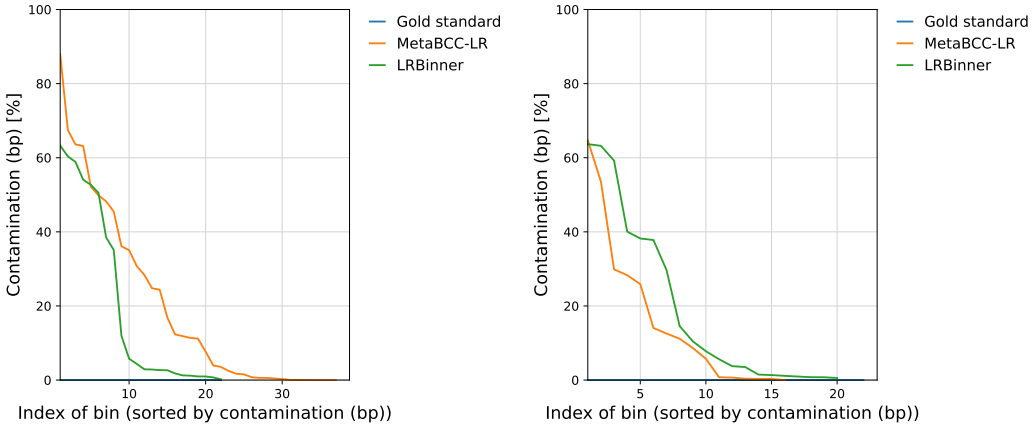
(a) Sim-1

(b) Sim-2



(c) MSA-1003

(d) Dataset SRX9569057



(d) Dataset SRX9569058

(e) Dataset SRX9569059

Figure 4 Comparison of bin contamination between MetaBCC-LR and LRBinner.

Another advantage of binning prior to assembly is the reduction of the computing resources for assembly. As demonstrated in Table 2, the peak-memory usage has been drastically reduced in both wtdbg2 (upto 10×) and metaFlye (upto 4×) assemblies. Note that the CPU time is comparable as binning long reads may not lead to significant reduction of k-mer indexing cost and the construction and simplification of assembly graphs.

Table 2 Comparison of assembled genome fractions, CPU time consumed for assembly and peak memory usage of assembly before and after binning the reads.

Dataset	Assembly Tool	Genome Fraction		CPU Hours		Peak Memory (GB)	
		Raw	Binned	Raw	Binned	Raw	Binned
Sim-8	wtdbg2	98.80%	**98.90%**	**0.26**	0.84	9.28	**0.96**
	metaFlye	**99.90%**	99.85%	16.13	**11.64**	44.12	**10.65**
Sim-20	wtdbg2	97.84%	**99.19%**	**0.16**	2.28	10.60	**0.92**
	metaFlye	**99.80%**	99.75%	**19.44**	20.28	44.70	**11.23**
MSA-1003	wtdbg2	67.45%	**82.50%**	**0.31**	1.05	23.43	**19.61**
	metaFlye	91.40%	**91.74%**	**155.96**	158.59	62.28	**45.38**
SRX9569057	wtdbg2	40.40%	**73.02%**	**0.26**	1.56	21.72	**3.88**
	metaFlye	**77.73%**	73.68%	122.00	**116.20**	57.91	**26.31**
SRX9569058	wtdbg2	37.51%	**80.65%**	**0.30**	1.98	30.79	**3.86**
	metaFlye	79.16%	**79.63%**	**211.61**	212.58	87.62	**41.37**
SRX9569059	wtdbg2	41.00%	**80.38%**	**0.26**	1.82	25.63	**3.80**
	metaFlye	**79.69**	77.46%	152.64	**129.41**	62.62	**30.56**

5 Implementation

In order to restrict the iterative search for clusters, we use early termination parameters in our algorithm. We stop drawing seed points when the remaining number of reads reaches below *min_cluster_size* (=5000 by default) or the number of iterations has passed *max_iterations* (=1000 by default). We executed MetaBCC-LR and LRBinner on 5 sub-sampled datasets from **Sim-8**. We set MetaBCC-LR to skip the sampling step to make a fair comparison with LRBinner. Figure 5 illustrates the variation of time with increasing dataset size. It is evident that LRBinner scales well whereas time consumption of MetaBCC-LR grows much rapidly. Note that we do not consider BusyBeeWeb for this evaluation as it does not bin complete datasets.

LRBinner was implemented using C++ and Python version 3.7. The deep learning component is implemented using PyTorch [19] and Numpy [5]. We conducted our assemblies on NCI Australia with 2 x 24-core Intel Xeon Platinum 8274 (Cascade Lake) 3.2 GHz CPUs 192GB RAM and binning on Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 16GB RAM (4 core 8 threads) running Ubuntu Linux. We used 28 cores (with 56 threads hyper-threading) for assembly and 8 threads for binning.

6 Conclusion

In this paper, we presented LRBinner, a long read binner capable of binning error-prone long reads using both coverage and composition information. Our work extends the use of variational auto-encoders to combine raw features and learn a better latent representation for long-read binning. Furthermore, we presented a novel clustering strategy that can

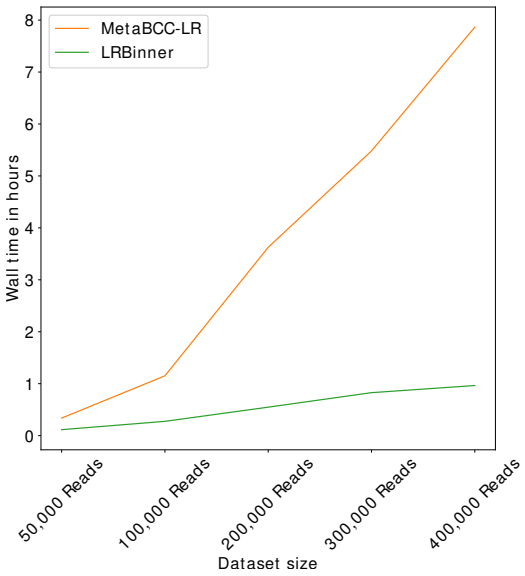


Figure 5 Wall time used by MetaBCC-LR and LRBinner without any sampling of data.

perform clustering on large datasets with varying cluster sizes. Performance of LRBinner was evaluated against existing long-read binners using simulated and real datasets. Our experimental results show that LRBinner outperforms state-of-the-art long-read binning tools and also improves resource usage of downstream assembly.

One limitation of LRBinner is the inability to distinct shared reads that arise shared genomic regions between different species. Resolution of such regions demands more experiments and significant improvements to the methodology as future work. Furthermore, we intend to introduce better noise handling to the clustering algorithm and investigate the potential of combining the binning and assembly of long reads simultaneously.

References

1 Takashi Abe, Shigehiko Kanaya, Makoto Kinouchi, Yuta Ichiba, Tokio Kozuki, and Toshimichi Ikemura. Informatics for unveiling hidden genome signatures. *Genome Research*, 13(4):693–702, 2003. URL: `http://genome.cshlp.org/content/13/4/693.full.pdf+html`.

2 Johannes Alneberg, Brynjar Smári Bjarnason, Ino de Bruijn, et al. Binning metagenomic contigs by coverage and composition. *Nature Methods*, 11:1144, September 2014.

3 Kevin Chen and Lior Pachter. Bioinformatics for Whole-Genome Shotgun Sequencing of Microbial Communities. *PLOS Computational Biology*, 1(2), July 2005.

4 P J Deschavanne, A Giron, J Vilain, G Fagot, and B Fertil. Genomic signature: characterization and classification of species assessed by chaos game representation of sequences. *Molecular Biology and Evolution*, 16(10):1391–1399, October 1999. URL: `http://oup.prod.sis.lan/mbe/article-pdf/16/10/1391/9592103/mbe1391.pdf`.

5 Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. `doi:10.1038/s41586-020-2649-2`.

6 Dongwan D. Kang, Jeff Froula, Rob Egan, and Zhong Wang. MetaBAT, an efficient tool for accurately reconstructing single genomes from complex microbial communities. *PeerJ*, 3:e1165, 2015.

7 Dongwan D. Kang, Feng Li, Edward Kirton, Ashleigh Thomas, et al. MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. *PeerJ*, 7:e7359, 2019.

8 Daehwan Kim, Li Song, Florian P. Breitwieser, and Steven L. Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research*, 26(12):1721–1729, 2016. URL: `http://genome.cshlp.org/content/26/12/1721.full.pdf+html`.

9 Mikhail Kolmogorov, Derek M. Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy P. L. Smith, and Pavel A. Pevzner. metaflye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, November 2020. `doi:10.1038/s41592-020-00971-x`.

10 S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. `doi:10.1214/aoms/1177729694`.

11 Cedric C. Laczny, Christina Kiefer, Valentina Galata, et al. BusyBee Web: metagenomic data analysis by bootstrapped supervised binning and annotation. *Nucleic Acids Research*, 45(W1):W171–W179, May 2017. URL: `http://oup.prod.sis.lan/nar/article-pdf/45/W1/W171/18137403/gkx348.pdf`.

12 Cedric C. Laczny, Nicolás Pinel, Nikos Vlassis, and Paul Wilmes. Alignment-free Visualization of Metagenomic Data by Nonlinear Dimension Reduction. *Scientific Reports*, 4:4516, March 2014.

13 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, May 2018. URL: `http://oup.prod.sis.lan/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf`.

14 Hsin-Hung Lin and Yu-Chieh Liao. Accurate binning of metagenomic contigs via automated clustering sequences using information of genomic signatures and marker genes. *Scientific reports*, 6:24175–24175, April 2016. 27067514[pmid]. `doi:10.1038/srep24175`.

15 Peter Menzel, Kim Lee Ng, and Anders Krogh. Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nature Communications*, 7:11257, April 2016. Article.

16 Fernando Meyer, Peter Hofmann, Peter Belmann, Ruben Garrido-Oter, Adrian Fritz, Alexander Sczyrba, and Alice C McHardy. AMBER: Assessment of Metagenome BinnERs. *GigaScience*, 7(6), June 2018. giy069. `doi:10.1093/gigascience/giy069`.

17 Alla Mikheenko, Vladislav Saveliev, and Alexey Gurevich. MetaQUAST: evaluation of metagenome assemblies. *Bioinformatics*, 32(7):1088–1090, November 2015. URL: `http://oup.prod.sis.lan/bioinformatics/article-pdf/32/7/1088/19568745/btv697.pdf`.

18 Jakob Nybo Nissen, Joachim Johansen, Rosa Lundbye Allesøe, Casper Kaae Sønderby, Jose Juan Almagro Armenteros, Christopher Heje Grønbech, Lars Juhl Jensen, Henrik Bjørn Nielsen, Thomas Nordahl Petersen, Ole Winther, and Simon Rasmussen. Improved metagenome binning and assembly using deep variational autoencoders. *Nature Biotechnology*, January 2021. `doi:10.1038/s41587-020-00777-4`.

19 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

20 David Pellow, Itzik Mizrahi, and Ron Shamir. Plasclass improves plasmid sequence classification. *PLOS Computational Biology*, 16(4):1–9, April 2020. `doi:10.1371/journal.pcbi.1007781`.

21 Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, January 2013. URL: http://oup.prod.sis.lan/bioinformatics/article-pdf/29/5/652/702231/btt020.pdf.

22 Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158, 2020. doi:10.1038/s41592-019-0669-3.

23 Bianca K. Stöcker, Johannes Köster, and Sven Rahmann. SimLoRD: Simulation of Long Read Data. *Bioinformatics*, 32(17):2704–2706, May 2016. URL: http://oup.prod.sis.lan/bioinformatics/article-pdf/32/17/2704/17346032/btw286.pdf.

24 Ziye Wang, Zhengyang Wang, Yang Young Lu, et al. SolidBin: improving metagenome binning with semi-supervised normalized cut. *Bioinformatics*, April 2019. btz253. URL: http://oup.prod.sis.lan/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btz253/28579442/btz253.pdf.

25 Aaron M. Wenger, Paul Peluso, William J. Rowell, Pi-Chuan Chang, Richard J. Hall, Gregory T. Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D. Olson, Armin Töpfer, Michael Alonge, Medhat Mahmoud, Yufeng Qian, Chen-Shan Chin, Adam M. Phillippy, Michael C. Schatz, Gene Myers, Mark A. DePristo, Jue Ruan, Tobias Marschall, Fritz J. Sedlazeck, Justin M. Zook, Heng Li, Sergey Koren, Andrew Carroll, David R. Rank, and Michael W. Hunkapiller. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, 37(10):1155–1162, October 2019. doi:10.1038/s41587-019-0217-9.

26 Anuradha Wickramarachchi, Vijini Mallawaarachchi, Vaibhav Rajan, and Yu Lin. MetaBCC-LR: metagenomics binning by coverage and composition for long reads. *Bioinformatics*, 36(Supplement_1):i3–i11, July 2020. doi:10.1093/bioinformatics/btaa441.

27 Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014.

28 Yu-Wei Wu, Blake A. Simmons, and Steven W. Singer. MaxBin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets. *Bioinformatics*, 32(4):605–607, October 2015.

29 Yu-Wei Wu, Yung-Hsu Tang, Susannah G. Tringe, et al. Maxbin: an automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome*, 2(1):26, 2014.

30 Guoxian Yu, Yuan Jiang, Jun Wang, et al. BMC3C: binning metagenomic contigs using codon usage, sequence composition and read coverage. *Bioinformatics*, 34(24):4172–4179, June 2018. URL: http://oup.prod.sis.lan/bioinformatics/article-pdf/34/24/4172/27088792/bty519.pdf.

A Information About Datasets

Tables 3 and 4 demonstrate the simulated and real dataset information respectively. Note that the Table 3 tabulates the coverages used for simulation using SimLoRD [23].

Table 3 Information of simulated datasets.

Dataset	Number of Reads	Total Size	Species	Coverage
Sim-8	432,333	3.5Gb	Acetobacter pasteurianus	25
			Bacillus cereus	50
			Chlamydophila psittaci	80
			Escherichia coli	125
			Haemophilus parainfluenzae	350
			Lactobacillus casei	200
			Thermococcus sibiricus	150
			Streptomyces scabiei	100
Sim-20	666,735	5.3Gb	Amycolatopsis mediterranei	25
			Arthrobacter arilaitensis	65
			Brachyspira intermedia	20
			Corynebacterium ulcerans	40
			Erysipelothrix rhusiopathiae	55
			Enterococcus faecium	50
			Mycobacterium bovis	80
			Photobacterium profundum	85
			Streptococcus pyogenes	100
			Xanthobacter autotrophicus	150
			Rhizobium leguminosarum	100
			Francisella novicida	150
			Candidatus Pelagibacter ubique	67
			Halobacterium sp	65
			Lactobacillus delbrueckii	60
			Paenibacillus mucilaginosus	90
			Rickettsia prowazekii	100
			Thermoanaerobacter brockii	110
			Yersinia pestis	105
			Nitrosococcus watsonii	95

Table 4 Information of real datasets.

Dataset	Number of Reads	Total Size	Species	Abundance
MSA-1003	2,358,257	19Gb	Acinetobacter baumannii	0.18%
			Bacillus pacificus	1.80%
			Bacteroides vulgatus	0.02%
			Bifidobacterium adolescentis	0.02%
			Clostridium beijerinckii	1.80%
			Cutibacterium acnes	0.18%
			Deinococcus radiodurans	0.02%
			Enterococcus faecalis	0.02%
			Escherichia coli	18.0%
			Helicobacter pylori	0.18%
			Lactobacillus gasseri	0.18%
			Neisseria meningitidis	0.18%
			Porphyromonas gingivalis	18.0%
			Pseudomonas aeruginosa	1.80%
			Rhodobacter sphaeroides	18.0%
			Schaalia odontolytica	0.02%
			Staphylococcus aureus	1.80%
			Staphylococcus epidermidis	18.0%
			Streptococcus agalactiae	1.80%
			Streptococcus mutans	18.0%
SRX9569057	1,978,852	17Gb	Faecalibacterium prausnitzii	14.82%
			Veillonella rogosae	20.01%
			Roseburia hominis	12.47%
			Bacteroides fragilis	8.36%
			Prevotella corporis	6.28%
			Bifidobacterium adolescentis	8.86%
			Fusobacterium nucleatum	7.56%
			Lactobacillus fermentum	9.71%
			Clostridioides difficile	1.10%
			Akkermansia muciniphila	1.62%
SRX9569058	2,770,833	25Gb	Methanobrevibacter smithii	0.17%
SRX9569059	2,480,208	20Gb	Salmonella enterica	0.0065%
			Enterococcus faecalis	0.0011%
			Clostridium perfringens	0.00009%
			Escherichia coli (JM109)	1.83%
			Escherichia coli (B-3008)	1.82%
			Escherichia coli (B-2207)	1.65%
			Escherichia coli (B-766)	1.66%
			Escherichia coli (B-1109)	1.77%
			Candida albicans	0.16%
			Saccharomyces cerevisiae	0.16%

Compression of Multiple k-Mer Sets by Iterative SPSS Decomposition

Kazushi Kitaya
Tokyo, Japan

Tetsuo Shibuya
Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan

Abstract

A set of k-mers is used in many bioinformatics tasks, and much work has been done on methods to efficiently represent or compress a single set of k-mers. However, methods for compressing multiple k-mer sets have been less studied in spite of their obvious benefits for researchers and genome-related database maintainers. This paper proposes an algorithm to compress multiple k-mer sets, which works by iteratively splitting SPSS (spectrum-preserving string sets). In experiments with 3292 k-mer sets constructed from E. coli whole-genome sequencing data and 2555 k-mer sets constructed from human RNA-Seq data, the proposed algorithm could reduce the compressed file sizes by 34.7% and 13.2% respectively compared to one of the state-of-the-art colored de Bruijn graph representations. Also, our method used less memory than the colored de Bruijn graph method. This paper also introduces various methods to make the compression algorithm efficient in terms of time and memory, one of which is a parallelizable small-weight SPSS construction algorithm.

2012 ACM Subject Classification Applied computing → Molecular sequence analysis

Keywords and phrases sequencing data, k-mer, de Bruijn graph, compression, colored de Bruijn graph

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.12

Supplementary Material *Software (Source Code)*: `https://github.com/kkty/kmer-sets-compression`; archived at `swh:1:dir:ee89fa1752240bda5cf199275118ba3884b7eac2`

Funding This work was supported by JSPS KAKENHI Grant 17H01693, 20K21827, and JST CREST Grant JPMJCR1402JST. The super-computing resource was provided by Human Genome Center, the Institute of Medical Science, the University of Tokyo.

1 Introduction

With the advent of next-generation sequencers, the cost of obtaining genomic information has decreased dramatically. It enabled a wide range of research, along with the development of public repositories such as Sequence Read Archive [9]. At the same time, there has been a growing need for efficient methods of processing and storing the increasing amount of data.

For fast processing with less memory, many bioinformatics tasks take as input a set of k-mers, or the de Bruijn graph [7] represented by a set of k-mers. For example, the DNA fragment assembly can be done efficiently by finding an Eulerian path in de Bruijn graphs [15]. Also, a set of k-mers can be thought of as a sketch of the original sequence data, so it can be used to index sequence datasets [17].

Much work has been done on techniques to efficiently represent a single k-mer set or a single de Bruijn graph. For example, a succinct de Bruijn graph [4] can be used to efficiently represent a de Bruijn graph, keeping essential queries fast, and we can see its usage in MEGAHIT assembler [11] and other works. When we do not need to support fast query

responses, or just want to efficiently store a set of k-mers to disk, there are several other approaches that can be simple or more efficient in terms of disk usage. For example, unitigs, a set of strings that can be obtained by finding non-branching paths in de Bruijn graphs, can be used to represent a set of k-mers efficiently. Finding non-branching paths in de Bruijn graphs (and merging those paths) are formulated as "compacting de Bruijn graphs" and researches have been done on methods to efficiently compact de Bruijn graphs as in [6]. The concept of unitigs was generalized in [16] which introduced spectrum-preserving string sets (SPSS). Also, the paper showed a heuristic algorithm that can be used to get SPSS from a set of k-mers, which achieves far better compression than unitigs.

In addition to the attempts to efficiently represent a single set of k-mers or a single de Bruijn graph, researchers have looked for efficient representations of multiple sets of k-mers. [8] suggested that "colored de Bruijn graphs", which can represent multiple k-mer sets obtained from multiple datasets and can efficiently respond to some types of queries, were suited for tasks like genotyping. Further studies have been conducted to find more efficient ways to represent colored de Bruijn graphs, one of which is Rainbowfish [2], which combined the succinct de Bruijn graph [4] with succinct color information representation. Mantis [13], which was primarily designed for indexing datasets, can also be thought of as a colored de Bruijn graph, and its color representation was greatly improved recently by the author of the Rainbowfish paper [1].

As we have seen, there has been research on methods to store a single k-mer set, some of which support efficient query responses (e.g., succinct de Bruijn graphs), and some of which focus on simplicity (e.g., unitigs) or better compression (e.g., SPSS). Also, there has been research on methods to efficiently store multiple k-mer sets which support fast query responses (e.g., colored de Bruijn graphs).

However, methods to represent multiple k-mer sets with a primary focus on compression capability are not well studied, in spite of their obvious benefits for researchers who want to work with multiple datasets and for maintainers of genome-related databases who want to store a number of datasets.

This paper tackles this problem by introducing a new algorithm to compress multiple k-mer sets, which works by iteratively splitting SPSS.

With the proposed algorithm, it was possible to compress 3292 k-mer sets constructed from E. coli whole-genome sequencing data and 2555 k-mer sets constructed from human RNA-Seq data to 2.42% and 6.04%, respectively. They were better by 60.0% and 25.0% than compressing them individually, which led to 6.03% and 8.06%. They were also better by 34.7% and 13.2% than one of the recent colored de Bruijn graph representations, Mantis [13, 1], which led to 3.70% and 6.97%. Also, our method used 26.1GB and 24.8GB of memory, better by 22.1% and 9.5% than Mantis, which used 33.5GB and 27.4GB.

This paper also introduces various methods to make the compression algorithm efficient in terms of speed and memory usage, one of which is a small-weight SPSS construction algorithm that performs well with multiple threads. On 16-core machines, the proposed algorithm was faster by 83.2% than the existing non-parallelizable algorithm, UST [16].

2 Preliminaries

This chapter covers the concepts of k-mers, canonical k-mers, de Bruijn graphs, compacted de Bruijn graphs, and spectrum-preserving string sets (SPSS), which will be utilized by the proposed algorithms.

2.1 k-mer and canonical k-mer

A string s is called a k-mer when $|s| = k$. Here, the alphabet Σ is equal to $\{A, C, G, T\}$.

Let s be a k-mer. For each $c \in \Sigma$, $next(s, c)$ is the k-mer that can be constructed by concatenating the suffix of length $k-1$ in s and c. Similarly, for each $c \in \Sigma$, $prev(s, c)$ is the k-mer that can be constructed by concatenating c and the prefix of length $k-1$ in s.

Let s be a k-mer. The reverse complement of s is the k-mer that can be constructed by reversing s and replacing characters A, C, G, and T with T, G, C, and A, respectively.

Let s be a k-mer and s' be the reverse complement of s. The canonical form of s (and the canonical form of s') is $\min(s, s')$. Here, we use the dictionary order of strings.

Let s be a k-mer. s is a canonical k-mer if the canonical form of s is equal to s.

In addition to k-mers, we defined canonical k-mers and other related terms to handle data from double-stranded structures like DNA. Instruments like next-generation sequencers produce a set of strings (reads). We often pre-process the data by listing all the k-mers that appears as a substring in the reads and making a set of canonical k-mers from them.

2.2 de Bruijn graph of canonical k-mers

Let $S = \{s_1, s_2, ..., s_n\}$ be a set of canonical k-mers. The de Bruijn graph G represented by S is an undirected graph with the following properties.

- It has n vertices. Let them be $\{v_1, v_2, ..., v_n\}$.
- Each vertex has two sides: "left" side and "right" side.
- Each vertex is labeled by a k-mer. Let us suppose that for each i, $label(v_i) = s_i$.
- There is an edge from the right side of v_i to the left side of v_j if $c \in \Sigma$ exists such that $next(label(v_i), c)$ is equal to $label(v_j)$.
- There is an edge from the right side of v_i to the right side of v_j if $c \in \Sigma$ exists such that $next(label(v_i), c)$ is equal to the reverse complement of $label(v_j)$.
- There is an edge from the left side of v_i to the right side of v_j if $c \in \Sigma$ exists such that $prev(label(v_i), c)$ is equal to $label(v_j)$.
- There is an edge from the left side of v_i to the left side of v_j if $c \in \Sigma$ exists such that $prev(label(v_i), c)$ is equal to the reverse complement of $label(v_j)$.

A sequence of vertices $p = \{v_1, v_2, ..., v_m\}$ in a de Bruijn graph G is a path in G if the following conditions are satisfied.

- For each $1 \leq i < j \leq m$, $v_i \neq v_j$.
- For each $1 \leq i \leq m-1$, v_i and v_{i+1} are connected by an edge.
- For each $2 \leq i \leq m-1$, if v_{i-1} is connected to the left side of v_i, v_{i+1} is connected to the right side of v_i.
- For each $2 \leq i \leq m-1$, if v_{i-1} is connected to the right side of v_i, v_{i+1} is connected to the left side of v_i.

The string represented by p is the string that will be returned by the following procedure.

1. If $m = 1$, return $label(v_1)$.
2. If v_2 is connected to the right side of v_1, initialize s with $label(v_1)$. Otherwise, initialize s with the reverse complement of $label(v_1)$.
3. For $i = 2, 3, ..., m$, perform the following operations.
   a. If v_{i-1} is connected to the left side of v_i, append the last character of $label(v_i)$ to the end of s. Otherwise, append the last character of the reverse complement of $label(v_i)$ to s.
4. Return s.

Figure 1 shows an example of de Bruijn graphs along with a path in it.

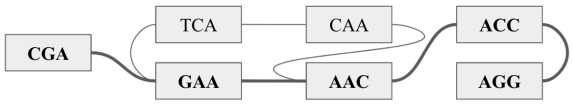


Figure 1 The de Bruijn graph represented by a set of canonical k-mers $\{CGA, TCA, CAA, ACC, GAA, AAC, AGG\}$ and a path in it that represents "CGAACCT".

2.3 Unitigs and compacted de Bruijn graph

A compacted de Bruijn graph is a graph that can be built by "merging" non-branching paths of a de Bruijn graph. In this section, the concept of compacted de Bruijn graphs is introduced along with the concept of unitigs.

Let $G = (V, E)$ be a de Bruijn graph. The unitigs of G are a set of strings that can be constructed with the following procedure.

1. Find a min-size path cover W in G such that each edge in any of its paths does not share a vertex's side with other edges in E.
2. For each path p in W, make the string represented by p.

Note that the path cover is uniquely determined when we do not care about orientations.

Also, note that unitigs can be defined by edge contraction as well. But we define them as above because we are focusing on paths in de Bruijn graphs throughout this paper.

Let $pre(s_i)$ be the k-mer that corresponds to the prefix of s_i whose length is k, and $suf(s_i)$ be the k-mer that corresponds to the suffix of s_i whose length is k.

Let $S = \{s_1, s_2, ..., s_n\}$ be the unitigs of a de Bruijn graph G. The compacted de Bruijn graph G' corresponding to G is an undirected graph with the following properties. It is similar to the definition of a de Bruijn graph.

- It has n vertices. Let them be $\{v_1, v_2, ..., v_n\}$.
- Each vertex has two sides: "left" side and "right" side.
- Each vertex is labeled by a string. Let us suppose that for each i, $label(v_i) = s_i$.
- There is an edge from the right side of v_i to the left side of v_j if $c \in \Sigma$ exists such that $next(suf(label(v_i)), c)$ is equal to $pre(label(v_j))$.
- There is an edge from the right side of v_i to the right side of v_j if $c \in \Sigma$ exists such that $next(suf(label(v_i)), c)$ is equal to the reverse complement of $suf(label(v_j))$.
- There is an edge from the left side of v_i to the right side of v_j if $c \in \Sigma$ exists such that $prev(pre(label(v_i)), c)$ is equal to $suf(label(v_j))$.
- There is an edge from the left side of v_i to the left side of v_j if $c \in \Sigma$ exists such that $prev(pre(label(v_i)), c)$ is equal to the reverse complement of $pre(label(v_j))$.

A path in a compacted de Bruijn graph can be defined the same way as for a de Bruijn graph.

Let $p = \{v_1, v_2, ..., v_m\}$ be a path in a compacted de Bruijn graph. The string represented by p is the string that will be returned by the following procedure.

1. If $m = 1$, return $label(v_1)$.
2. If v_2 is connected to the right side of v_1, initialize s with $label(v_1)$. Otherwise, initialize s with the reverse complement of $label(v_1)$. Here, the reverse complement of a string is defined the same way as for k-mers.

3. For $i = 2, 3, ..., m$, perform the following operation.
   a. If v_{i-1} is connected to the left side of v_i, append the suffix of $label(v_i)$ whose length is $|label(v_i)| - (k-1)$ to the end of s. Otherwise, append the suffix of the reverse complement of $label(v_i)$ whose length is $|label(v_i)| - (k-1)$ to the end of s.
4. Return s.

Figure 2 shows an example of compacted de Bruijn graphs along with a path in it.

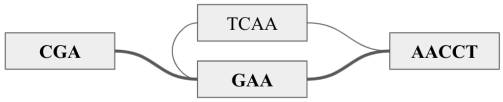


Figure 2 The compacted de Bruijn graph built from the de Bruijn graph illustrated in Figure 1 along with a path in it that represents the string "CGAACCT".

2.4 Spectrum-preserving string sets (SPSS)

In this section, the concept of SPSS (Spectrum-Preserving String Sets) [16] is described.

Let S be a set of canonical k-mers. A set of strings X is SPSS of S if the following conditions are satisfied.

- For each $x \in X$, $|x| \geq k$.
- For each $s \in S$, one of the following is satisfied.
  - There exists $x \in X$ such that s appears as a substring of x.
  - There exists $x \in X$ such that the reverse complement of s appears as a substring of x.
- $\sum_{x \in X} |x| = |S| + (k-1)|X|$

Note that it is possible to reconstruct S from X by finding all the k-mers in X and getting the canonical form of each.

Let $\sum_{x \in X} |x|$ be the weight of SPSS X. As it is possible to reconstruct S from X, making small-weight SPSS corresponds to compressing a k-mer set.

3 Related work

[16] introduced SPSS, a representation of a single k-mer set. In addition, the paper proposed an algorithm to build efficient SPSS from a compacted de Bruijn graph. The algorithm, UST, works by heuristically finding a small-size path cover in a compacted de Bruijn graph and listing up the strings represented by each path. Specifically, it repeats the following procedure. First, a vertex that is not visited is arbitrarily picked. Second, a path from that vertex is constructed using vertices that are not visited. During this step, if multiple choices (of edges) are available, one of them is arbitrarily picked. Also, the path in construction will be merged to a previously-constructed path if possible.

Simplitigs [5] are a concept that is mostly equivalent to SPSS. The paper also proposes a heuristic algorithm like UST for constructing simplitigs from a de Bruijn graph.

Both SPSS and simplitigs are concepts for representing a single k-mer set. Although [5] uses pan-genomes in some of its experiments, the combination of simplitigs (or SPSS) and a technique to exploit similarities in multiple k-mer sets for better compression has been left as a future work.

Mantis [13] is one of the recent colored de Bruijn graph representations. It represents multiple k-mer sets with a CQF-based color table that maps k-mers to color IDs and a RRR-based color class table that maps color IDs to its existences in each k-mer set. Also, color IDs are picked so that a color that appears frequently gets a smaller ID.

Our approach is to use SPSS to compress multiple k-mer sets using the technique called "Iterative SPSS Decomposition". This is a novel technique to use the power of SPSS along with a method to utilize similarities in multiple k-mer sets. This paper includes promising experimental results of the approach with a comparison to Mantis, an existing technique to efficiently represent multiple k-mer sets.

4 Methods

In this chapter, a compression algorithm for multiple k-mer sets is described along with the corresponding decompression algorithm. The overall algorithm is first described, followed by implementation techniques to perform the compression efficiently in terms of time and memory. Also, a parallel algorithm for SPSS construction, which can be used by itself and also for the speed-up of the compression algorithm, is introduced.

4.1 Compression of multiple k-mer sets by iterative decomposition

Given N k-mer sets $S_1, S_2, ..., S_N$, the following procedure compresses them and saves the result to disk, split to multiple files.

1. Initialize a variable n with N.
2. Create a directed graph G with n vertices $v_1, v_2, ..., v_n$.
3. Repeat the following operations.
   a. Find $1 \leq i < j \leq n$ such that $|S_i \cap S_j|$ is maximized.
   b. Create a set $S_{n+1} = S_i \cap S_j$.
   c. Update S_i and S_j as follows: $S_i \leftarrow S_i \backslash S_{n+1}, S_j \leftarrow S_j \backslash S_{n+1}$.
   d. Add a vertex v_{n+1} to G.
   e. Add two edges in G: one from v_i to v_{n+1} and one from v_j to v_{n+1}.
   f. Update n as follows: $n \leftarrow n + 1$.
4. For each $i = 1, 2, ..., n$, construct SPSS from S_i and save it to disk.
5. Save G to disk.

Figure 3 illustrates the idea of the algorithm.

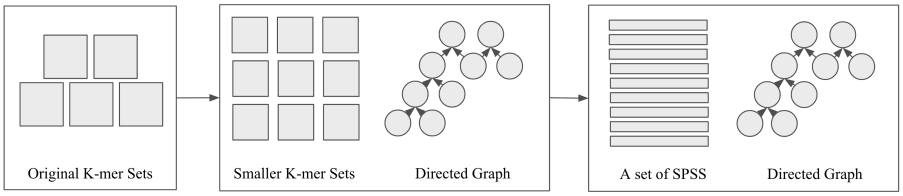


Figure 3 Illustration of the compression algorithm.

In each iteration (step 3), $\sum_{1 \leq x \leq n} |S_x|$, which corresponds to the number of k-mers that have to be saved to disk, decreases by $\max_{1 \leq i < j \leq n} |S_i \cap S_j|$. If they were to be saved to disk naively (e.g., without using SPSS), it is apparent that the required disk usage would decrease as well. However, the construction of efficient (small-weight) SPSS gets harder when the size

of a k-mer set is small. The main question, which will be answered in the later chapter by showing that the compression actually works with real data, is whether the decrease in the number of total k-mers can trump the growing difficulty of making efficient SPSS.

In our implementation, when saving SPSS to disk (step 4), bzip2 was used to further compress the strings, the alphabet of which is $\{A, C, G, T\}$. Speaking of G, its adjacency list representation was saved to disk.

Time complexity

Suppose that we can check the equality of two k-mers in constant time and that we can calculate a hash value from a k-mer in constant time. By representing a k-mer set with a hash table, we can perform the iteration of all the k-mers in linear time and the existence check of a k-mer in constant time. This is a reasonable assumption because k is usually small.

Let C be $\max_{1 \leq i \leq N} |S_i|$ where $S_1, S_2, ..., S_N$ are the input data. In each iteration, the calculation of $|S_x \cap S_y|$ $(1 \leq x < y \leq n)$ can be done in $O(C)$ time. The selection of i and j to maximize $|S_i \cap S_j|$ (step 3a), if implemented naively, will take $O(C(N+M)^2)$ time where M is the number of iterations in step 3. Then, the time complexity for all the iterations will be $O(CM(N+M)^2)$.

This can be improved by using the binary heap [18]. Before the first iteration, $|S_i \cap S_j|$ is calculated for each $1 \leq i < j \leq N$, and the heap is made out of the values. These can be done in $O(CN^2)$ time and $O(N^2)$ time, respectively. In each iteration, the minimum element in the heap is looked up in $O(1)$ time, and $O(N+M)$ elements are updated. The calculation of the updated values will take $O(C(N+M))$ time, and the update of the heap will take $O((N+M)log(N+M))$ time. The resulting time complexity for all the iterations will be $O(CN^2 + M(C(N+M) + (N+M)log(N+M)))$. Further speed-up is possible with multiple threads. Let T be the number of threads available. The total time complexity would then be $O(\frac{CN^2}{T} + N^2 + M(\frac{C(N+M)}{T} + (N+M)log(N+M)))$. Note that multiple threads do not speed up the construction of and the updates of the heap.

Another possible approach is to use a hash table to keep track of $|S_i \cap S_j|$ for each $1 \leq i < j \leq N$. This leads to $O(\frac{CN^2 + M(C(N+M) + (N+M)^2)}{T})$ time because it requires $O(\frac{(N+M)^2}{T})$ time for finding i and j to maximize $|S_i \cap S_j|$ in each iteration instead of the construction of the heap ($O(N^2)$ time) and the update of the heap ($O((N+M)log(N+M))$ time in each iteration). Because of its simplicity and the fact that C is often much larger than others, this approach was used in our implementation.

Selection of iteration size

The number of iterations M is a parameter in the algorithm. The more the iterations, the smaller each k-mer set $(S_1, S_2, ..., S_n)$ becomes, but it does not necessarily mean that it leads to smaller resulting file size, or better compression. This is because it gets harder to make efficient (small-weight) SPSS when the input k-mer set is small.

We chose to dynamically select M by monitoring the total weight of SPSS during the iterations of step 3. The value, which can be calculated by constructing SPSS to represent each k-mer set and summing up the lengths of all the elements (strings), can be thought of as an approximate of the final file size. In our implementation, the value was monitored in a regular interval, and when its decrease was below a threshold, the loop was stopped. Specifically, the interval was set to $\lfloor N/8 \rfloor + 1$, and the threshold was set to 1.25 %.

4.2 Decompression

Given the compressed data saved to disk and i ($1 \leq i \leq N$), it is possible to reconstruct the original value of S_i with the following procedure.

1. Load G from disk.
2. Find vertices $v_{x_1}, v_{x_2}, ...v_{x_m}$ in G that are reachable from v_i using breadth-first search.
3. Obtain $S_{x_1}, S_{x_2}, ..., S_{x_m}$ by loading files from disk and getting k-mer sets from SPSS.
4. Return $S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_m}$.

As can be seen, by separating compressed data into multiple files, it was made possible to efficiently reconstruct the original k-mer sets. That is, we do not necessarily have to load all of $S_1, S_2, ..., S_n$ to memory when reconstructing one k-mer set that is requested.

The proof of correctness of this algorithm is shown in the appendix.

4.3 Implementation

The compression algorithm, if implemented naively, requires a lot of time and memory, even with the methods described in the "Time complexity" section. This section describes techniques that can be further employed to make it fast and memory efficient.

The compression algorithm, combined with these techniques, is named "Iterative SPSS Decomposition".

Representation of k-mer sets

$S_1, S_2, ..., S_n$ are primarily represented by SPSS in favor of its low memory usage. Specifically, the SPSS represented by $\{s_1, s_2, ...s_m\}$ is saved to a bit vector whose length is $2(|s_1| + |s_2| + ... + |s_m|)$, along with an array of non-negative integers $\{|s_1| - k, |s_2| - k, ..., |s_m| - k\}$. It can be more memory efficient than having m bit vectors, as having multiple bit vectors entails keeping multiple pointers, whose sizes cannot be ignored. Furthermore, as the array of integers often contains small numbers, the variable-length integer encoding can be used to reduce its memory usage. In our implementation, StreamVByte [10] was used because it supports fast compression and decompression with SIMD instructions.

Update of k-mer sets in each iteration

In each iteration of the compression algorithm (step 3), two k-mer sets are updated, and one k-mer set is created. Here, a method to efficiently perform the operations is described.

First, it uses a bucketed structure to represent a k-mer set internally. When we have a k-mer, it can be represented with $2k$ bits (e.g., by representing A, C, G, and T with 00, 01, 10, and 11, respectively). In the bucketed structure, k-mers are divided into 2^h buckets. h higher bits out of the $2k$ bits are used to select buckets, and $2k - h$ lower bits are saved to each bucket. Figure 4 illustrates the bucketed structure with example data.

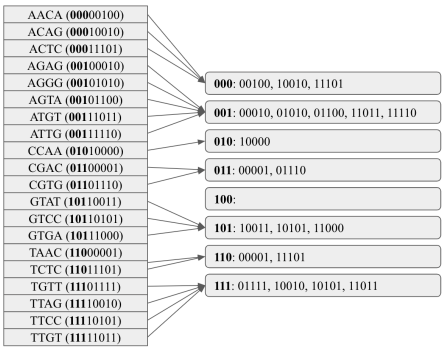


Figure 4 Illustration of the bucketed structure with example data ($k = 4, h = 3$, 20 k-mers).

With the bucketed structure, it is possible to calculate the intersection or the differences between two k-mer sets efficiently with multiple threads by dividing and assigning buckets to multiple threads. It is also possible to efficiently construct the bucketed structure from a k-mer set represented by SPSS with multiple threads, by dividing the strings into chunks, assigning chunks to multiple threads, making a bucketed k-mer set for each chunk, and merging them using 2^h mutex locks.

When two k-mer sets S_i, S_j represented by SPSS are given, SPSS to represent $S_i \cap S_j$, $S_i \backslash S_j$, and $S_j \backslash S_i$ can be constructed as follows. First, the bucketed structures are created for S_i and S_j. Second, the calculations of the intersection and the differences are performed on them. Finally, the SPSS for each of the three obtained k-mer sets are constructed. This procedure is illustrated in Figure 5. All of the operations can be done efficiently with multiple threads by utilizing the bucketed structure and by using the SPSS construction algorithm that will be introduced later.

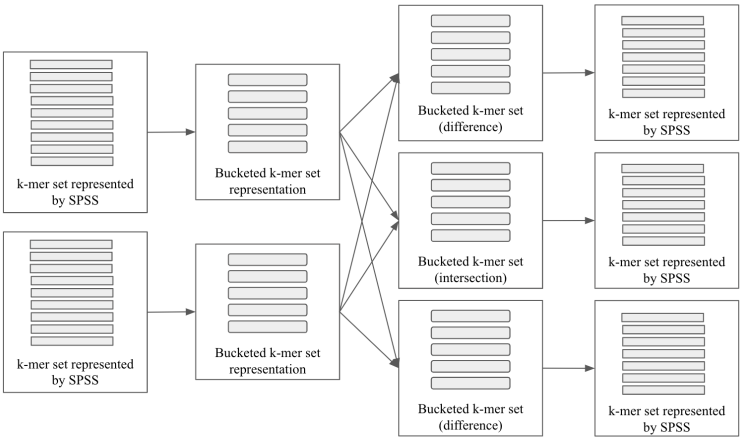


Figure 5 Illustration of the update of k-mer sets in each iteration of the compression algorithm (step 3). All the calculations can be done efficiently with multiple threads, using the feature of the bucketed structure and the SPSS construction algorithm that will be introduced later.

Calculation of intersection size

Calculation of the intersection size between two k-mer sets is repeated during the compression. Here, a method to perform the operation with small memory usage is discussed.

We take the approach of approximating intersection sizes by "sampling" k-mers from each k-mer set. The sampling is performed by constructing the bucketed structure from a k-mer set (represented by SPSS) and randomly deleting buckets. Here, the same set of buckets gets deleted for each k-mer set. If $x\%$ of the buckets are left after the sampling, we can calculate an estimate of the intersection size between the original two k-mer sets by calculating the intersection size between the sampled k-mer sets and by multiplying the value by $\frac{100}{x}$. This idea of using sampled k-mer sets to calculate intersection sizes is illustrated in Figure 6.

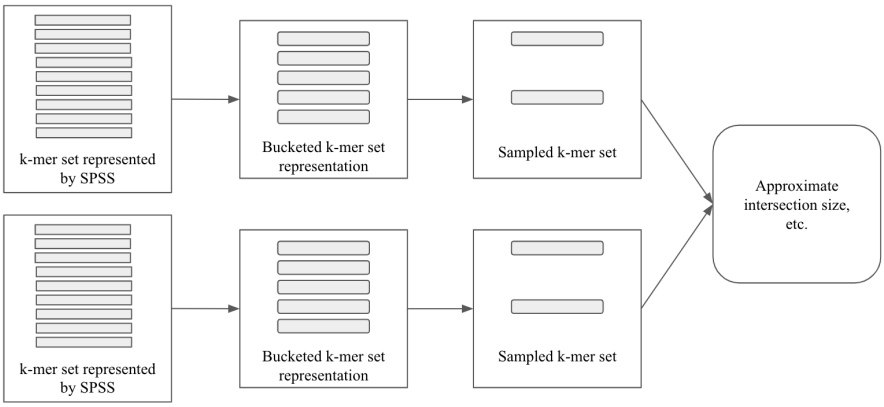


Figure 6 Illustration of the calculation of an approximate intersection size between 2 k-mer sets.

In our implementation, as constructing the bucketed structure and performing the sampling take some time, we kept the sampled k-mer sets for each k-mer set on memory during the iterations, and they got updated when the underlying k-mer sets were updated or when a new k-mer set was created. Speaking of the sampling size, $x = 2$ was used.

4.4 Fast construction of small-weight SPSS

The small-weight SPSS construction algorithm presented in [16], UST, does not support parallel processing. Here, we introduce a parallel small-weight SPSS construction algorithm.

Let $G = (V, E)$ be a compacted de Bruijn graph. The following procedure creates small-weight SPSS for the corresponding k-mer set.

1. Create a graph G' whose vertex set is the same as G. Each vertex in G' has two sides as in G.
2. Prepare n mutex locks $mu_1, mu_2, ..., mu_n$.
3. For each vertex v in G', perform the following operations in parallel.
   a. If there is a vertex v' such that $v' \neq v$ and there is an edge between the right side of v and the left side of v' in G, perform the following operations.
      i. Acquire the locks $mu_{id(v)\%n}$ and $mu_{id(v')\%n}$. Here, id is a bijective function from V to integers from 0 to $|V| - 1$.
      ii. If there are no edges incident to the right side of v and there are no edges incident to the left side of v' in G', add an edge in G' to connect the right side of v and the left side of v'.
      iii. Release the locks $mu_{id(v')\%n}$ and $mu_{id(v)\%n}$.
   b. If there is a vertex v' such that $v' \neq v$ and there is an edge between the right side of v and the right side of v' in G, perform the similar operations as (a).

   c. If there is a vertex v' such that $v' \neq v$ and there is an edge between the left side of v and the right side of v' in G, perform the similar operations as (a).
   d. If there is a vertex v' such that $v' \neq v$ and there is an edge between the left side of v and the left side of v' in G, perform the similar operations as (a).
4. Divide V to $V_1, V_2, ..., V_m$ so that the elements of each are connected and each is maximal.
5. For $i = 1, 2, ..., m$, perform the following operation in parallel.
   a. If both the sides of v have edges incident to it for all $v \in V_i$, remove one arbitrary edge that connects two vertices in V_i.
6. Make a set of vertices V' such that for each $v \in V'$, there is no edges incident to the left side of v or there is no edges incident to the right side of v.
7. Prepare a set of strings S.
8. For each vertex v in V', perform the following operations in parallel.
   a. Find the longest path from v in G'. It is uniquely determined.
   b. Let the endpoint of the path be v'. If the length of the path is 1 or $label(v) < label(v')$, add to S the string represented by the path. Here, the dictionary order of strings is used to compare strings.
9. Return S.

Step 4 can be done efficiently if we use a disjoint-set data structure that can be used by multiple threads. In our experiments, an implementation based on [3] was used.

The proof of correctness of this algorithm is shown in the appendix.

5 Experiments and Results

In this chapter, the effectiveness of the compression algorithm is shown along with the effects of the number of input files and the comparison to the compression method based on Mantis [13, 1], one of the state-of-the-art colored de Bruijn graph representations. The performance gain of the proposed SPSS construction algorithm in multi-thread environments is also shown along with the comparison to the existing method, UST [16].

5.1 Compression of multiple k-mer sets

The proposed algorithm to compress multiple k-mer sets was tested with 3292 E. coli whole-genome sequencing data and 2555 human RNA-Seq data. The former is the data that was accessible for Project PRJNA292667 on Sequence Read Archive [9] on 12/4/2020. The latter is the data used in [17], but some files were omitted because 5 files were corrupted (SRR346129, SRR346128, SRR346127, SRR448331, and SRR448330) and some files contained no reads whose length is at least k. For the E. coli data, we used $k = 23$ and the cutoff value of 4, i.e., k-mers that appeared 3 times or less were ignored to leave out erroneous data. For the RNA-Seq data, we used $k = 23$, and different cutoff values were used for different input files, following the procedures in [17]. The reason we did not apply the dynamic cutoff setting to the E. coli data is that the file sizes did not vary that much in the E. coli data compared to the human RNA-Seq data. 32-core Intel Cascade Lake machines running at 2800.262 MHz were used.

Results

As a result, the E. coli genome data could be compressed to 2.42% in size from the naive representation of the k-mers (the representation in which each k-mer uses $2k$ bits), whereas the compression rate was 6.03 % with zero iterations, which corresponds to skipping step 3 of the compression algorithm and individually compressing k-mer sets with SPSS and bzip2.

This shows that by repeating the iterations, the algorithm successfully reduced the file size, thereby illustrating the algorithm's effectiveness. The effectiveness was also shown with the human RNA-Seq data, and the results are summarized in Table 1.

Table 1 Comparison between the compression ratio that could be achieved without iterations, which corresponds to individually compressing each k-mer set with SPSS and bzip2, and the one that could be achieved with iterations, which corresponds to the actual result by the proposed compression algorithm.

	Without iterations	Actual result
E. coli genome (3292 files)	6.03%	**2.42%**
Human RNA-Seq (2555 files)	8.06%	**6.04%**

Effects of the number of input files

We conducted an experiment to see how the compression capability is affected by the number of input files.

First, the 3292 E. coli data were shuffled randomly. Then, we applied our algorithm with the first 400, 800, ..., and 3200 files. The compression rates for each setting are shown in Figure 7. We can see that the more files there are, the better compression it can achieve.

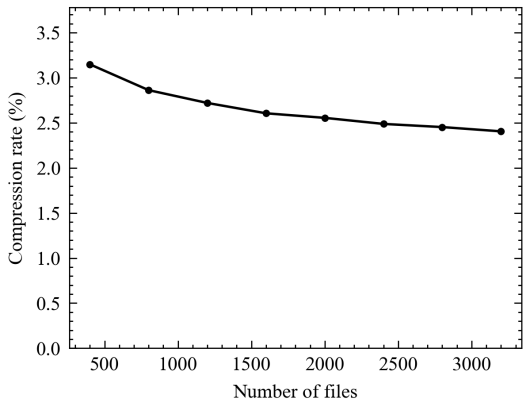


Figure 7 Compression rates that could be achieved with our proposed algorithm with 400, 800, ..., and 3200 files of E. coli data.

Comparison to existing methods

As we saw in the earlier chapter, colored de Bruijn graphs can be used to represent multiple k-mer sets efficiently. In this section, one of the recent colored de Bruijn graph implementations, Mantis [13, 1], is compared with our proposed method in terms of compression capability, required time, and memory usage.

To see the compression capability of Mantis, the following steps were taken. First, we used Squeakr [14] to pre-process the 3292 E. coli data or the 2555 human RNA-Seq data (with "-n" option) and used "mantis build" command (with "-s 30" option) followed by "mantis mst" command to build the colored de Bruijn graph. For "mantis mst", we specified to use 32 threads, whereas we did not have the multi-threading option for "mantis build". And boundaries.bv, deltas.bv, parents.bv, meta_info.json, and dbg_cqf.ser were further compressed with bzip2 and their total file size was measured. The applications were run on machines with the same CPU specifications as the ones that were used in the experiments for our method.

As you can see in Table 2, our proposed method performed better than Mantis in terms of compression capacity. We can also see that our implementation used more time than Mantis but used less memory than Mantis.

Table 2 Comparison between our method and Mantis [13, 1].

Data	Metric	Mantis	Proposed
E. coli genome (3292 files)	Time	**2h24m10s**	3h59m08s
	Memory	33.5GB	**26.1GB**
	Compression	3.70%	**2.42%**
Human RNA-Seq (2555 files)	Time	**1h14m54s**	2h44m21s
	Memory	27.4GB	**24.8GB**
	Compression	6.97%	**6.04%**

Note that our algorithm and Mantis have different strengths other than the differences in compression capability, time, and memory. The structure of Mantis, once loaded to memory, achieves fast responses to queries necessary for tasks like genotyping, for which our method cannot be used. On the other hand, with our proposed method, it is not necessary to load all the data to memory for decompression, whereas we have to do so with Mantis.

5.2 Fast construction of small-weight SPSS

We introduced a small-weight SPSS construction algorithm that can be parallelized. In this section, the performance of the proposed algorithm is measured, and it is compared with the performance of the existing non-parallelizable algorithm, UST [16].

For the experiments, we obtained a set of canonical k-mers from whole-genome sequencing data of E. coli (SRR3160259 from Sequence Read Archive [9]). k was set to 23, and the cutoff was set to 4. All the experiments were repeated 10 times on 16-core Intel Cascade Lake machines running at 2800.262 MHz, and the average values were considered.

Effects of the number of locks

In this section, the effect of the number of locks, a parameter of the algorithm, is measured.

We fixed the number of threads to 16 and saw the effects of the number of mutex locks. Figure 8 shows the time required for construction of SPSS with 8, 16, 32, ..., or 2048 locks. We can see a downward trend with the number of locks, but it is almost flat after 512 locks. Based on this result, we chose to use 1024 mutex locks in the rest of the experiments.

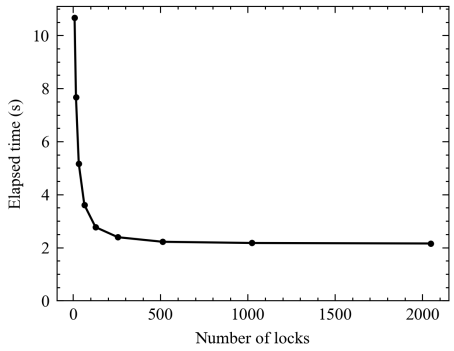


Figure 8 Required time for construction of SPSS using 8, 16, 32, ..., or 2048 mutex locks.

Comparison to existing methods

Here, we compare the performance of the proposed SPSS construction algorithm with the existing algorithm, UST [16], and show that our algorithm performs well with multiple threads whereas UST does not support parallelization.

We varied the number of threads and measured the proposed algorithm's performance. Figure 9 shows the required time with 1, 2, ..., or 16 threads. It took 17.4 seconds with 1 thread, 10.0 seconds with 2 threads, and down to 2.18 seconds with 16 threads.

With UST, the operation took 13.0 seconds. So, our method performed worse than UST with a single thread, but the runtime could be reduced by 83.2% with 16 threads compared to UST with a single thread. Note that UST cannot utilize multiple threads.

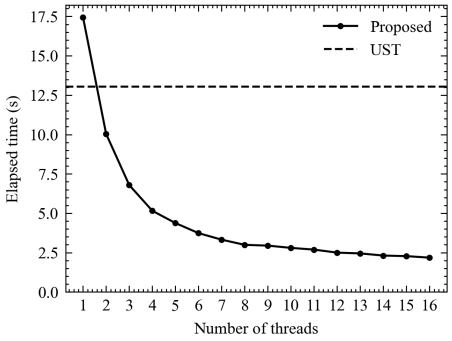


Figure 9 Required time for construction of SPSS using 1, 2, ..., or 16 threads with the proposed algorithm. The baseline value (required time with UST [16]) is also shown.

6 Conclusion and Future Work

In this paper, we introduced a compression algorithm for multiple k-mer sets that combines the power of SPSS along with a method to utilize similarities in k-mer sets, along with techniques to make the compression algorithm efficient, including a small-weight SPSS construction algorithm that runs fast in multi-core environments.

With the compression algorithm, it was possible to compress 3292 E. coli whole-genome sequencing data and 2555 human RNA-Seq data with better compression rate than Mantis [13, 1], one of the state-of-the-art colored de Bruijn graph representations.

In addition to the high compression capability, our compression algorithm has some other benefits. First, the performance of the compression algorithm improves well with multiple threads. This is especially important on machines where many cores are available, which is often the case these days. Second, the proposed method requires small memory usage, as shown in the comparison to Mantis. It indicates that our method can be applied to larger files. Third, with the proposed algorithm, it is not necessary to load all the compressed data to memory when doing the decompression, thereby achieving smaller memory usage on decompression, which will not be the case with methods based on colored de Bruijn graphs.

There are possible extensions to the compression algorithm. Currently, it only supports sets of unique k-mers, and does not support multi-sets of k-mers, or abundance of k-mers, which can preserve more information of original sequence data. Some researches have been done on methods to efficiently handle abundance of k-mers in a single data source, one of which is REINDEER data structure [12]. It is interesting to see whether our compression algorithm can be applied to multi-sets of k-mers.

The proposed compression algorithm can be helpful for researchers who want to work with multiple datasets, maintainers of genome-related databases, etc. Also, the proposed small-weight SPSS construction algorithm, which was faster by 83.2 % than existing methods in 16-core environments and was used in the implementation of the compression algorithm, will speed up SPSS-based programs in multi-core environments.

References

1. Fatemeh Almodaresi, Prashant Pandey, Michael Ferdman, Rob Johnson, and Rob Patro. An efficient, scalable and exact representation of high-dimensional color information enabled via de Bruijn graph search. In *Proceedings of the International Conference on Research in Computational Molecular Biology*, pages 1–18, 2019.
2. Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: A succinct colored de Bruijn graph representation. In *Proceedings of the International Workshop on Algorithms in Bioinformatics*, pages 18:1–18:15, 2017.
3. Richard J. Anderson and Heather Woll. Wait-free parallel algorithms for the union-find problem. In *Proceedings of the ACM symposium on Theory of Computing*, pages 370–380, 1991.
4. Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *Proceedings of the International Workshop on Algorithms in Bioinformatics*, pages 225–235, 2012.
5. Karel Břinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de bruijn graphs. *Genome biology*, 22(1):1–24, 2021.
6. Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.
7. Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995.
8. Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.
9. Rasko Leinonen, Hideaki Sugawara, and Martin Shumway. The sequence read archive. *Nucleic Acids Research*, 39(suppl_1):D19–D21, 2010.
10. Daniel Lemire, Nathan Kurz, and Christoph Rupp. Stream VByte: Faster byte-oriented integer compression. *Information Processing Letters*, 130:1–6, 2018.
11. Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.
12. Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. REINDEER: Efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement_1):i177–i185, 2020.
13. Prashant Pandey, Fatemeh Almodaresi, Michael A. Bender, Michael Ferdman, Rob Johnson, and Rob Patro. Mantis: A fast, small, and exact large-scale sequence-search index. *Cell Systems*, 7(2):201–207, 2018.
14. Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. Squeakr: An exact and approximate k-mer counting system. *Bioinformatics*, 34(4):568–575, 2018.
15. Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An eulerian path approach to DNA fragment assembly. *National Academy of Sciences*, 98(17):9748–9753, 2001.
16. Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving string sets. In *Proceedings of the International Conference on Research in Computational Molecular Biology*, pages 152–168, 2020.
17. Brad Solomon and Carl Kingsford. Fast search of thousands of short-read sequencing experiments. *Nature Biotechnology*, 34(3):300–302, 2016.
18. John William Joseph Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

A Proof of correctness of the compression algorithm

In this chapter, we prove the correctness of the compression and decompression algorithm presented in 4.1 and 4.2.

We first consider the following condition.

▶ **Condition 1.** *Let* $\{v_{x_1}, v_{x_2}, ..., v_{x_{m_i}}\}$ *be a set of vertices in* G *that are reachable from* v_i. $S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_{m_i}}$ *is equal to the initial value of* S_i.

Here, "the initial value of S_i" refers to the value given as part of an input to the compression algorithm if $1 \leq i \leq N$, and it refers to the value initialized in step 3b of the compression algorithm if $N < i$.

We prove the correctness by showing the following theorem.

▶ **Theorem 2.** *If Condition 1 holds true for* $i = 1, 2, ..., n$ *at the beginning of an iteration of step 3 in the compression algorithm, Condition 1 also holds true for* $i = 1, 2, ..., n$ *at the end of the iteration.*

Proof. Let n' be the value of n at the beginning of the iteration, and suppose that v_j and v_k are selected in step 3a of the iteration.

If v_j and v_k cannot be reached from $v_i (i \in \{1, 2, ..., n'\})$, the set of vertices $\{v_{x_1}, v_{x_2}, ..., v_{x_{m_i}}\}$ that are reachable from v_i does not change during the iteration. As $S_{x_1}, S_{x_2}, ..., S_{x_{m_i}}$ do not change as well, the value of $S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_{m_i}}$ gets unchanged, and Condition 1 holds true after the iteration.

If v_j can be reached from v_i and v_k cannot be reached from $v_i (i \in \{1, 2, ..., n'\})$, $v_{n'+1}$ gets added to the set of vertices that are reachable from v_i. If we suppose that the set of vertices that are reachable from v_i at the beginning is $\{v_j, v_{x_1}, v_{x_2}, ..., v_{x_{m_i}}\}$, the set of vertices that are reachable from v_i at the end of the iteration will be $\{v_j, v_{x_1}, v_{x_2}, ..., v_{x_{m_i}}, v_{n'+1}\}$. As S_j gets subtracted by $S_{n'+1}$ and $S_{x_1}, S_{x_2} ..., S_{x_{m_i}}$ do not change over the iteration, the value of $S_j \cup S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_{m_i}}$ at the beginning of the iteration is equal to the value of $S_j \cup S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_{m_i}} \cup S_{n'+1}$ at the end of the iteration. Hence, Condition 1 holds true after the iteration. If v_k can be reached from v_i and v_j cannot be reached from v_i, the same argument applies.

If v_j and v_k are reachable from $v_i (i \in \{1, 2, ..., n'\})$, $v_{n'+1}$ gets added to the set of vertices that are reachable from v_i. If we suppose that the set of vertices that are reachable from v_i at the beginning is $\{v_j, v_k, v_{x_1}, v_{x_2}, ..., v_{x_{m_i}}\}$, the set of vertices that are reachable from v_i at the end of the iteration will be $\{v_j, v_k, v_{x_1}, v_{x_2}, ..., v_{x_{m_i}}, v_{n'+1}\}$. As S_j and S_k gets subtracted by $S_{n'+1}$ and $S_{x_1}, S_{x_2} ..., S_{x_{m_i}}$ do not change over the iteration, the value of $S_j \cup S_k \cup S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_{m_i}}$ at the beginning of the iteration is equal to the value of $S_j \cup S_k \cup S_{x_1} \cup S_{x_2} \cup ... \cup S_{x_{m_i}} \cup S_{n'+1}$ at the end of the iteration. Hence, Condition 1 holds true after the iteration.

As $v_{n'+1}$ does not have outgoing edges, $v_{n'+1}$ itself is the only vertex that is reachable from $v_{n'+1}$ after the iteration. As $S_{n'+1}$ is initialized in the iteration, Condition 1 holds true after the iteration for $i = n' + 1$. ◀

By combining Theorem 1 with the fact that Condition 1 holds true for $i = 1, 2, ..., n$ before the iterations, we can prove that Condition 1 holds true after any number of iterations, thereby showing the correctness of the algorithm.

B Proof of correctness of the SPSS construction algorithm

In this chapter, we prove the correctness of the SPSS construction algorithm presented in 4.4.

As shown in [16], a set of strings represented by the paths of a path cover on a compacted de Bruijn graph is SPSS of the corresponding k-mer set. We prove that the proposed algorithm produces SPSS by showing that step 8 considers all the vertices exactly once and that a path cover on the compacted de Bruijn graph is considered.

Proof. After step 3 of the algorithm, for each vertex v in G', one of the following is true.

- There is one edge incident to the left side of v and there is one edge incident to the right side of v.
- There is one edge incident to the left side of v and there are no edges incident to the right side of v.
- There are no edges incident to the left side of v and there is one edge incident to the right side of v.
- There are no edges incident to v.

Hence, in step 5, for each $i = 1, 2, ..., m$, one of the following is true.

- There is a path p in G' such that p contains all the vertices in V_i and either of p's endpoint vertices has a side without edges.
- There is a loop such that it contains all the vertices in V_i. Here, a loop refers to a set of vertices such that they are connected and each of them has one edge incident to its left side and one edge incident to its right side.

Loops get removed during step 5, and the former of the above is true for each $i = 1, 2, ..., m$ after step 5.

If $|V_i| = 1 (i \in \{1, 2, ..., m\})$, the vertex $v \in V_i$ will be added to V' in step 6. The label of v will be considered once in step 8.

If $|V_i| > 1 (i \in \{1, 2, ..., m\})$, there is a path $\{v_1, v_2, ..., v_{|V_i|}\}$ such that it contains all the vertices in V_i, one side of v_1 does not have edges incident to it, one side of $v_{|V_i|}$ does not have edges incident to it, and $v_2, v_3, ..., v_{|V_i|-1}$ have edges on both of their sides. v_1 and $v_{|V_i|}$ will be added to V' in step 6. And the path will be considered twice in step 8a. But as only one of $label(v_1) < label(v_{V_i})$ and $label(v_{V_i}) < label(v_1)$ can be true, the path is only considered once in step 8 as a whole. ◀

Compressing and Indexing Aligned Readsets

Travis Gagie
Dalhousie University, Halifax, Canada

Garance Gourdel
IRISA – Inria Rennes – Université Rennes 1 – ENS, France

Giovanni Manzini
University of Pisa, Italy

Abstract

Compressed full-text indexes are one of the main success stories of bioinformatics data structures but even they struggle to handle some DNA readsets. This may seem surprising since, at least when dealing with short reads from the same individual, the readset will be highly repetitive and, thus, highly compressible. If we are not careful, however, this advantage can be more than offset by two disadvantages: first, since most base pairs are included in at least tens reads each, the uncompressed readset is likely to be at least an order of magnitude larger than the individual's uncompressed genome; second, these indexes usually pay some space overhead for each string they store, and the total overhead can be substantial when dealing with millions of reads.

The most successful compressed full-text indexes for readsets so far are based on the Extended Burrows-Wheeler Transform (EBWT) and use a sorting heuristic to try to reduce the space overhead per read, but they still treat the reads as separate strings and thus may not take full advantage of the readset's structure. For example, if we have already assembled an individual's genome from the readset, then we can usually use it to compress the readset well: e.g., we store the gap-coded list of reads' starting positions; we store the list of their lengths, which is often highly compressible; and we store information about the sequencing errors, which are rare with short reads. There is nowhere, however, where we can plug an assembled genome into the EBWT.

In this paper we show how to use one or more assembled or partially assembled genome as the basis for a compressed full-text index of its readset. Specifically, we build a labelled tree by taking the assembled genome as a trunk and grafting onto it the reads that align to it, at the starting positions of their alignments. Next, we compute the eXtended Burrows-Wheeler Transform (XBWT) of the resulting labelled tree and build a compressed full-text index on that. Although this index can occasionally return false positives, it is usually much more compact than the alternatives. Following the established practice for datasets with many repetitions, we compare different full-text indices by looking at the number of runs in the transformed strings. For a human Chr19 readset our preliminary experiments show that eliminating separators characters from the EBWT reduces the number of runs by 19%, from 220 million to 178 million, and using the XBWT reduces it by a further 15%, to 150 million.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases data compression, compact data structures, FM-index, Burrows-Wheeler Transform, EBWT, XBWT, DNA reads

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.13

Supplementary Material *Software (Source Code)*: `https://github.com/fnareoh/Big_XBWT` archived at `swh:1:dir:ec67fec8b70e6b837a1b497c7fc07cb5d179c512`

Funding *Travis Gagie*: Funded by NSERC Discovery Grant RGPIN-07185-2020, NIH R01HG011392 and NSF IIBR 2029552.
Garance Gourdel: Partially funded by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).
Giovanni Manzini: Supported by the Italian MIUR PRIN project 2017WR7SHH.

Acknowledgements Many thanks to Jarno Alanko and Uwe Baier for their XBWT-construction software, and to Diego Díaz, Richard Durbin, Filippo Geraci, Giuseppe Italiano, Ben Langmead, Gonzalo Navarro, Pierre Peterlongo, Nicola Prezza, Giovanna Rosone, Jared Simpson, Jouni Sirén and Jan Studený for helpful discussions.

1 Introduction

The FM-index [23] is an important data structure in both combinatorial pattern matching and bioinformatics. Its most important application so far has been in standard short-read aligners – Bowtie [39, 38] and BWA [41] have together over 70 thousand citations and are used every day in clinics and research labs worldwide – but it has myriad other uses and more are still being discovered. Just within computational genomics, FM-indexes have been generalized from single strings to collections of strings for tools such as BEETL [15], RopeBWT [40] and Spring [11], to de Bruijn graphs for tools such as BOSS [8], VARI [48] and Rainbowfish [2], and to graphs for tools such as vg [27]. Recent breakthroughs [25] mean we can now scale FM-indexes to massive but highly repetitive pan-genomic datasets for a new generation of tools [36].

As genomic datasets grow exponentially (from the Human Genome Project to the 1000 Genomes Project and the 100K Genomes Project) and standards for sequencing coverage increase (from less than 10x a few years ago to 30x and 50x now and over 100x for some applications), an obvious question is whether and how the recent breakthroughs in FM-indexing of repetitive datasets can be turned into comparable advances in indexing readsets, so more researchers can efficiently mine them for biomedical insights. For example, extrapolating from previous experiments [36], it should be possible to index both haplotypes from 2705 individuals in less than 100 GB of RAM. In contrast, the readset from the final phase of the 1000 Genomes Project consisted of reads from 2705 individuals and was released as a 464 GB Burrows-Wheeler Transform (BWT) [17], which is beyond the resources of most labs to process. This almost five-fold increase (from 100 to 464 GB) seems reasonable, given the range of lengths and the error rate of short-read sequencing technologies, but those reads were trimmed and error-corrected before their BWT was computed, making that increase harder to justify and thus a target for improvement. Although experimenting with that particular readset is beyond the scope of this paper, since it occupies 87 TB uncompressed, we expect the insights and techniques we develop here will eventually be useful in software able to handle efficiently inputs of that scale.

Recent results on FM-indexing repetitive datasets [25] have shown that the index performance depends on the number of runs in the transformed sequence, where a run is a maximal non-empty unary substring. For example, if the BWT of a dataset of (uncompressed) size n has r runs, we can design an FM-index of size $O(r \log \log n)$ supporting the count and locate operations in optimal linear time. Hence, if a BWT variant produces a transformed string with a smaller number of runs, the resulting index will be smaller and equally fast. The naïve approach to FM-indexing readsets is to concatenate the reads with copies of a separator character between them, and FM-index the resulting single string. However, computing the BWT of such a long string is a challenge and each separator character causes several runs in that BWT. The most competitive indexes for readsets are based on Mantaci et al.'s [46] Extended Burrows Wheeler Transform, which is also easier to build for readsets. The first index for readsets based on the EBWT was BEETL [15], followed by RopeBWT [40]; recently the EBWT has been used also by the Spring compressor [11] specialized for FASTQ reads. BEETL and RopeBWT use explicit separator characters but such characters could be replaced by bitvectors marking positions at the ends of reads.

BEETL and RopeBWT use a heuristic to reduce the number of runs in the EBWT: they conceptually put the separator characters at the ends of reads into the co-lexicographic order (lexicographic order on the reverse string, also referred to as reverse lexicographic order) of the reads, so that the final characters or reads with similar suffixes are grouped together in

the EBWT. This often works surprisingly well but in the worst case it cannot make up for the lack of context for sorting those characters into their places in the EBWT. Our proposal in this paper is to graft the reads onto their assembled genome, or a reference genome to which they align well, and index the resulting labelled tree with Ferragina et al.'s [22] XBWT. To this end we assume that we know how the reads align to the assembled/reference genome: this is not an unreasonable assumption since alignment is the initial step of any readset analysis.

In order to implement our idea we have to overcome a significant hurdle: as the coverage increases so does the amount of raw data produced by a single NGS experiment. Although the high coverage implies that the data is highly compressible, the actual compression process, ie the construction and the compression of the XBWT, must be done partially in externally memory since the input will be usually much larger than the available RAM. Another contribution of the paper is therefore the adaptation of the prefix-free parsing (PFP) technique [7] to the construction of the XBWT. PFP has been proposed for the construction of BWTs of collections of similar genomes: the initial parsing phase is able to compress the input maintaining enough information to compute the BWT working on the compressed representation. In this paper we adapt PFP to readsets, taking care also of the "grafting" of the single reads to the reference/assembled genome. Given a pattern P, our index could answer $count(P)$ and $locate(P)$ queries which report respectively the number of positions where P occurs and the list of positions where P occurs. The main drawback to our index, apart from taking one or more assembled or partially assembled genomes as a base, is that it can return a false-positive in the *count* operation when an occurrence of a pattern starts in the trunk of an alignment tree and ends in a branch. In other words, the index can report a match that is not completely contained within a read but would be if we padded the read on the left with enough characters copied from just before where it aligns. In a locate operation false-positives could be identified, but this operation is much slower. Even this is not entirely bad, however, and it is conceivable this bug could sometimes be a feature. The analysis of those false positive and the size of the bit vectors marking the end of reads is left as future work.

The rest of the paper is organized as follows. In Section 2 we first describe the BWT and FM-indexes, then the EBWT and XBWT and the concept of Wheeler graph that unifies them. In Section 3 we introduce our idea for indexing aligned readsets with the XBWT and we prove some theoretical results supporting it. In Section 4 we describe how we adapt PFP to indexing readsets, which allows us to experiment with larger files than would otherwise be possible with reasonable resources. In Section 5 we present our experimental results showing that applying the XBWT to index readsets works well in practice as well as in theory. Finally, we outline in Section 6 how our study of storing reads with the XBWT may improve the space usage of the hybrid index [20, 21, 26].

2 Concepts

For a better understanding of the problem context, we give a succinct description of the second generation sequencing technique. Most publicly available readsets are from Illumina sequencers [35] which rely on sequencing by synthesis. For this process, millions or billions of single-stranded snippets of DNA called templates are deposited onto a slide and amplified into clusters of clones. In each sequencing cycle we learn one base of each template: we add DNA polymerase and specially terminated bases; the polymerase attaches a terminated base to each strand, complementary to the next base in the strand; we shine a light on the slide

and the terminated bases glow various colours; we take a photo and note the colour of each cluster; and finally, we treat the slide to remove the terminators. Sometimes, however, one of the added bases is not correctly terminated, so the polymerase attaches first it and then another base to a strand in some cluster; that strand is then out of step with the rest of the cluster, and the cluster will have a mix of colours in the photos for subsequent sequencing cycles. As we go through more and more sequencing cycles, more strands tend to fall out of step, resulting in less reliable results. (For futher discussion we refer the reader to, e.g., Langmead's lecture on this topic [37].) This tendency means sequencing by synthesis has an asymmetric error profile, with errors more likely towards the ends of the reads. It follows that sequencing errors tend to be near the end of the reads: our index is designed to take advantage of this feature (see Theorem 2).

2.1 BWT and FM-index

The Burrows-Wheeler Transform (BWT) [10] of a string S is a permutation of the characters in S into the lexicographic order of the suffixes that immediately follow them, considering S to be cyclic. For example, as shown on the left in Figure 1, the BWT of `GATTAGATACAT$` is `TTTCGGAA$AATA`, assuming `$` is a special end-of-string symbol lexicographically smaller than all other characters. Because the BWT groups together characters that precede similar suffixes, it tends to convert global repetitiveness into local homogeneity: e.g., for any string α, the BWT of α^t consists of $|\alpha|$ unary substrings of length t each; even the BWT in our example has length 13 but consists of only 8 maximal unary substrings (called runs). This property led Burrows and Wheeler to propose the BWT as a pre-processing step for data compression and Seward [55] used it as the basis for the popular `bzip2` compression program.

The BWT is also the basis for the FM-index [23], one of the first and most popular compressed indexes, which is essentially a rank data structure over the BWT combined with a suffix-array sample. The FM-index is an important data structure in combinatorial pattern matching and bioinformatics, and is itself the basis for popular tools such as Bowtie [39, 38] and BWA [41] that align DNA reads to reference genomes. We refer the reader to Navarro's [49] and Mäkinen et al.'s [45] textbooks for detailed discussions of how FM-indexes are implemented and used for read alignment.

2.2 EBWT

Although alignment against one or more reference genomes remains a key task in bioinformatics, there is growing interest in compressed indexing of sets of reads [17, 34]. The FM-index plays a central role here too: Mantaci et al. [47] generalized the BWT to the Extended BWT (EBWT), which applies to collections of strings, and then Cox et al. [5, 13, 30] used an FM-index built on the EBWT in their index BEETL for readsets. The same construction was also used in subsequent indexes for readsets, such as RopeBWT [40] and Spring [11].

The EBWT of a collection of strings is a permutation of the characters in those strings into the lexicographic order of the suffixes that immediately follow them, considering each string to be cyclic. For example, as shown on the right in Figure 1, the EBWT of `GATTA$`, `TTAGA$`, `TAGATA$`, `GATAC$` and `ATACAT$` is `TCAAATTGTTTTCGG$GAAAA$$ATAAAT$A$`. When we see the BWT and EBWT as permutations of characters, the BWT of a single string has a single cycle, whereas the EBWT of a collection of strings has a cycle for each string. This means it is easier to build the EBWT and update it when a string is added or deleted, than to build and update the BWT of the concatenation of the collection with the strings separated by copies of a special character. We refer the reader to Egidi et al.'s [18, 19] and Díaz-Domínguez and Navarro's [16] recent papers for descriptions of efficient construction and updating algorithms.

	F	L
0	$GATTAGATACAT	
1	ACAT$GATTAGAT	
2	AGATACAT$GATT	
3	AT$GATTAGATAC	
4	ATACAT$GATTAG	
5	ATTAGATACAT$G	
6	CAT$GATTAGATA	
7	GATACAT$GATTA	
8	GATTAGATACAT$	
9	T$GATTAGATACA	
10	TACAT$GATTAGA	
11	TAGATACAT$GAT	
12	TTAGATACAT$GA	

	F	L		F	L
0	$ATACAT		16	ATTA$	G
1	$GATA	C	17	C$GAT	A
2	$GATT	A	18	CAT$ATA	
3	$TAGATA		19	GA$TT	A
4	$TTAG	A	20	GATA$TA	
5	A$GAT	T	21	GATAC	$
6	A$TAGAT		22	GATTA	$
7	A$TTA	G	23	T$ATACA	
8	AC$GA	T	24	TA$GA	T
9	ACAT$AT		25	TA$TAGA	
10	AGA$T	T	26	TAC$G	A
11	AGATA$T		27	TACAT$A	
12	AT$ATAC		28	TAGA$	T
13	ATA$TAG		29	TAGATA$	
14	ATAC$	G	30	TTA$G	A
15	ATACAT$		31	TTAGA	$

Figure 1 The matrices whose rows are the lexicographically sorted rotations of `GATTAGATACAT$` **(left)** and of `GATTA$`, `TTAGA$`, `TAGATA$`, `GATAC$` and `ATACAT$` **(right)**. The BWT and EBWT are `TTTCGGAA$AATA` and `TCAAATTGTTTTCGG$GAAAA$$ATAAAT$A$` with 8 and 19 runs, respectively.

Despite its benefits, the EBWT sometimes does not take full advantage of its input's compressibility. In our example, as Figure 1 shows, even though all the strings in the collection are substrings of `GATTAGATACAT$` with copies of `$` appended to them, their EBWT has more than twice as many runs as its BWT. As a heuristic for reducing the number of runs, and thus reducing BEETL's space usage, Cox et al. suggested considering the lexicographic order of the copies of `$` to be the strings' co-lexicographic order. This does not help in cases such as our example, however, for which the EBWT still has 19 runs even with that ordering. Bentley et al. [6] recently gave a linear-time algorithm to find the ordering of the copies of `$` that minimizes the number of runs, but it has not been implemented and it is unclear whether it is practical for large readsets.

Another way to potentially reduce the number of runs is to remove the copies of `$` entirely, and store an auxiliary ternary vector marking which characters in the EBWT are the first and last characters in the strings. If there are t strings in the collection with total length n, then storing this vector takes $O(t \log(n/t) + t)$ bits (even if some of the strings are empty or consist of only one character). As shown in Figure 2, the EBWT becomes `TTTTTTGTCGGGAACAAAAAATTAAAA`, with only 10 runs. The idea of replacing `$`'s with an auxiliary vector is relatively new since it originates from seeing the EBWT as a special case of Wheeler graphs [24] which are described in the next section.

2.3 Wheeler Graphs and XBWT

Wheeler graphs were introduced by Gagie, Manzini and Sirén [24] as a unifying framework for several extensions of the BWT, including the EBWT, Ferragina et al.'s [22] eXtended BWT (XBWT) for labelled trees, Bowe, et al's. [9] index (BOSS) for de Bruijn graphs, and Sirén et al.'s [56] Generalized Compressed Suffix Array (GCSA) for variation graphs. A directed edge-labelled graph is a Wheeler graph if there exists a total order on the vertices such that

- vertices with in-degree 0 are earliest in the order;
- if (u, v) is labelled a and (u', v') is labelled b with $a \prec b$, then $v < v'$;
- if (u, v) and (u', v') are both labelled a and $u < u'$ then $v \leq v'$.

		F L			F L
0	0	ACATAT	14	+	GATA C
1	0	ACGA T	15	0	GATATA
2	0	AGATAT	16	0	GATT A
3	−	AGAT T	17	+	GATT A
4	0	AGAT T	18	0	TACATA
5	+	ATACAT	19	0	TACG A
6	0	ATAC G	20	+	TAGATA
7	−	ATAGAT	21	0	TAGA T
8	0	ATATAC	22	0	TAGA T
9	0	ATATAG	23	−	TATACA
10	0	ATTA G	24	0	TATAGA
11	−	ATTA G	25	0	TTAG A
12	0	CATATA	26	+	TTAG A
13	−	CGAT A			

Figure 2 The matrix whose rows are the lexicographically sorted rotations of GATTA, TTAGA, TAGATA, GATAC and ATACAT. The EBWT is TTTTTTGTCGGGAACAAAAAATTAAAA with 10 runs.

Figure 3 shows an example of a Wheeler graph with a valid order on the vertices. The ordering is obtained by lexicographically sorting the strings spelling the labels in the upward path from each vertex to the root where the ties are broken deterministically (following an arbitrary order on the branches). For example, vertex 0 has upward path ε, vertex 3 has upward path AG, vertex 30 has upward path TAG and so on. Notice that for directed acyclic graphs such as trees, such order on the vertices can be computed quickly with an adaptation of the doubling algorithm [33].

Once we have a valid order, the standard representation of a Wheeler graph is defined considering the vertices in that order and listing the labels on the outgoing edges of each vertex. In addition, for each vertex we represent its out-degree and in-degree in unary thus obtaining two additional binary arrays. For example, for the graph in Figure 3 the first five vertices have outgoing edges labelled GG T T T TT, so the label array starts with GGTTTTT$\cdots$ and the out-degree bit-array starts with 001010101001$\cdots$. This simple representation, combined with rank and select primitives, supports efficient search and navigation operations on Wheeler graphs. We refer the reader to Prezza's [53] recent survey for a discussion of Wheeler graphs and related results.

Note that the graph in Figure 3 is a labelled tree: indeed its Wheeler Graph representation is equivalent to the output of the XBWT [22] applied to the same tree (details in the full paper). For clarity of presentation in the following we will still refer to the EBWT and XBWT even if they are both special cases of Wheeler graphs.

3 Our contribution

Figure 3 can be seen as a representation of a "genome" GATTAGATACAT and of five "reads" GATTA, TTAGA, TAGATA, GATAC and ATACAT extracted, without errors, from it. Starting with the vertex with rank 28, corresponding to the last symbol of the "genome", and navigating the tree we are able to recover all the individual strings. Notice however, that the XBWT has only 7 runs while the BWT of the "genome" and the EBWT of the "reads" in Figure 1 have 8 and 19 runs, respectively. The EBWT without $ of the reads alone in Figure 2 has 10

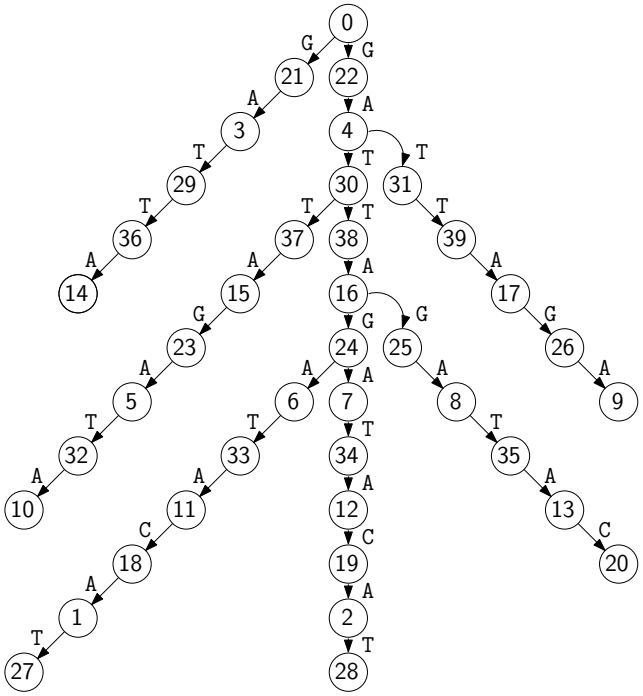


Figure 3 A directed, edge-labelled tree whose vertices are labelled to show it is a Wheeler graph. The XBWT is `GGTTTTTTTTTCCCGGGGAAAAAAAAATTTTAAAAAAAA` with 7 runs.

runs. We refer the reader to Giuliani et al.'s [28, 29] recent papers for a discussion of the impact of the `$` and of the direction of the string on the number of runs in the BWT. The following theorem shows that the example in Figure 3 is not a coincidence: if the "reads" have no errors and they are appended to the reference in the proper positions, then the XBWT has the same number of runs as the BWT of the *reverse* of the "genome".

▶ **Theorem 1.** *Suppose we sample substrings from a string and we form a labelled tree by grafting (appending) the substrings in the same position they were sampled so that all edge labels at the same depth are equal. Then the XBWT of the tree has the same number of runs as the BWT of the reverse of the string.*

Proof. Consider the tree shown in Figure 3. The tree satisfies the hypothesis of Theorem 1 since it was obtained by sampling some substrings from `GATTAGATACAT` and then grafting them onto it such that all the edge labels at the same depth are equal (so a horizontal line always hits edges with only the same label). Clearly, all the labels at the same depth not only are equal, but they have the same upward-path label, which is the prefix preceding the corresponding character in the string. Since the XBWT is built by sorting labels according to the string spelled by their upward path, we see that each symbol of the original string will be adjacent to all reads symbols at the same horizontal level, and that all such symbols are identical. Finally, observe that also in the BWT of the reverse of the string symbols are sorted according to the prefix preceding them; hence the XBWT can be obtained by replacing each symbol in the BWT, except the `$`, by a run of the same symbol and the thesis follows. ◀

Figure 3 and Theorem 1 suggest a new way to compress and index readsets: graft the reads onto a fully or partially assembled genome, or a reference genome if need be, and store the XBWT of the resulting tree. We note that, although assembly-free indexing is a more

general problem, indexing assembled reads is still of practical interest [17]. Many readsets have coverage of 30x or even 50x, which makes them extremely large but should also make run-length compression practical on the XBWTs. If we want to index readsets from several individuals, we can simply graft the reads onto the appropriate assembled genomes and compute the XBWT of the forest, which is also a Wheeler graph.

Theorem 1, provides an extremely good estimate of the number of runs of the XBWT, but it holds under the unrealistic assumption that the reads have no errors. However, we can take advantage of the fact that sequencing by synthesis has an asymmetric error profile: errors are much more likely at the end of a read than at the beginning. The following result shows that errors at the end of the reads have a limited impact to the overall number of runs in the XBWT.

▶ **Theorem 2.** *In the hypothesis of Theorem 1 suppose that the sampled substrings may differ from the reference string and that the average distance from* first *difference (insertion, deletion, or substitution) to the* end *of the substring is δ. Then, with respect to Theorem 1 the XBWT of the tree will have at most 2δ additional runs per substring.*

Proof. Consider a single substring of length ℓ in which the distance between the first difference and the end of the substring is d (we assume $d = 0$ if there are no differences). Reasoning as in the proof of Theorem 1, we see that the first $\ell - d$ symbols of the substring will end up in the same run as the corresponding symbol of the reference string (the one at the same depth in the tree). Each of the other d symbols will, in the worst case, end in the middle of a run of a different symbol thus creating two additional runs. Summing this additional runs over all substrings we get a total number of additional runs upper bounded by 2δ runs per substring. ◀

To guarantee that most of the errors are at the end of the reads, we propose to build two trees: one for the assembled genome and one for its reverse complement. Having two trees means we do not have to reverse and complement half the reads before grafting them onto a single tree: the reversal of the string would be problematic in view of Theorem 2 since it would move an error from the end of the read to its front. We can build two trees with a small additional cost since the alignment algorithm will tell us whether each read aligns to the reference or to its reverse complement.

Assuming our scheme guarantees an improvement in compression we want to be sure the resulting index is also efficient. Prezza [52] recently showed how to generalize Gagie, Navarro and Prezza's [25] results about fast locating from run-length compressed BWTs to run-length compressed XBWTs, at the cost of storing the trees' shapes, which takes a linear number of bits. For trees with far more internal vertices than leaves, however, it is relatively easy to support fast locating in small space, as a corollary of the following theorem.

▶ **Theorem 3.** *Let G be a Wheeler graph and r be the number of runs in a Burrows-Wheeler Transform of G, and suppose G can be decomposed into v edge-disjoint directed paths whose internal vertices each have in- and out-degree exactly 1. We can store G in $O(r + v)$ space such that later, given a pattern P, in $O(|P| \log\log |G|)$ time we can count the vertices of G reachable by directed paths labelled P, and then report those vertices in $O(\log\log |G|)$ time per vertex.*

▶ **Corollary 4.** *Let T be a labelled tree on n vertices obtained by grafting reads onto their assembled genome as described. Let r be the number of runs in the XBWT and let t be the number of reads. We can store T in $O(r + t)$ words of space such that later, given a pattern P, in $O((|P| + k) \log\log n)$ time we can report all the k vertices reachable by paths labelled P.*

We sketch a proof of Theorem 3 in A, although we omit the details because, at least when dealing with short reads, it may be more practical just to descend until we reach a branching node (in which case the pattern is in the assembled genome, not in a read) or a leaf. We have not yet considered carefully whether Nishimoto and Tabei's [50] faster locating can be applied to improve Theorem 3 or Corollary 4.

Before we concentrate on optimizations we should consider two basic questions: are our XBWTs for readsets significantly smaller than their EBWTs in practice and, if so, how can we build them efficiently? Theorem 2 offers some guarantees of compression, but to test how our idea works in practice in Section 5 we build the XBWT and EBWT for a real, high-coverage readset and see how the numbers of runs in them compare. In Section 4 instead we face the problem of the efficient construction of XBWTs for large datasets.

4 XBWT via Prefix Free Parsing

The problem of building the XBWT for a set of reads as described in Section 3 is non trivial because the input typically consists in tens of gigabytes of data and we cannot make use of the available algorithms [1, 3] which are designed to work in RAM. However, the fact that reads are copies (possibly with errors), of portions of a relatively small reference suggests that the overall amount of information content is relatively small. Therefore we decided to compute the XBWT using the technique of Prefix Free Parsing (PFP) that has been successfully utilized for computing the BWT for large collections of genomes from individuals of the same species. Our implementation was done in C++ and is available on `https://github.com/fnareoh/Big_XBWT`. Note that our algorithm does not take as input a labelled tree, but rather a reference genome and a set of reads aligned to that genome (in the format of a `.bam` file); the alignment implicitly defines a labeled tree as described in Section 3.

In the PFP construction of the BWT the input is parsed into overlapping phrases using context-triggered piecewise hashing [7]. If the input contains many repetitions, the use of context-triggered hashing ensures that the parsing will contain a relatively small number of distinct phrases. The actual construction of the BWT is done using only the dictionary of distinct phrase and the parse (which describes how the dictionary phrases can be used to reconstruct the input). For repetitive datasets the dictionary and the parse fit in RAM even when the original input does not. Unmodified, however, PFP does not work well on readsets since the phrases generated at the beginning and end of each read will likely be unique. As a result, the dictionary will be quite large and the algorithm inefficient. To prevent this, we extend the reads forward and backward so they begin and end with complete phrases. The extension is done using the symbols in the reference immediately before and after the position where the read aligns, so that the phrases are likely to be not unique (if the read has no errors the phrases will be exactly the same generated when parsing the reference). Although this technique maintains the dictionary small, the tricky part is to exclude these extensions when computing the actual XBWT.

Summing up, our implementation is divided in three main phases. In the first phase we partition the reference and the reads into phrases; the set of distinct phrases is called the *dictionary* and the way phrases form the reference and the reads is called the *parse*. We use the extension trick mentioned before, and ,if the reference and the reads are similar, the dictionary will be relatively small. In the second phase we compute the XBWT of the parse. Since phrases are relatively large, the number of symbols in the parse is much smaller than in the original input, so the parse fits in RAM and the computation can be done using a doubling algorithm [33]. Finally, in the third phase we recover the XBWT of the input from the XBWT of the parse. The details of the three phases are given below.

4.1 Construction of the Dictionary and the Parse

We start by scanning the reference as in the PFP BWT construction algorithm. The algorithm takes as input parameters a window size w, and a modulo m. We slide a window of length w over the text, at each step computing the Karp-Rabin fingerprint [32] of the window. We define a terminating windows as a window with Karp-Rabin fingerprint equal to zero modulo m. Terminating windows decompose the text into overlapping phrases: each phrase is a minimal substring that begins and ends with a terminating window. Note that each terminating window is a suffix of the current phrase and the prefix of the next phrase so consecutive phrases have a size-w overlap. Note that defining phrases using terminating windows ensures that no phrase is a prefix (or a suffix) of another phrase, hence the name "prefix free parsing".

In addition to keeping track of window fingerprints, we also maintain a different hash $h(p_i)$ of the current phrase p_i. For simplicity in the following we assume distinct phrases always have distinct hashes, if not we detect it and crash. At the end of this scanning phase, the reference has been parsed into the (overlapping) phrases $p_1, p_2, \ldots, p_z$. We build a vector $S[1, z]$ storing for each phrase p_i its starting position s_i in the reference and its hash $h(p_i)$. We also build as we go the dictionary that associate to each hash value $h(p_i)$ the corresponding phrase p_i (stored as a simple string) and $occ(p_i)$ the number of occurrences of that phrase. We will later also need the length of each phrase but we don't store it explicitly, just deduce it from the string stored in the dictionary.

After parsing the reference, we process the reads one by one. From the file of aligned reads, we obtain both the read r as a string and the position l where the read aligns to the reference. We binary search in S for the rightmost phrase p_s that starts before position l and for the leftmost phrase p_e that ends after position $l + |r| - 1$. Let p'_s (resp. p'_e) denote the prefix (resp. suffix) of p_s (resp. p_e) ending (resp. starting) immediately before (resp. after) position l (resp. $l + |r| - 1$). We define the extended read $r_{ext} = p'_s \cdot r \cdot p'_e$ where $\cdot$ here denotes string concatenation. We slide a window onto r_{ext}, decomposing it into phrases, as we did for the reference. Since r_{ext} starts and ends with a terminating window the phrases we add while parsing r_{ext} still form a prefix-free parsing. However, as we do not want to index the whole r_{ext} in the final XBWT, for each read we keep track and store to disk the starting and ending position of r in r_{ext}.

When processing the reads we continue adding the hashes of the phrases to the end parse, using a special value as separator between reads. If we parse a new phrase, we add it to the dictionary. However, as previously pointed out, the phrases coming from the extended reads are likely to be equal to phrases in the reference so we expect the dictionary not to grow significantly (the dictionary would not grow at all if all the reads were substrings of the reference). From the starting and ending position of the original read in the extended read we deduce for each phrase what characters are part of the original read (the reads without extensions) and we store a starting and ending position for each phrase.

Once all the reads have been processed, we sort the phrases in the dictionary in reverse lexicographic order and we output a new parse where each hash of phrase is replaced by its reverse lexicographic rank, the separator symbol is replaced by the number of phrases plus one. To summarize, at the end of this phase we have produced the following output files:

1. `file.dict`: the dictionary in co-lexicographic order;
2. `file.occ`: the frequency of each phrases;
3. `file.parse`: the parse with each phrase represented by its co-lexicographic rank;
4. `file.limits`: the starting and ending position of the original input (reads without extension) in each phrase.

4.2 XBWT of the Parse

The main goal of this phase is to construct the XBWT of the parse, using the co-lexicographic rank as meta-characters. To this end we load the parse on RAM, reconstruct its tree structure, and compute the XBWT of this tree via a doubling algorithm [33]. Then, rather than storing the XBWT as is, we construct an inverted list as this structure will be more appropriate for the next phase. For each phrase p_i we store the list of XBWT positions where p_i appears. The size of the inverted list for p_i is equal to its frequency; since frequencies were computed in the first phase, we can output the inverted list as a plain concatenation of positions.

In this phase we also permute the limits (the starting and ending position in the original input) of each phrase according to their order in the XBWT. This way, in the next phase, with the inverted list, we can easily access the limit of any given phrase in the parse. In this phase, we also compute and write to disk for every phrase, the list of phrases (with multiplicities) that immediately follow in the parse. This list will be used to index the characters that precede a full word. However because we only want to index the characters that are in the original input, we only add it after checking the limits. Finally, because we are not storing special characters to mark the end of a read or of the reference (as they would break runs), we construct a bit vector marking such positions and we permute it according to the XBWT order. To summarize, at the end of this phase we have produced the following output files:

1. `file.dict`: the dictionary of the reversed phrases (from the first phase).
2. `file.occ`: the frequency of each phrases (from the first phase).
3. `file.ilist`: the inverted list of the parse.
4. `file.xbwt_limits`: the limits of the phrases in XBWT order.
5. `file.xbwt_end`: markers of the phrases where a read or reference ends in XBWT order.
6. `file.full_children`: for every word, the list of words that follows it.

4.3 Building the final XBWT

This is the final phase where we compute the XBWT of the reference and of the readset. We start by sorting lexicographically the suffixes of the strings in the dictionary D. At this stage the dictionary D contains the phrases reversed, so this is equivalent to sort in reverse lexicographic order the prefixes of all phrases. We ignore the suffixes of length $\leq w$ as they correspond to the terminating window which also belongs to the previous phrase. The sorting is done by the gSACAK algorithm [43] which computes the SA and LCP array for the set of dictionary phrases. We scan the sorted elements of D, for s a proper suffix, there are two cases, all the elements in D which have s as a proper suffix have the same preceding character, in this case we add it the correct number of times using the frequency of each phrase. In the other case, we use a heap to merge the inverted list writing the appropriate characters accordingly. Here when writing a character we first check that the suffix length is between the limits and only write it to file if it does. We also check if the character to be added is the last of its sequence (read or reference), if so output a 1 to signal the end of a sequence, else 0. When finding a suffix s' that corresponds to an entire phrase, we use the children file to output the character at the start of the following phrase. At the end of this phase we have written to disk a file with the XBWT of the reference and readset as well as a bit vector marking which positions are the last character of a read or a genome. To summarize, in this phase we use `file.dict`, `file.occ`, `file.ilist`, `file.full_children` and `file.xbwt_limits`; all other files can be discarded. We output the XBWT in plain text as `file.bwt` and `file.is_end` is the compressed bit vector marking the end of reads.

5 Experiments

In this section we present a first experimental evaluation of our XBWT-based approach for compressing a set of aligned reads and we compare it with the known methods based on the EBWT. We compare ourselves to the EBWT and not other compression tools for aligned readset as our long-term goal is to create an index and not just compression. Recall that our implementation and experimental pipeline is available on github.com/fnareoh/Big_XBWT. For simplicity we compare the numbers of runs produced by the different algorithms. The actual compression depends on the algorithm used for encoding the run lengths: preliminary experiments with the γ encoder show that the number of runs is a good proxy for measuring the actual compression. An accurate comparison of the time efficiency is left as a future work: we only compared the number of runs produced by our XBWT with the number of runs produced by the EBWT and some of its variants. Note that our implementation computes the XBWT of the reference genome and the readset (as described in the previous section), while the EBWT and its variants were applied only to the readset. We computed all EBWT variants using ropeBWT2 [40]; in addition to plain EBWT we also tested 2 heuristics that reorder the reads to reduce the number of runs in the EBWT: Spring [11] and reverse lexicographic order (RLO) [14], the latter obtained using the option `-s` in ropeBWT2. Since our XBWT implementation does not use the `$` symbol, for a fair comparison we measured the number of runs with and without the `$` for EBWT, Spring+EBWT and RLO+EBWT (therefore ignoring for all algorithms the extra cost of implicitly encoding the ending position of each string). In our tests, we used the following readsets:

- **E.coli** and **S.aureus** from the single-cell dataset [12], the references used are those linked on the single-cell website[1],[2].
- **R.sphaeroides** We have HiSeq and MiSeq sequencing, raw and trimmed versions of the reads from the GAGE-B dataset [44]. The reference used is the longest contig assembled by MSRCA v1.8.3 [59] as it was the most accurate assembler according to the Gage-b companion paper [44]. We only considered the longest contig because our implementation doesn't handle forests of trees yet.
- **Human Chromosome 19** We used as a reference Chromosome 19 from the CHM1 human assembly [57] and one of the HiSeq 2000 readsets[3] used to compute that assembly, considering only the reads that aligned with the reference.

None of those readsets are aligned, so we used bwa mem [42] to align them to the chosen reference. In this preliminary experiments we discarded the reads that bwa aligned with the reverse-complement of the reference genome. As mentioned in Section 3 our final prototype will build an XBWT of the tree with the reference and of the tree of the reversed-complemented reference. In Table 1, we present statistics on the readsets we used: those statistics where computed only on the reads that aligned forward to the reference.

Preliminary experiments, not reported here, show that removing the `$` in the EBWT (all variants) reduces the number of runs between 2.7% and 29.2%. Consequently, we focus our analysis on the comparison of Plain (no read reordering) EBWT (without dollars), SPRING+EBWT (without dollars), RLO+EBWT, with and without `$` and XBWT.

The results of this comparison are reported in Figures 4a and 4b. They show that in general the plain EBWT performs worse followed by the SPRING reordering, RLO ordering with dollars then RLO ordering without dollars and finally XBWT performs best. XBWT yields a smaller number of runs than RLO+EBWT (with or without `$`) on all datasets,

[1] `https://www.ncbi.nlm.nih.gov/nuccore/NC_000913`
[2] `https://www.ncbi.nlm.nih.gov/nuccore/87125858`
[3] `https://www.ncbi.nlm.nih.gov/sra/SRX966833[accn]`

Table 1 Statistics on each dataset used in the experiments. Those statistics where computed only on the reads that aligned forward to the reference. We call sequencing error (or simply error) any difference between the genome and the reads. The coverage is simply defined as the total number of base-pairs in the reads compared to the number of base-pairs in the reference. The average distance between the first sequencing error and the end of the read and the end is computed considering that for error less read this distance is 0. Note that this parameter is exactly δ in Theorem 2.

Dataset	Number of reads	Read length	Coverage	Avg. dist. from the first sequencing err. to the end	Prop. of reads without seq. error	Error rate
E.coli [12]	14139182	100	304×	13	57.30%	0.01%
S.aureus [12]	26654420	100	927×	7	88.79%	0.01%
Human Chr19 [57]	34167479	100	57×	15	71.62%	0.01%
R.sphaeroides [44]						
HiSeq raw	166820	101	46×	27	31.34%	0.04%
HiSeq trimmed	134207	up to 101	37×	6	83.26%	0.01%
MiSeq raw	23102	251	24×	122	0.25%	0.15%
Miseq trimmed	20046	up to 251	20×	29	63.55%	0.03%

although the number is comparable on some datasets this is still a significant improvement considering that RLO+EBWT already has far less run than the EBWT baseline. On the Chr19 dataset, using RLO+EBWT-no-$ over plain BWT-no-$ (not reported in Figure 4a) reduced the number of runs by 49%; using the XBWT reduced the number of runs by an additional 16%. On S.aureus and E.coli the reduction between RLO+EBWT-no-$ and XBWT is of only 3% and 8% respectively.

The R.sphaeroides datasets are especially interesting as they involve two NGS technologies that generate reads of different lengths, different coverages, and with different error profiles. We can first notice that our method brings greater benefits on the HiSeq sequencing which has smaller reads with less errors that are located towards the end of the string. This is an experimental validation of the statement of Theorem 2. We can also observe the effect of trimming the reads on the number of runs. On the HiSeq sequencing, trimming reduces the coverage only from 46x to 37x but yields a reduction in the number of XBWT runs by 86%. Note that, as a result, on HiSeq trimmed, the number of XBWT runs is less than half the number of runs in plain RLO+EBWT.

6 Application to the JST

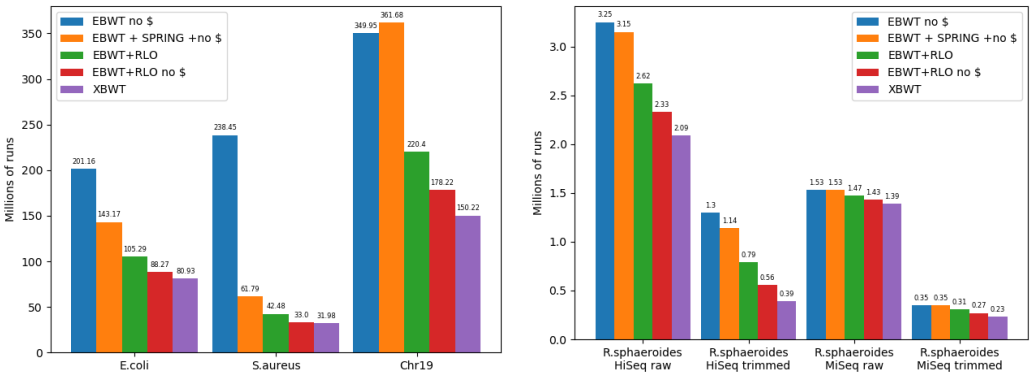
From a certain angle, Figure 3 is reminiscent of Figure 5, from Rahn, Weese and Reinert's [54] paper on their Journaled String Tree (JST). This raises the question of whether the XBWT and JST can be used to improve the space usage of the hybrid index [20, 26, 21] and eventually the PanVC [58] pan-genomic read aligner, which is based on the hybrid index.

Figure 5 shows a JST supporting search for patterns of length up to 4 in four aligned sequences: the reference

$r =$ TAGCGTAGCAGCTATGAGGAGGACCGAGTT

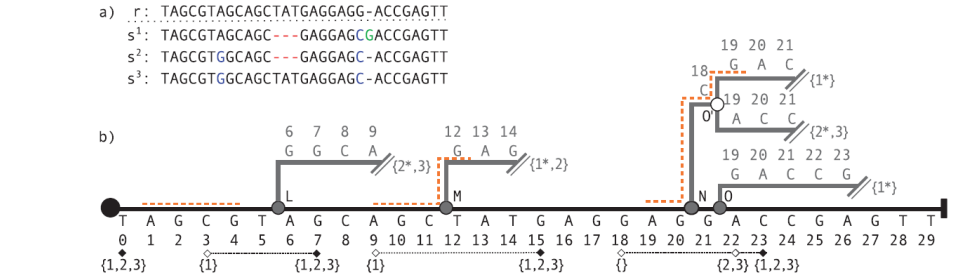
and three others,

$s^1 =$ TAGCGTAGCAGCGAGGAGCGACCGAGTT,
$s^2 =$ TAGCGTGGCAGCGAGGAGCACCGAGTT,
$s^3 =$ TAGCGTGGCAGCTATGAGGAGCACCGAGTT.



(a) E.coli, S.aureus, Human Chr19. **(b)** R.sphaeroides.

Figure 4 Comparison of run-lengths compression by RLO+EBWT with and without $ and XBWT on various species (4a) and on two sequencing of R.sphaeroides (HiSeq and MiSeq) and for reads both raw and trimmed (4b).



Online search of the pattern $p = AGCG$ over a JST depicted in (**b**) representing the four aligned sequences in (**a**), where r is chosen as the reference. The window length is 4. The bullets on the black line represent the branch-nodes: $L = (6, G, \{2, 3\}, R)$; $M = (12, TAT, \{1, 2\}, D)$; $N = (21, C, \{1, 2, 3\}, R)$; $O = (22, G, \{1\}, I)$. Below is plotted the reference with the reference coordinates. Branches are indicated by gray lines labeled with the respective sequence context depending on the window length and with the virtual positions of the chosen proxy (marked sequence number). The sets at the end denote the coverage. O' is a split-node induced by O in the branch of N. The dotted lines capped with diamonds below the reference positions represent the ranges in which invalid paths are suppressed. The orange (dashed) lines represent matches of the pattern p

Figure 5 An illustration of a JST [54].

The straight branch of the tree running along the bottom of the figure is labelled with r, and the other branches indicate places where the other sequences differ from r. The other branches end just before a window of size 4 sliding over their sequences matches an aligned window of size 4 sliding over r. For example, the first branch ends at position 9 because a sliding window of length 4 over positions 7 to 10 of sequences s^2 and s^3 (that is, containing the characters in columns 7 to 10 and the rows for s^2 and s^3 in the alignment shown at the top right in the figure), matches a sliding window of length 4 over positions 7 to 10 in r (that is, containing the characters in columns 7 to 10 and the row for r in the alignment).

Suppose we are looking for the pattern $p = \texttt{AGCG}$: considering the circle at the left as the root, p occurs 3 times as a substring (marked in orange) of root-to-leaf paths, and we can find those occurrences using a depth-first traversal of the tree. Since the sequences are similar, such a traversal is faster than running a sliding window over each sequence separately. If we find an occurrence of p in the tree that ends at a node not in the branch for r, then we have found occurrences in each of the sequences labelling the leaves in that node's subtree. If we find an occurrence of p in the branch for r, then we have found occurrences in r and possibly other sequences. Unfortunately this case is not illustrated in the figure, but if we

were looking for `GTAG` then the occurrence at position 4 in r would have a corresponding occurrence in s^1 but not in s^2 or s^3; this is shown by the dashed line between 3 and 7, with $\{1\}$ at the left end indicating that s^1 matches r between 3 and 7 and $\{1, 2, 3\}$ indicating that s^1, s^2 and s^3 all match r from 7 onward (until the next such interval starts at 9).

The hybrid index is conceptually similar to the JST, but the former is an index and the latter performs pattern matching by scanning the tree sequentially. To build the hybrid index supporting search for patterns of length up to 4 in r, s^1, s^2, s^3, we first build a string kernel consisting of r and substrings from s^1, s^2, s^3 that contain all the characters within distance 3 of variations from r, all separated by copies of a special symbol `$`:

`TAGCGTAGCAGCTATGAGGAGGACCGAGTT$CGTGGCA$AGCGAG$GAGCGACC$GAGCACC`.

Any substring of length at most 4 of the the four sequences r, s^1, s^2, s^3 is a substring of the string kernel, and any substring of length at most 4 of the string kernel that does not include a copy of `$` is a substring of at least one of those sequences. We then build an FM-index for the string kernel, with auxiliary data structure that allow us to quickly map occurrences of a pattern in the string kernel to occurrences in the sequences.

It seems interesting that the string kernel for the four sequences in Figure 5 has more characters than the JST: on top of r, the string kernel has a substring `$CGTGGCA` and the JST has a branch labelled `GGCA`; the string kernel has `$AGCGAG` and the JST has `GAG`; the string kernel has `$GAGCGACC` and the JST has `CGAC` and `GACCG` (a tie in this one case); the string kernel has `$GAGCACC` and the JST has `CACC` (with the first `C` shared with the branch ending `CGAC`). This difference is because the string kernel stores copies of the characters both before and after variation sites, whereas the JST stores copies only of the characters after them. If we build an index using the XBWT of the JST, therefore, it may be smaller than the hybrid index while having the same basic functionality. We leave exploring this possiblity as future work.

References

1. Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 911–930. Society for Industrial and Applied Mathematics, 2020.
2. Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: a succinct colored de Bruijn graph representation. In *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
3. Uwe Baier, Thomas Büchler, Enno Ohlebusch, and Pascal Weber. Edge minimization in de Bruijn graphs. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2020, Snowbird, UT, USA, March 24-27, 2020*, pages 223–232. IEEE, 2020. doi:10.1109/DCC47342.2020.00030.
4. Hideo Bannai, Travis Gagie, and I Tomohiro. Refining the r-index. *etical Computer Science*, 812:96–108, 2020.
5. Markus J Bauer, Anthony J Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theoretical Computer Science*, 483:134–148, 2013.
6. Jason W Bentley, Daniel Gibney, and Sharma V Thankachan. On the complexity of BWT-runs minimization via alphabet reordering. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
7. Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms for Molecular Biology*, 14(1):1–15, 2019.

8 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*, pages 225–235. Springer Berlin Heidelberg, 2012.

9 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *International workshop on algorithms in bioinformatics (WABI)*, pages 225–235. Springer, 2012.

10 Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer, 1994.

11 Shubham Chandak, Kedar Tatwawadi, Idoia Ochoa, Mikel Hernaez, and Tsachy Weissman. SPRING: a next-generation compressor for FASTQ data. *Bioinformatics*, 35(15):2674–2676, 2018. doi:10.1093/bioinformatics/bty1015.

12 Hamidreza Chitsaz, Joyclyn Yee-Greenbaum, Glenn Tesler, Mary-Jane Lombardo, Christopher Dupont, Jonathan Badger, Mark Novotny, Douglas Rusch, Louise Fraser, Niall Gormley, Ole Schulz-Trieglaff, Geoffrey Smith, Dirk Evers, Pavel Pevzner, and Roger Lasken. Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. *Nature biotechnology*, 29:915–21, September 2011. doi:10.1038/nbt.1966.

13 Anthony J Cox, Markus J Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinformatics*, 28(11):1415–1419, 2012.

14 Anthony J. Cox, Markus J. Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinform.*, 28(11):1415–1419, 2012. doi:10.1093/bioinformatics/bts173.

15 Anthony J. Cox, Tobias Jakobi, Giovanna Rosone, and Ole B. Schulz-Trieglaff. Comparing DNA sequence collections by direct comparison of compressed text indexes. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2012.

16 D. Díaz-Domínguez and G. Navarro. A grammar compressor for collections of reads with applications to the construction of the BWT. In *Proc. 31th Data Compression Conference (DCC)*, 2021. To appear.

17 Dirk D Dolle, Zhicheng Liu, Matthew Cotten, Jared T Simpson, Zamin Iqbal, Richard Durbin, Shane A McCarthy, and Thomas M Keane. Using reference-free compressed data structures to analyze sequencing reads from thousands of human genomes. *Genome research*, 27(2):300–309, 2017.

18 Lavinia Egidi, Felipe A Louza, Giovanni Manzini, and Guilherme P Telles. External memory BWT and LCP computation for sequence collections with applications. *Algorithms for Molecular Biology*, 14(1):1–15, 2019.

19 Lavinia Egidi and Giovanni Manzini. Lightweight merging of compressed indices based on BWT variants. *Theoretical Computer Science*, 812:214–229, 2020.

20 Héctor Ferrada, Travis Gagie, Tommi Hirvola, and Simon J Puglisi. Hybrid indexes for repetitive datasets. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130137, 2014.

21 Héctor Ferrada, Dominik Kempa, and Simon J Puglisi. Hybrid indexing revisited. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–8. SIAM, 2018.

22 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and Senthilmurugan Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM (JACM)*, 57(1):1–33, 2009.

23 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM (JACM)*, 52(4):552–581, 2005.

24 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical computer science*, 698:67–78, 2017.

25 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM (JACM)*, 67(1):1–54, 2020.

26 Travis Gagie and Simon J Puglisi. Searching and indexing genomic databases via kernelization. *Frontiers in Bioengineering and Biotechnology*, 3:12, 2015.

27 Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology*, 36(9):875–879, 2018.

28 Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Nicola Prezza, Marinella Sciortino, and Anna Toffanello. Novel results on the number of runs of the Burrows-Wheeler transform. In Tomáš Bureš, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdziński, Claus Pahl, Florian Sikora, and Prudence W.H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science*, pages 249–262, Cham, 2021. Springer International Publishing.

29 Sara Giuliani, Zsuzsanna Lipták, Francesco Masillo, and Romeo Rizzi. When a dollar makes a BWT. *Theoretical Computer Science*, 2019.

30 Lilian Janin, Ole Schulz-Trieglaff, and Anthony J Cox. BEETL-fastq: a searchable compressed archive for DNA reads. *Bioinformatics*, 30(19):2796–2801, 2014.

31 Juha Kärkkäinen, Giovanni Manzini, and Simon J Puglisi. Permuted longest-common-prefix array. In *Annual Symposium on Combinatorial Pattern Matching*, pages 181–192. Springer, 2009.

32 R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

33 Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, page 125–136, 1972. doi:10.1145/800152.804905.

34 Alice M Kaye and Wyeth W Wasserman. The genome atlas: Navigating a new era of reference genomes. *Trends in Genetics*, 2021.

35 Yuichi Kodama, Martin Shumway, and Rasko Leinonen. The sequence read archive: explosive growth of sequencing data. *Nucleic acids research*, 40(D1):D54–D56, 2012.

36 Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini. Efficient construction of a complete index for pan-genomics read alignment. *Journal of Computational Biology*, 27(4):500–513, 2020.

37 Ben Langmead. Algorithms for DNA sequencing: Base calling and sequencing errors, May 2015. URL: https://www.youtube.com/watch?v=U4QnpciIJhM&list=PL2mpR0RYFQsBiCWVJSvVAO3OJ2t7DzoHA&index=10.

38 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357, 2012.

39 Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome biology*, 10(3):1–10, 2009.

40 Heng Li. Fast construction of FM-index for long sequence reads. *Bioinform.*, 30(22):3274–3275, 2014. doi:10.1093/bioinformatics/btu541.

41 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

42 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.

43 Felipe A. Louza, Simon Gog, and Guilherme P. Telles. Inducing enhanced suffix arrays for string collections. *Theoretical Computer Science*, 678:22–39, 2017. doi:10.1016/j.tcs.2017.03.039.

44 Tanja Magoc, Stephan Pabinger, Stefan Canzar, Xinyue Liu, Qi Su, Daniela Puiu, Luke J. Tallon, and Steven L. Salzberg. GAGE-B: an evaluation of genome assemblers for bacterial organisms. *Bioinform.*, 29(14):1718–1725, 2013. doi:10.1093/bioinformatics/btt273.

45 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I Tomescu. *Genome-scale algorithm design*. Cambridge University Press, 2015.

46 S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An extension of the Burrows–Wheeler transform. *Theoretical Computer Science*, 387(3):298–312, 2007. doi:10.1016/j.tcs.2007.07.014.

47 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An extension of the Burrows-Wheeler transform. *Theoretical Computer Science*, 387(3):298–312, 2007.

48 Martin D Muggli, Alexander Bowe, Noelle R Noyes, Paul S Morley, Keith E Belk, Robert Raymond, Travis Gagie, Simon J Puglisi, and Christina Boucher. Succinct colored de Bruijn graphs. *Bioinformatics*, 33(20):3181–3187, 2017.

49 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

50 Takaaki Nishimoto and Yasuo Tabei. Faster queries on BWT-runs compressed indexes. *arXiv preprint*, 2020. arXiv:2006.05104.

51 Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. *Algorithmica*, 80(7):1986–2011, 2018.

52 Nicola Prezza. On locating paths in compressed tries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 744–760. SIAM, 2021.

53 Nicola Prezza. Subpath queries on compressed graphs: A survey. *Algorithms*, 14(1):14, 2021.

54 René Rahn, David Weese, and Knut Reinert. Journaled string tree - a scalable data structure for analyzing thousands of similar genomes on your laptop. *Bioinformatics*, 30(24):3499–3505, 2014. doi:10.1093/bioinformatics/btu438.

55 Julian Seward. bzip2 and libbzip2, 1996. avaliable at http://www.bzip.org.

56 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

57 Karyn Meltz Steinberg, Valerie A. Schneider, Tina A. Graves-Lindsay, Robert S. Fulton, Richa Agarwala, John Huddleston, Sergey A. Shiryev, Aleksandr Morgulis, Urvashi Surti, Wesley C. Warren, Deanna M. Church, Evan E. Eichler, and Richard K. Wilson. Single haplotype assembly of the human genome from a hydatidiform mole. *Genome Research*, 24(12):2066–2076, 2014. doi:10.1101/gr.180893.114.

58 Daniel Valenzuela, Tuukka Norri, Niko Välimäki, Esa Pitkänen, and Veli Mäkinen. Towards pan-genome read alignment to improve variation calling. *BMC genomics*, 19(2):123–130, 2018.

59 Aleksey V. Zimin, Guillaume Marçais, Daniela Puiu, Michael Roberts, Steven L. Salzberg, and James A. Yorke. The MaSuRCA genome assembler. *Bioinformatics*, 29(21):2669–2677, August 2013. URL: https://academic.oup.com/bioinformatics/article-pdf/29/21/2669/18533361/btt476.pdf.

A Proof Sketch for Theorem 3

Let G be a Wheeler graph with the vertices sorted according to the permutation π. A Burrows-Wheeler Transform (BWT) of G according to π is a permutation of G's edge labels such that, for any pair of edges $e = (u, v)$ and $e' = (u', v')$ labelled a and a' respectively, if $u < u'$ then a precedes a' in that permutation. For convenience, we assume that the labels of each vertex's out-edges appear in the order in π of their destinations. Notice there may be many BWTs for G because it may have many permutations π satisfying the Wheeler graph conditions.

Let B be a BWT of G according to π. By the definition of a Wheeler graph, for any pattern P over the alphabet of edge labels, the vertices reachable by directed paths labelled P form an interval in π. Moreover, if we store a rank data structure for B and partial sum data structures for the frequencies of the distinct edge labels and the vertices' in- and out-degrees, then given P we can find its interval in $O(|P| \log \log |G|)$ time. Let r is the number of runs

(i.e., maximal non-empty unary substrings) in B and suppose G can be decomposed into υ edge-disjoint directed paths whose internal vertices each have in- and out-degree exactly 1. Then these data structures take a total of $O(r + \upsilon)$ space, measured in words.

Let D be a such decomposition of G and n be the number of vertices in G, and assume the vertices are assigned numeric identifiers from 0 to $n - 1$ such that if (u, v) is an edge and neither u nor v is an endpoint of a path in D, and u has identifier i, then v has identifier $i + 1$. Notice these identifiers are not necessarily the vertices' ranks in π. For convenience, we assume that even though G is a multigraph, the number of edges is polynomial in n, so $\log\log|G| = O(\log\log n)$. We show how, still using $O(r + \upsilon)$ space, after we have found the interval for P we can then report the vertices in it using $O(\log\log n)$ time for each one.

We first prove a generalization of Bannai, Gagie and I's version [4] of Policriti and Prezza's Toehold Lemma [51], that lets us report the last vertex in the interval for P. We then define a generalization of Kärkkäinen, Manzini and Puglisi's ϕ function [31], that maps each vertex's identifier to the identifier of its predecessor in π. Finally, we give a generalization of a key lemma behind Gagie, Navarro and Prezza's r-index [25], that lets us compute our generalized ϕ function with $O(r + \upsilon)$-space data structures. Combined, these three results yield a generalized r-index for Wheeler graphs.

A.1 Generalized Toehold Lemma

For any pattern $P[0..m - 1]$, the interval for the empty suffix $P[m..m - 1]$ of P is all of π, because every vertex is reachable by an empty path. Assume we have found the interval $\pi[s_{i+1}, e_{i+1}]$ for $P[i + 1..m - 1]$ and now we want to find the interval $\pi[s_i, e_i]$ for $P[i..m - 1]$. With the partial sum data structure for the vertices' out-degrees, in $O(\log\log n)$ time we can find the interval in B containing the labels of the edges leaving the vertices in $\pi[s_{i+1}, e_{i+1}]$.

By the definition of a Wheeler graph, the edges labelled with the first and last occurrences of $P[i]$ in that interval in B, lead to the first and last vertices in the interval $\pi[s_i, e_i]$ for $P[i..m - 1]$. Using the partial sum data structures for the frequencies of the distinct edge labels and the vertices in-degrees, in $O(\log\log n)$ time we can find the ranks s_i and e_i in π of those first and last vertices in $\pi[s_i, e_i]$. It follows that in $O(\log\log n)$ time we can find $\pi[s_i, e_i]$ from $\pi[s_{i+1}, e_{i+1}]$; therefore, by induction, we can find the interval for P in $O(|P|\log\log n)$ time. We can count the vertices in that interval in the same asymptotic time by simply returning the size of the interval.

To be able to find the identifier of the last vertex in the interval for P, for each edge (u, v) we store u's and v's identifiers if any of the following conditions hold:

- (u, v)'s label a is the last label in a run in B;
- either u or v is an endpoint of a path in D;
- the vertex that follows u in π has out-degree 0.

We store a select data structure for B, a bitvector marking the labels a in B for whose edges (u, v) we have u's and v's identifiers stored, and a hash table mapping the position in B of each marked label a to the identifiers of its edge's endpoints. This again takes a total of $O(r + \upsilon)$ space.

By querying the rank data structure, the select data structure, the bitvector and the hash table in that order, we can find the identifier of the vertex reached by the edge labelled by the last copy of $P[m - 1]$ in B. By the definition of a Wheeler graph, this is the last vertex in the interval $\pi[s_{m-1}, e_{m-1}]$ for $P[m - 1]$. Assume we have found the interval $\pi[s_{i+1}, e_{i+1}]$ for $P[i + 1..m - 1]$ and the identifier of the last vertex u in that interval, and now we want to find the interval $\pi[s_i, e_i]$ for $P[i..m - 1]$ and the identifier of the last vertex v in that interval. We can find $\pi[s_i, e_i]$ as described above, so we need only say how to find v's identifier.

With the partial sum data structure on the vertices' out-degree and the rank data structure, in $O(\log\log n)$ time we can check whether u has an outgoing edge labelled $P[i]$. If it does then, of all its out-edges labelled $P[i]$, the one whose label appears last in B goes to v. By our assumption of how the vertices are assigned their identifiers, if neither u nor v are endpoints of a path in D, then v's identifier is u's identifier plus 1. If either u or v is an endpoint of a path in D, then we have v's identifier stored and we can use the hash table to find it from the position in B of the last label $P[i]$ on one of u's out-edges, again in $O(\log\log n)$ time.

If u does not have an outgoing edge labelled $P[i]$ then we can use the rank data structure to find the last copy of $P[i]$ in B that labels an edge leaving a vertex in $\pi[s_{i+1}, e_{i+1}]$. By the definition of a Wheeler graph, this edge (u', v) goes to v. Unlike in a BWT of a string, however, its label may not be the end of a run in B: u could have out-degree 0, u' could immediately precede u in π and the last of its outgoing edges' labels in B could be a copy of $P[i]$, and the first label in B of an outgoing edge of the successor of u in π could also be a copy of $P[i]$. This is why we store v's identifier if the vertex that follows u' in π has out-degree 0. If (u', v)'s label is the end of a run in B, of course, then we also have v's identifier stored. In both cases we use $O(\log\log n)$ time, so from the interval $\pi[s_{i+1}, e_{i+1}]$ for $P[i+1..m-1]$ and the identifier of the last vertex u in that interval, in $O(\log\log n)$ time we can compute the interval $\pi[s_i, e_i]$ for $P[i..m-1]$ and the identifier of the last vertex v in that interval. Therefore, by induction, in $O(|P|\log\log n)$ time we can find the interval for P and the identifier of the last vertex in that interval.

▶ **Lemma 5.** *We can store G in $O(r+v)$ space such that in $O(|P|\log\log n)$ time we can find the interval for P and identifier of the last vertex in that interval.*

A.2 Generalized ϕ

For a string S, the function ϕ takes a position i in S and returns the starting position of the suffix of S that immediately precedes $S[i..|S|-1]$ in the lexicographic order of the suffixes. In other words, ϕ takes the value in some cell of suffix array of S and returns the value in the preceding cell. Given a pattern P, if we can find the interval of the suffix array containing the starting positions of occurrences of P in S, and the entry in the last cell in that interval, then by iteratively applying ϕ we can report the starting positions of all the occurrences of P. This is the idea behind the r-index for strings, which uses a lemma saying it takes only space proportional to the number of runs in the BWT of S to store data structures that let us evaluate ϕ in $O(\log\log|S|)$ time.

We generalize ϕ to Wheeler graphs by redefining it such that it takes the identifier of some vertex u in G and returns the identifier of the vertex that immediately precedes u in π. (For our purposes here, it is not important how ϕ behaves when given the identifier of the first vertex in π.) Given a pattern P, if we can find the interval in π containing the vertices in G reachable by directed paths labelled P, and the identifier of the last vertex in that interval, then by iteratively applying ϕ we can report the identifiers of all those vertices.

Let J be the set that contains u's identifier if and only if any of the following conditions hold:

- u has out-degree not exactly 1;
- u has a single outgoing edge (u, v) but v has in-degree not exactly 1;
- the predecessor u' of u in π has out-degree not exactly 1;
- u' has a single outgoing edge (u', v') but v' has in-degree not exactly 1;
- the edges (u, v) and (u', v') have different labels.

We store a successor data structure for J and, if u's identifier is in J, then we store with it as satellite data the identifier of u's predecessor u' in π. Notice u's identifier is in J only if at least one of u or u' or v or v' is the endpoint of a path in D, or the label of (u', v') is the the last in a run in B and the label of (u, v) is the first in the next run. It follows that we can use $O(r + v)$ space for the successor data structure and have it support queries in $O(\log \log n)$ time.

Suppose we know the identifier of some vertex u with identifier i that is immediately preceded by u' in π with identifier i'. If $u \in J$ then we have i' stored as satellite data with $\succ (i) = i$. If $u \notin J$, then u has a single outgoing edge (u, v) and u' has a single outgoing edge (u', v') with the same label, say a, and v and v' each have in-degree exactly 1. By our assumption on how the identifiers are assigned, the identifiers of v and v' are $i + 1$ and $i' + 1$ and, by the definition of a Wheeler graph, v is immediately preceded by v' in π. It follows that if $i + \ell$ is the successor of i then it has stored with it as satellite data $i' + \ell$, and so we can compute ℓ and then i' in $O(\log \log n)$ time.

▶ **Lemma 6.** *We can store G in $O(r + v)$ space such that we can evaluate ϕ in $O(\log \log n)$ time.*

A.3 Discussion

Combining Lemmas 5 and 6, we generalize, we obtain Theorem 3. Since $v = 1$ for a single string labelling a simple path or cycle, Theorem 3 gives the same $O(r)$ space bound and $O(|P| + k \log \log n)$ time bound we achieve with the r-index for strings, where k is the number of occurrences. Nishimoto and Tabei [50] recently improved the query time of the r-index for strings to $O(P + k \log \log n)$ – or optimal $O(P + k)$ for polylogarithmic alphabets – without changing the space bound, and we conjecture this is achievable also for r-indexes for Wheeler graphs.

BPPart: RNA-RNA Interaction Partition Function in the Absence of Entropy

Ali Ebrahimpour-Boroojeny
Department of Computer Science, Columbia University, New York, NY, USA [1]
New York Genome Center, NY, USA

Sanjay Rajopadhye
Department of Computer Science, Colorado State University, Fort Collins, CO, USA

Hamidreza Chitsaz
Waymo, Mountain View, CA, USA[2]

Abstract

A few classes of RNA-RNA interaction (RRI) with complex roles in cellular functions, such as miRNA-target and lncRNAs, have already been studied. Accordingly, RRI bioinformatics tools proposed in the last decade are tailored for those specific classes. Interestingly, there are somewhat unnoticed mRNA-mRNA interactions in the literature with potentially drastic biological roles. Hence, there is a need for high-throughput *generic* RRI bioinformatics tools that can be used in more comprehensive settings. In this work, we revisit two of the RRI partition function algorithms, `piRNA` and `rip`. These are equivalent methods that implement the most comprehensive and computationally intensive thermodynamic model for RRI. We propose simpler models that are shown to retain the vast majority of the thermodynamic information that the more complex models capture. Specifically, we simplify the energy model by ignoring the system's entropy and show its equivalency to a base-pair counting model. We allow different weights for base-pairs to maximize the correlations with the full thermodynamic model. Our newly developed algorithm, `BPPart`, is 225× faster than `piRNA` and is more expressive and easier to analyze due to its simplicity and order of magnitude reduction in the number of dynamic programming tables. Still, based on our analysis of both the real and randomly generated data, its scores achieve a correlation of 0.855 with `piRNA` at $37^{\circ}C$. Finally, we illustrate one use-case of such simpler models to generate hypotheses about the roles of specific RNAs in various diseases. We have made our tool publicly available and believe that this faster and more expressive model will make the incorporation of physics-guided information in complex RRI analysis and prediction models more accessible.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases RNA-RNA Interaction, Partition Function, RNA Secondary Structure

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.14

Related Version *Full Version*: `https://arxiv.org/abs/1904.01235`

Supplementary Material *Software (Source Code)*: `https://github.com/Ali-E/bipart`
archived at `swh:1:dir:e8583c3964d5a1f404e893e461b70c2ece40c692`

1 Introduction

Since mid 1990s with the advent of RNA interference discovery, RNA-RNA interaction (RRI) has moved to the spotlight in modern, post-genome biology. RRI is ubiquitous and has increasingly complex roles in cellular functions. In human health studies, miRNA-target and lncRNAs are among an elite class of RRIs that have been extensively studied and shown to play significant roles in various diseases including cancer. Bacterial ncRNA-target and

[1] Work was done when the author was at Colorado State University.
[2] Work was done when the author was on the faculty of Computer Science at Colorado State University.

RNA interference are other classes of RRIs that have received significant attention. However, new evidence suggests that other classes of RRI, such as mRNA-mRNA interactions, are biologically important. The RISE database [16] reports a number of biologically significant instances of mRNA-mRNA interactions. These representative mRNA-mRNA interactions suggest that general RRIs, including mRNA-mRNA interactions, play major roles in human biology. Hence, there is a need for high-throughput *generic* RNA-RNA interaction bioinformatic tools for all types of RNAs.

In this paper, we revisit the well-studied problem of RNA-RNA interaction, and investigate the trade-off of complexity of the full thermodynamic models, such as `piRNA` [8] and `rip` [21], and accuracy of the scores they can generate. The aforementioned models are computationally intensive, and this prohibits their application to not only large-scale studies, but even for average sized pairs of RNAs. Because of the equivalency of these models, and availability of `piRNA` (the links to the tool provided by Huang et al. [21] are broken), we chose `piRNA` as the representative of the two in our experiments and analysis. `piRNA` is a dynamic programming algorithm that computes the partition function, base-pairing probabilities, and structure for the comprehensive Turner energy model in $O(n^4m^2 + n^2m^4)$ time and $O(n^4 + m^4)$ space. Due to intricacies of the energy model, including various (kissing) loops such as hairpin loop, bulge/internal loop, and multibranch loop, `piRNA` involves 96 different dynamic programming tables and needs multiple table look-ups for computing their values. An implementation of `piRNA` is currently available at `http://chitsazlab.org/software/pirna`.

In this paper, we introduce a strategic retreat from the slower comprehensive models such as `piRNA` by simplifying the energy model; we ignore the systems' entropy and derive a model that only requires the consideration of simple weighted base-pair counting. We develop the `BPPart` algorithm which aims to solve this simpler model with a much simpler approach. We also allow different weights for base-pairs which helps us to attain a model which correlates well with the full thermodynamic ones. In addition to this algorithm, we implemented a correct version of an earlier developed method, IRIS [39], which is based on base-pair maximization criterion, to have a thorough comparison between all these methods which are vastly different in terms of complexity. The implementation of this model, which we named `BPMax`, is also available in our publicly-available repository, and the results related to that are available in the Supplementary Material.

By the explosion of experimental data and the necessity to have higher-throughput methods, this retreat seems necessary, especially if one is willing to have more expressive models or wants to build physics-guided models that retain most of the information that can be derived from the thermodynamic system of RRI. `BPPart` involves eight 4-dimensional dynamic programming tables, and `BPMax` involves only one 4-dimensional table. Both `BPPart` and `BPMax` compared with `piRNA` are simpler dynamic programming algorithms which are more than 225× and 1300× faster, respectively, on the 50,500 RRI samples we used for our experiments. The reason for this noticeable speed-up is reducing the number of tables and the number of table look-ups for computing the new values and also the fact that the 96 large tables of `piRNA` renders `piRNA` memory- rather than compute-bound in practice. Moreover, the significantly reduced memory footprint of `BPPart` and `BPMax` makes them feasible targets for optimization on different hardware platforms like GPU based accelerators, an avenue we plan to explore in the future.

The key question concerns the accuracy we lose by simplifying the scoring model from the comprehensive Turner model to simply weighted base-pair counting. We answer this by computing both the Pearson and Spearman's rank correlations at different temperatures between the results of `BPPart`, `BPMax`, and `piRNA` on 50,500 experimentally characterized

RRIs in the RISE database [16]. We find that the Pearson correlations between `BPPart` and `piRNA` is 0.855 and `BPMax` and `piRNA` is 0.836 at $37°C$. Based on the results, we conclude that `BPPart` and `BPMax` capture a significant portion of the thermodynamic information. The simpler and faster algorithms, allow them to be used in high-throughput methods and be complemented with machine learning techniques in the future for more accurate predictions.

1.1 Related work

During the last few decades, several computational methods emerged to study the secondary structure of single and interacting nucleic acid strands. Most use a thermodynamic model such as the well-known Nearest Neighbor Thermodynamic model [32, 6, 13, 8, 38, 50, 54, 44, 33, 51]. Some previous attempts to analyze the thermodynamics of multiple interacting strands concatenate input sequences in some order and consider them as a single strand [2, 3, 12]. Alternatively, several methods avoid internal base-pairing in either strand and compute the minimum free energy secondary structure for their hybridization under this constraint [42, 11, 31]. The most comprehensive solution is computing the joint structure between two interacting strands under energy models with a growing complexity [40, 1, 29, 10, 23, 8, 21].

Other methods predict the secondary structure of individual RNA independently, and predict the (most likely) hybridization between the unpaired regions of the two interacting molecules as a multistep process: 1) unfolding of the two molecules to expose bases needed for hybridization, 2) the hybridization at the binding site, and 3) restructuring of the complex to a new minimum free energy conformation [35, 49, 5, 7]. The success of such methods, including our biRNA algorithm [7], suggests that the thermodynamic information vested in subsequences and pairs of subsequences of the input RNAs can provide valuable information for predicting features of the entire interaction.

In addition to general RNA-RNA interaction tools, many tools have been developed to predict the secondary structure of interacting RNAs for a specific type of interest which has been shown to be more effective in some cases due to the utilization of certain properties belonging to that type. As mentioned earlier, miRNA-target prediction is one such class of high interest for which such specialized tools have been created to incorporate various properties specific to miRNAs; some of these tools use the seed region of a miRNA which is highly conserved [26, 25, 27, 53], some consider the free energy to compute accessibility to the binding site in $3'$ UTR [18, 29, 25], some utilize the conservation level which is derived using the phylogenetic distance [36, 4, 41, 15, 26, 25], and some others consider other target sites as well, such as the $5'$ UTR, Open Reading Frames (ORF), and the coding sequence (CDS) for mRNAs [43, 34, 19, 52].

There are also several other tools developed for other specific types of RNA; IntaRNA [5, 30] is one such tool that although is used for RNA-RNA interaction in general, it is primarily designed for predicting target sites of non-coding RNAs (ncRNAs) on mRNAs. There are many other examples, such as PLEXY [24] which is a tool designed for C/D snoRNAs, RNAsnoop [45] that is designed for H/ACA snoRNAs, TargetRNA [46] which is a tool aimed at predicting interaction of bacterial sRNAs [48].

2 MATERIALS AND METHODS

Here we describe how our algorithm, `BPPart`, utilizes a dynamic programming approach to compute the partition function for RNA-RNA interaction when entropy is ignored and only a weighted score for pairing different nucleotides is considered. This algorithm is guaranteed to be mutually exclusive on the set of structures, i.e., it counts each structure exactly once.

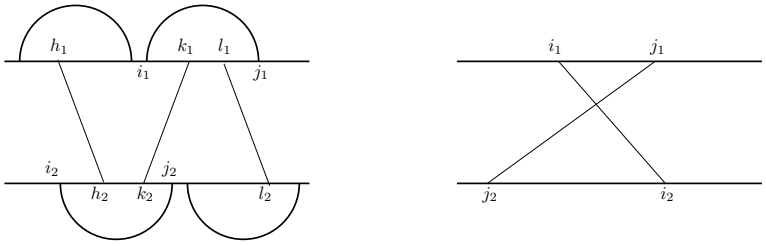


Figure 1 An illustration of a zigzag (left) and a crossing bond (right), which are excluded in our algorithm.

For `BPMax` which maximizes the (weighted scores) of base-pairs, such mutual exclusion is not necessary because the max operator is idempotent (counting the same structure multiple times does not affect the value of the objective function) and we can derive a simpler recursion. Our codes are freely available under open source license.

Preliminaries

In this paper, we mostly follow the notations and definitions used to develop our `piRNA` algorithm [8]. We denote the two nucleic acid strands by $\mathbf{R}$ and $\mathbf{S}$. Strand $\mathbf{R}$ is indexed from 1 to L_R, and $\mathbf{S}$ is indexed from 1 to L_S both in $5'$ to $3'$ direction. Note that the two strands interact in opposite directions, e.g. $\mathbf{R}$ in $5' \rightarrow 3'$ with $\mathbf{S}$ in $3' \leftarrow 5'$ direction; however, we consider the reverse of $\mathbf{S}$ in our figures for clearer illustration of the configurations. Each nucleotide is paired with at most one nucleotide in the same or the other strand. The subsequence from the i^{th} nucleotide to the j^{th} nucleotide, inclusive, in either strand is denoted by $[i, j]$.

An intramolecular base pair between the nucleotides i and j (by convention, $i < j$) in a strand is called an *arc* and denoted by a bullet $i \bullet j$. We represent the score of such arc by score(i, j). Essentially, score(i, j) is c_1 if $i \bullet j$ is GU or UG, is c_2 if $i \bullet j$ is AU or UA, and is c_3 if $i \bullet j$ is CG or GC. An intermolecular base pair between the nucleotides k_1 and k_2, where $k_1 \in R, k_2 \in S$, is called a *bond*, denoted by a circle $k_1 \circ k_2$. We represent the score of such a bond by iscore(k_1, k_2). Essentially, iscore(k_1, k_2) is c'_1 if $k_1 \circ k_2$ is GU or UG, is c'_2 if $k_1 \circ k_2$ is AU or UA, and is c'_3 if $k_1 \circ k_2$ is CG or GC.

An arc $i \bullet j$ in R *covers* a bond $k_1 \circ k_2$ if $i_1 < k_1 < j_1$. We call $i \bullet j$ an *interaction arc* in R if there is a bond $k_1 \circ k_2$ covered by $i \bullet j$. The *scope* of an interaction arc is the interval $[i+1, j-1]$. We call a base on either strand an *event* if it is either the end point of a bond or that of an interaction arc. In our explanation we may use arc and bond as verbs. Two bonds $i_1 \circ i_2$ and $j_1 \circ j_2$ are called *crossing bonds* (right case of Figure 1) if $i_1 < j_1$ and $i_2 > j_2$, or vice versa. An interaction arc $i_1 \bullet j_1$ in a strand *subsumes* a subsequence $[i_2, j_2]$ in the other strand if none of the bases in $[i_2, j_2]$ has a bond with a base outside this arc. Mathematically, for all bonds $k_1 \circ k_2$ where $i_2 < k_2 < j_2$, k_1 lies within the scope of $i_1 \bullet j_1$. Two interaction arcs are *equivalent* if they subsume one another. Two interaction arcs $i_1 \bullet j_1$ and $i_2 \bullet j_2$ are part of a *zigzag*, if neither $i_1 \bullet j_1$ subsumes $[i_2, j_2]$ nor $i_2 \bullet j_2$ subsumes $[i_1, j_1]$ (left case of Figure 1).

In this work, we assume there are no pseudoknots in individual secondary structures of $\mathbf{R}$ and $\mathbf{S}$, and also there are no crossing bonds and no zigzags between $\mathbf{R}$ and $\mathbf{S}$. These constraints allow a polynomial algorithm – the general case of considering all possible structures is NP-hard [1]. We denote the ensemble of unpseudoknotted structures of $\mathbf{R}$ and $\mathbf{S}$ by $\mathcal{S}(\mathbf{R})$ and $\mathcal{S}(\mathbf{S})$ respectively. The ensemble of unpseudoknotted, crossing-free, and zigzag-free joint interaction structures is denoted by $\mathcal{S}^I(\mathbf{R}, \mathbf{S})$.

For a given joint interaction structure $s \in \mathcal{S}^I(\mathbf{R}, \mathbf{S})$, let $\mathrm{AU}(s)$ denote the number of A-U base pairs in s. Similarly, $\mathrm{CG}(s)$ and $\mathrm{GU}(s)$ denote the number of C-G and G-U base pairs in s, respectively. We define *bpcount* as a weighted sum, for some constants, $c_1, \ldots, c_3$

$$\mathrm{bpcount}(s) = c_1\mathrm{GU}(s) + c_2\mathrm{AU}(s) + c_3\mathrm{CG}(s). \tag{1}$$

Rivas-Eddy Diagrams

For the sake of completeness, we describe the "Rivas-Eddy diagram" notation that we adopt in this paper in the Supplementary Material. The Rivas-Eddy diagram to compute a certain function is written like a formal (context free) grammar. The left hand side is labeled with the name of a table (structure), and the right hand side has a number of alternate substructures separated by vertical bars. Often, some of the boundary cases (e.g., singleton or empty subsequences) are omitted for brevity.

2.1 Problem Definition

The Gibbs free energy

$$\Delta G = \Delta H - T\Delta S \tag{2}$$

is composed of a term ΔH called enthalpy that does not depend on temperature and a term $T\Delta S$ that includes entropy and is linearly dependent on temperature T. Intuitively, enthalpy is the chemical energy that is often released upon formation of chemical bonds such as base pairing. Entropy, on the other hand, captures the size of all possible spatial conformations for a fixed secondary structure. In other words, entropy captures the amount of 3D freedom of the molecule. A base-pair brings enthalpy down, hence favorable from an enthalpy point of view, and decreases freedom (entropy), hence unfavorable from an entropy point of view. These two opposing objectives are combined linearly through the temperature coefficient.

In the full thermodynamic model, we consider both terms. In the base pair counting, we consider only a simplistic enthalpy term. Partition function for the full thermodynamic model is

$$\sum_{s \in \mathcal{S}^I} e^{-\Delta G/RT}, \tag{3}$$

in which R is the gas constant, and S^I are all possible states of the system, assuming that they form a countable set (which they do in our case by we considering all possible ways the two RNAs pair with one another). Now, by ignoring the term with the entropy, and considering the approximation $\Delta G \sim \Delta H$, we can simplify the model as follows

$$\sum_{s \in \mathcal{S}^I} e^{-\Delta G/RT} \approx \sum_{s \in \mathcal{S}^I} e^{-\Delta H/RT} \approx \sum_{s \in \mathrm{S}^I(\mathbf{R},\mathbf{S})} e^{-\mathrm{bpcount}(s)/RT}. \tag{4}$$

To make the 3rd term a better approximation for the first one, we allow different weights for different base-pairs (`AC`, `GT`, and `CG`) in our model. We optimize these weights to maximize the correlation of the scores with those of `piRNA` (which is based on the first term above)

and verify the consistency of the computed weights using a randomly generated dataset. So, basically, by allowing the base-pairs to have different weights and finding the optimum ones, we seek to minimize the information we lose by ignoring the term with the entropy.

In our experiments, we also perform analysis on the base-pair maximization model, `BPMax`, which finds the structure that has the maximum weighted base pair count, i.e.

$$\text{BPMax}(\mathbf{R},\mathbf{S}) = \max_{s\in \text{S}^I(\mathbf{R},\mathbf{S})} \text{bpcount}(s). \tag{5}$$

This problem (without the weights for base-pairs) was previously studied by Pervouchine [40] in an algorithm called `IRIS`. However, there is no publicly available correct implementation of `IRIS`. As in `BPPart`, we allow weighted scores for the base-pairs in the `BPMax` algorithm to maximize the correlation of its scores with those of `piRNA`. We give a dynamic programming algorithm for this model in the Supplementary Material.

2.2 BPPart Algorithm

In this section, we give a dynamic programming algorithm, `BPPart`, to compute the partition function. It is well-known that the partition function can be computed by developing similar recursions as the one introduced in the simpler base-pair maximization models, such as `BPMax` and `IRIS`, with two simple modifications. The first is that algebraically, we operate with the field of reals rather than the max-plus semi-ring. Here, the additive identity is 0, rather than `INT_MIN` and the multiplicative identity is 1, rather than 0. The second is that because addition is not idempotent, we must carefully ensure that we enumerate substructures in a mutually exclusive manner. Before starting to explain the algorithm and its recursions, we have to mention that similar and equivalent (except for the weighted base-pairs feature that is being to our model to decrease the effect of ignoring entropy) algorithms can be derived from the complete models (`piRNA` and `rip`). However, we found it easier to come with decompositions and recursions from scratch and build our 8 dynamic programming tables, rather than starting with the complete models with over 90 tables, and eliminating or merging those that capture cases not required in our simplified model. This also helps us to come up with less and cleaner equations, and avoid any potential problems in reducing those methods to solve our problem. Still the overall structure of the algorithms would probably seem similar due to their common approach toward computing the partition function, namely decomposing more general structure to simpler ones and using dynamic programming.

First, we start with the recursions for computing the partition function on a single strand which is going to occur in many cases of the double-stranded version. Let $\text{Q}_{i,j}$ represent the partition function of the subsequence $[i,j]$. As shown in Figure 2, there are two mutually exclusive cases: either (the right case) there is no arc, or (the left case) there is a unique leftmost arc (the cyan fill ensures this) which starts at k, and a substructure on $[k,j]$ with an arc starting at k, for which we introduce a new table Qz.

To define $\text{Qz}_{i,j}$, let $i \bullet k$ (read as "let i arc k") for some index k. This results in two Q substructures, one on $[i+1,k-1]$, and the other on $[k+1,j]$. Therefore, the value of $\text{Qz}_{i,j}$ can be computed using Equation (7) which accounts for the assumption that no pairing is allowed between two bases that are less than 3 bases apart:

$$\text{Q}_{i,j} = \begin{cases} 1 & j \le i \\ 1 + \sum_{k=i}^{j-4} \text{Qz}_{k,j} & \text{otherwise,} \end{cases} \tag{6}$$

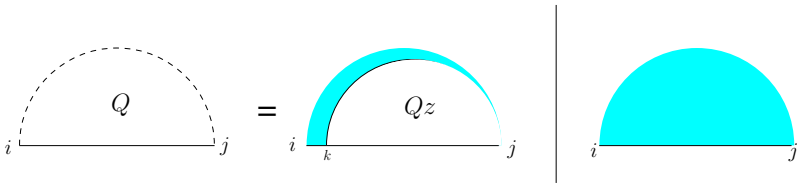


Figure 2 For computing Q, notice that either there is no pairing or there is at least one arc which starts at some index k and results in a case of Qz.

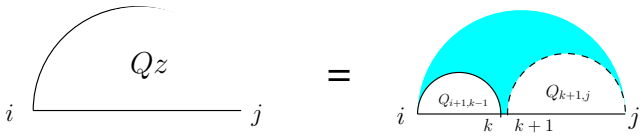


Figure 3 Computing Qz can be achieved by considering the base k that is paired with i and the two Q substructures it forms, one between i and k and one after k.

$$\mathrm{Qz}_{i,j} = \begin{cases} 0 & j - i < 4 \\ \sum_{k=i+4}^{j} \mathrm{Q}_{i+1,k-1} \times e^{\mathrm{score}(i,k)} \times \mathrm{Q}_{k+1,j} & \text{otherwise.} \end{cases} \tag{7}$$

For the partition function of a pair of RNA sequences, we consider a 4-dimensional table QI in which $\mathrm{QI}_{i_1,j_1,i_2,j_2}$ is the value of base pair counting partition function for the subsequences $[i_1, j_1]$ on **R** and $[i_2, j_2]$ on **S**. As Figure 4 shows, we can split the set of all possible structures of QI into three mutually exclusive subsets. The leftmost case shows the structures in which there exist no bonds (the first term of Equation (8). The other two cases occur when there is at least one bond, and hence, unique leftmost events on both **R** and **S**, at positions k_1 and k_2, respectively. In the second (middle) case, these leftmost events are end points of a bond, $k_1 \circ k_2$; hence, this case can be broken into: a bond-free section on the left of the bond itself, and a general case of QI on the right of the bond. The third case occurs when k_1 and k_2 are not end points of a bond. We call this structure QIa, and explain it next.

$$\begin{aligned} &\mathrm{QI}_{i_1,j_1,i_2,j_2} = \\ &\quad \mathrm{Q}^{(1)}_{i_1,j_1} \times \mathrm{Q}^{(2)}_{i_2,j_2} + \\ &\quad \sum_{k_1=i_1}^{j_1} \sum_{k_2=i_2}^{j_2} L_{i_1,j_1,k_1,i_2,j_2,k_2} + \\ &\quad \sum_{k_1=i_1}^{j_1} \sum_{k_2=i_2}^{j_2} \left(\mathrm{Q}^{(1)}_{i_1,k_1-1} \times \mathrm{Q}^{(2)}_{i_2,k_2-1} \times \mathrm{QIa}_{k_1,j_1,k_2,j_2} \right), \end{aligned} \tag{8}$$

$$L_{i_1,j_1,k_1,i_2,j_2,k_2} = \mathrm{Q}^{(1)}_{i_1,k_1-1} \times \mathrm{Q}^{(2)}_{i_2,k_2-1} \times e^{\mathrm{iscore}(k_1,k_2)} \times \mathrm{QI}_{k_1+1,j_1,k_2+1,j_2}. \tag{9}$$

For computing $\mathrm{QIa}_{i_1,j_1,i_2,j_2}$, (see Figure 5) we have to consider the property of this structure that the leftmost bases on both **R** and **S** have to be events, but they cannot both be the end points of a bond. Therefore, either one or both of them have to be end points of an interaction arc. There are two possibilities.

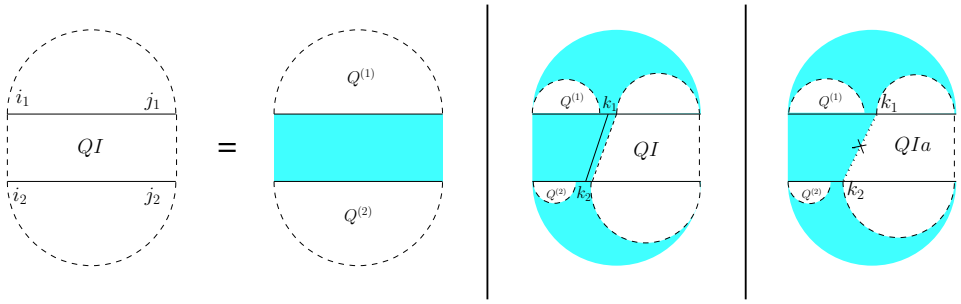


Figure 4 Each case of a QI structure (left side of the equation) can lead to three cases: either no bonds exist (leftmost case), or at least one bond exists. If the first event on both of the sequences is a bond (middle case) the subsequences to the left of the bond involve only Q and the subsequences to the right recurs on QI. Otherwise (rightmost case) we will have QIa (see Figure 5).

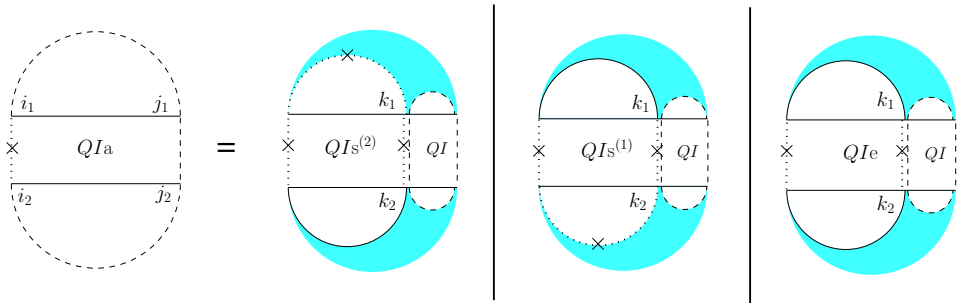


Figure 5 There are three cases for computing the QIa structure; either the leftmost base of only one of the strands is an end point of an arc or both end points are.

First, if both i_1 and i_2 are end points of some interaction arcs, $i_1 \bullet k_1$ and $i_2 \bullet k_2$, these arcs must be equivalent (or else, we have a zigzag). As shown in the rightmost diagram in Figure 5, QIa then splits into two exclusive substructures, namely one where the first and last bases on each strand are paired, and the two arcs are equivalent (we call it $\text{QIe}_{i_1,k_1,i_2,k_2}$ and derive its recursion later), followed by $\text{QI}_{k_1+1,j_1,k_2+1,j_2}$ on the suffixes of these arcs.

Otherwise, exactly one of the leftmost events on **R** and **S** is an end point of a bond, and we have two symmetric cases ($\text{QIs}^{(1)}$ and $\text{QIs}^{(2)}$), one where the interaction arc is in **R**, and the other where it is in **S**. In the first case (middle diagram in Figure 5), let k_1 be the event in **R** such that $i_1 \bullet k_1$ is an interaction arc, and $[i_2, k_2]$ is the longest subsequence in **S** that $i_1 \bullet k_1$ subsumes, and k_2 is an event. The suffix of this substructure recurs on QI. We derive $\text{QIs}^{(1)}$ later.

To derive $\text{QIe}_{i_1,j_1,i_2,j_2}$, note that removing the arcs $i_1 \bullet j_1$ and $i_2 \bullet j_2$ yields the general case of $\text{QI}_{i_1+1,j_1-1,i_2+1,j_2-1}$ for the inner-section with an additional constraint that there has be at least one bond in that region because the assumption is that the extracted arcs were interaction arcs. We can fulfill this constraint by excluding all cases where no bonds exist (i.e., considering only the two rightmost substructures of Figure 4).

To derive $QIs^{(1)}_{i_1,j_1,i_2,j_2}$ let k_1 be the leftmost event in the subsequence $[i_1 + 1, j_1 - 1]$. Note that such a k_1 is guaranteed to exist because first, $i_1 \bullet j_1$ subsumes $[i_2, j_2]$ and we know that i_2 is an event, i.e., the end point of either a bond (subsumed by $i_1 \bullet j_1$) or of an interaction arc. Then (see Figure 6) we define a new substructure, $\text{QIaux}^{(1)}$, after removing $i_1 \bullet j_1$ and the prefix of **R** up to k_1.

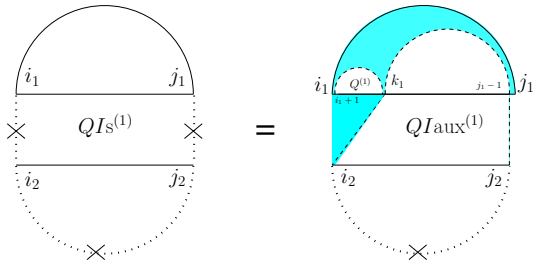


Figure 6 $QIs^{(1)}$ has one arc that can be extracted and the structure derived will have the property that the two end bases of the bottom strand cannot be paired (the new structure inherits this property from $QIs^{(1)}$). On the top strand, we consider the leftmost event. This new structure is $QIaux^{(1)}$.

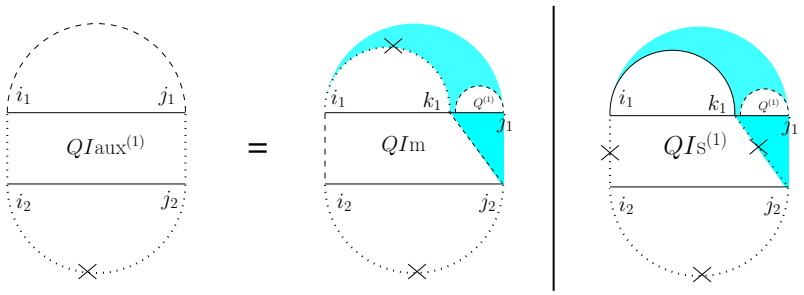


Figure 7 Two cases must be considered for the $QIaux^{(1)}$ structure, in which the two end points of the bottom strand are events. For the top strand, only the leftmost end point is required to be an event. It can either be the end point of an arc (rightmost case) or not (leftmost case).

To derive $QIaux^{(1)}_{i_1,j_1,i_2,j_2}$, note that the context of its definition implies that i_1, i_2 and j_2 are all three events. Let, as shown in Figure 7, k_1 be the *last* event on $[i_1, j_1]$. Now, if $i_1 \bullet k_1$, then recur on $QIs^{(1)}$. Otherwise, k_1 is an event that does not pair with i_1. We define a new substructure, QIm, where all four corners are events, and neither $i_1 \bullet j_1$ nor $i_2 \bullet j_2$ is allowed.

For computing QIm_{i_1,j_1,i_2,j_2}, since there are four corners each of which can be the end point of either a bond or of an arc, there might be at most sixteen possibilities. Upon combining some of those sixteen possibilities, we have to consider four mutually exclusive cases (see Figure 8). The first one is the case where $i_1 \circ i_2$ and $j_1 \circ j_2$ and the remaining part will be $QI_{i_1+1,j_1-1,i_2+1,j_2-1}$. That case corresponds to all four corner events being the end points of bonds. Since we assume there are no crossing bonds, we must have $i_1 \circ i_2$ and $j_1 \circ j_2$. In the second case, i_1 and i_2 are the end points of a bond, i.e., $i_1 \circ i_2$, but either j_1 or j_2 (or both) does not form a bond. That captures three out of the sixteen total possibilities. Since j_1 and j_2 are both events but do not form a bond, we define a term QIac which is the sum of QIe and the two symmetric QIs's, since they preserve the constraints that arise in the first term in the definition of QIa (see Figure 5). Note that we do not need a separate dynamic programming table for QIac because it can simply be replaced with the sum of the terms it represents. However, using this terms helps us to keep the equations easier to follow. The prefix of this substructure in the second case is a general recursion on QI on the subsequences $[i_1 + 1, k_1 - 1]$ and $[i_2 + 1, k_2 - 1]$. The third case is the symmetric case of the second case, i.e., there is no bond between i_1 and i_2, but $j_1 \circ j_2$. The prefix of this bond is a recursion on QIa. That captures three out of the sixteen total possibilities. Finally, the

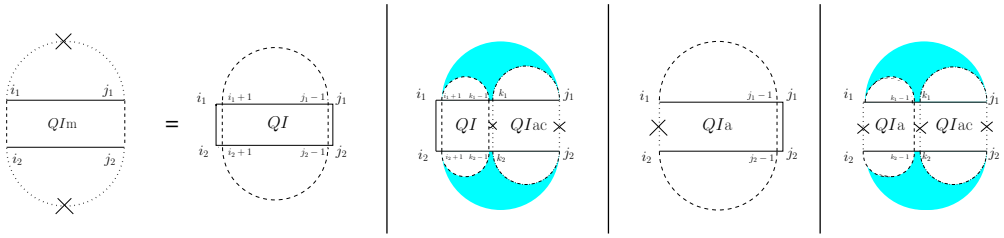


Figure 8 For computing QIm, since we know the four end points are events, but none of the two end points in one strand can form an arc, we must consider the four different cases shown above. For convenience, arcs of QIac structure are shown with dash-dotted lines because it represents the sum of three structures in which each of the arcs could be present or not (we could replace the second and fourth cases with three cases, one for each term of Equation (11)).

fourth case corresponds to either i_1 or i_2 (or both) does not form a bond and either j_1 or j_2 (or both) does not form a bond. That captures the remaining nine out of the sixteen total possibilities.

Putting all those together, we obtain

$$\mathrm{QIa}_{i_1,j_1,i_2,j_2} = \sum_{k_1=i_1}^{j_1} \sum_{k_2=i_2}^{j_2} \mathrm{QIac}_{i_1,k_1,i_2,k_2} \times \mathrm{QI}_{k_1+1,j_1,k_2+1,j_2}, \tag{10}$$

$$\mathrm{QIac}_{i_1,j_1,i_2,j_2} = \mathrm{QIs}^{(1)}_{i_1,j_1,i_2,j_2} + \mathrm{QIs}^{(2)}_{i_1,j_1,i_2,j_2} + \mathrm{QIe}_{i_1,j_1,i_2,j_2}, \tag{11}$$

$$\mathrm{QIe}_{i_1,j_1,i_2,j_2} = \begin{cases} 0 & j_1 - i_1 < 4 \\ & \text{or} \quad j_2 - i_2 < 4 \\ M_{i_1,j_1,i_2,j_2} & \text{otherwise,} \end{cases} \tag{12}$$

$$M_{i_1,j_1,i_2,j_2} = \left(\mathrm{QI}_{i_1+1,j_1-1,i_2+1,j_2-1} - \mathrm{Q}^{(1)}_{i_1+1,j_1-1} \times \mathrm{Q}^{(2)}_{i_2+1,j_2-1}\right) \times e^{\mathrm{score}(i_1,j_1)+\mathrm{score}(i_2,j_2)}, \tag{13}$$

$$\mathrm{QIs}^{(1)}_{i_1,j_1,i_2,j_2} = \begin{cases} 0 & j_1 - i_1 < 4 \quad \text{or} \quad j_2 < i_2 \\ \displaystyle\sum_{k_1=i_1+1}^{j_1-1} \mathrm{Q}^{(1)}_{i_1+1,k_1-1} \times e^{\mathrm{score}(i_1,j_1)} \times \mathrm{QIaux}^{(1)}_{k_1,j_1-1,i_2,j_2} & \text{otherwise,} \end{cases} \tag{14}$$

$$\mathrm{QIs}^{(2)}_{i_1,j_1,i_2,j_2} = \begin{cases} 0 & j_1 < i_1 \quad \text{or} \quad j_2 - i_2 < 4 \\ \displaystyle\sum_{k_2=i_2+1}^{j_2-1} \mathrm{Q}^{(2)}_{i_2+1,k_2-1} \times e^{\mathrm{score}(i_2,j_2)} \times \mathrm{QIaux}^{(2)}_{i_1,j_1,k_2,j_2-1} & \text{otherwise,} \end{cases} \tag{15}$$

$$\mathrm{QIaux}^{(1)}_{i_1,j_1,i_2,j_2} = \sum_{k_1=i_1}^{j_1} \left(\mathrm{QIs}^{(1)}_{i_1,k_1,i_2,j_2} + \mathrm{QIm}_{i_1,k_1,i_2,j_2}\right) \times \mathrm{Q}^{(1)}_{k_1+1,j_1}, \tag{16}$$

$$\mathrm{QIaux}^{(2)}_{i_1,j_1,i_2,j_2} = \sum_{k_2=i_2}^{j_2} \left(\mathrm{QIs}^{(2)}_{i_1,j_1,i_2,k_2} + \mathrm{QIm}_{i_1,j_1,i_2,k_2}\right) \times \mathrm{Q}^{(2)}_{k_2+1,j_2}, \tag{17}$$

$$\mathrm{QIm}_{i_1,j_1,i_2,j_2} = \begin{cases} e^{\mathrm{iscore}(i_1,i_2)} & i_1 = j_1 \quad \text{and} \quad i_2 = j_2 \\ N_{i_1,j_1,i_2,j_2} & i_1 < j_1 \quad \text{and} \quad i_2 < j_2 \\ 0 & \text{otherwise,} \end{cases} \tag{18}$$

$$\begin{aligned} N_{i_1,j_1,i_2,j_2} = \\ & e^{\mathrm{iscore}(i_1,i_2)+\mathrm{iscore}(j_1,j_2)} \times \mathrm{QI}_{i_1+1,j_1-1,i_2+1,j_2-1} + \\ & e^{\mathrm{iscore}(i_1,i_2)} \times \sum_{k_1=i_1+1}^{j_1} \sum_{k_2=i_2+1}^{j_2} \mathrm{QI}_{i_1+1,k_1-1,i_2+1,k_2-1} \times \mathrm{QIac}_{k_1,j_1,k_2,j_2} + \\ & e^{\mathrm{iscore}(j_1,j_2)} \times \mathrm{QIa}_{i_1,j_1-1,i_2,j_2-1} + \\ & \sum_{k_1=i_1}^{j_1} \sum_{k_2=i_2}^{j_2} \mathrm{QIa}_{i_1,k_1,i_2,k_2} \times \mathrm{QIac}_{k_1+1,j_1,k_2+1,j_2}. \end{aligned} \tag{19}$$

3 Results

To investigate the correlation between the scores of `BPPart` and `BPMax`, and those of `piRNA`, we used the RISE database [16] which combines information about interacting RNAs from multiple experiments. For the human dataset, we extracted all the interaction windows for those pairs that have this data in RISE. We eliminated the ones with an interaction window size of less than 15 because they are too short for an unbiased comparison. Then, we sorted the remaining pairs based on the product of the lengths of the interacting windows (which is the base of the term that appears in the time-complexity of the algorithms). Finally, the first 50,500 pairs of sequences were chosen as our primary dataset for different experiments and analysis.

We ran `piRNA` on our primary dataset at $37°C$. In order to run `BPPart` on this dataset, we first have to choose the range of values that we want to explore for the weights of each base-pair. In general, we want to use the *stack* energies of the Turner model as a starting point for computing this range. Since the parameters form a projective space (invariant results with respect to scaling), we considered a fixed weight of 3 for `CG` (and `GC`). Using the experimentally computed stack energies of the Turner model, minimum and maximum values for the weights of `AU` and `GU` were computed. That is, to compute the maximum weight of `AU` (and `UA`), we consider the maximum released energy when `AU` (or `UA`) is stacked with another pair; this happens when `UA` is stacked with `CG` and 2.4 $kcal/mol$ energy is released. Then, we considered the minimum value of released energy in an stack for `CG` or `GC` (for which we assumed a constant weight of 3), which is 1.4 $kcal/mol$. We derived the maximum weight of `AU` and `UA` as 5.143 by multiplying 2.4 by $\frac{3}{1.4}$. Finally, we made sure that the range of values that we explore for the weight of `AU` and `UA` contains this maximum value (we chose 5.5 as the upper-bound). For finding the minimum weight of `AU` and `UA`, we consider their minimum stack energy, which is 0.6 $kcal/mol$. Given the maximum energy of `CG`, namely

3.4 $kcal/mol$, the value of interest is computed as $0.6 \times \frac{3}{3.4} = 0.529$. However, for the sake of comprehensiveness and exploring the shape of the plots, we used much smaller lower-bound, -4.5, for our explorations.

Assuming a fixed weight of 3 for `CG`, we computed the Pearson and Spearman's Rank correlations with the scores from `piRNA`, for all the combinations of weights of `AU` and `GU` in steps of 0.5. When computing the correlations, to normalize the scores from all algorithms, we divide them by the sum of the lengths of corresponding sequences, $L_R + L_S$. This normalization mitigates the effect of length bias on the computed correlations. This step is necessary because, generally, as the length of the pair of sequences increases the scores of all three algorithms increases, and if unnormalized scores are used, a biased higher correlation will be derived. Note that for partition functions, `piRNA` and `BPPart`, we are computing the *log* of the scores; that is why we factor out the sum of the lengths for normalization. If the original values were to be used, we would have to take the $(L_R + L_S)$th roots of the scores. Figure 9 (a) shows the Pearson correlations for different combinations of weights of `AU` and `GU` at $37°C$. Figure 9 (b) shows the scatter plot of the scores for the best combination of weights, which are 0.5, 1.0, and 3 for `AU`, `GU`, and `CG`, respectively. In this plot, the red line shows the regression line that is fitted to the points by minimizing the mean squared error (MSE). We performed the same steps and analysis for `BPMax` method (more details on this method can be found in the Supplementary Material). The optimum values of correlation are presented in Table 1. As the results show, there is a high correlation between `piRNA` and `BPPart` as well as between `piRNA` and `BPMax`.

Table 1 Correlations between `piRNA` and `BPPart` and between `piRNA` and `BPMax` at $37°C$.

Method	Pearson	Spearman's Rank
BPPart	0.855	0.864
BPMax	0.836	0.830

To make sure that the base-pair weights derived by our optimization approach are not data-dependent, in spite of the our observation of very similar optimization plots on smaller portions of the primary data, we conducted the same experiments for randomly generated sequences. To factor out the effect of length, for each pair in our primary dataset, we generated a pair of random sequences with the same lengths as those of the pair in our primary dataset. Our results show similar optimized weights, but lower correlations on this dataset (this will be discussed in the next section). More details on the results for this dataset are provided in the Supplementary Material.

To better understand the behavior of the surface around the higher values in the correlation plot of Figure 9 (a) and Figure 15 (b) in the Supplementary Material, we computed the Shannon entropy for the values above a threshold. Figure 10 shows these values for the top 30 values of Pearson and Spearman's Rank correlation at each temperature. We discuss these results in the the next section.

Finally, we designed a pipeline for generating hypothesis about the roles of RNAs in different diseases using our newly developed algorithm which makes large-scale analysis of RRI datasets practical in a reasonable time (3 hours vs. one month using `piRNA`) with reasonable resources (6.6 GB of RAM vs. about 70 GB of RAM for `piRNA`). We elaborate on this pipeline and our results in the Supplementary Material.

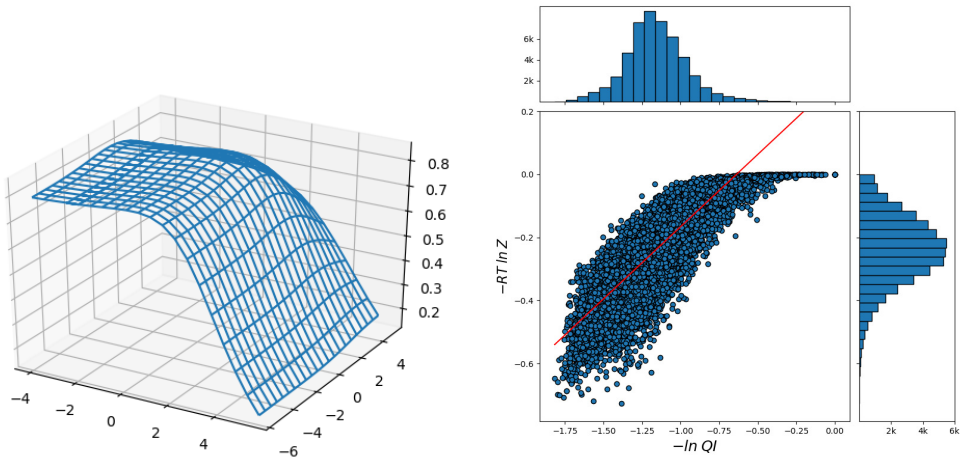


Figure 9 (a) Pearson correlation between `piRNA` and `BPPart` (vertical axis), on the primary dataset, at $37°C$ for different weights of AU (left axis) and GU (right axis). The weight of CG pair is fixed to 3. (b) Scatter plot of the scores form `piRNA` (y-axis) and `BPPart` (x-axis) at $37°C$. The read line is fitted to the points to minimize the Mean Squared Error (MSE).

4 Discussion

Note that we can rewrite equation 3 as the following

$$-\frac{\Delta G}{T} = -\frac{\Delta H}{T} + \Delta S, \tag{20}$$

and it is clear that as $T \to 0$, $-\Delta H/T \to \infty$ and the contribution of ΔS is diminished to 0 since it is finite. Hence, at low temperatures, the effect of entropy becomes negligible, and we expect to see strong correlation between the base pair counting model and full thermodynamic model. To verify that the scores computed with our models follow this theoretical observation, we computed the correlations at different temperatures, ranging from -180 ($°C$) to 37 ($°C$) (at temperatures lower than -180 ($°C$) the implementation of `piRNA` was unstable and resulted in NaN values, which prevented us from computing the correlation values). Figure 11 shows the Pearson correlations between `BPPart` and `BPMax` scores with `piRNA` scores for for their best combination of base-pair weights at 37 ($°C$). These optimum weights for `BPPart` are 0.5, 1.0, and 3 for `AU`, `GU`, and `CG`, respectively, and for `BPMax` are 1.0, 1.5, and 3 for `AU`, `GU`, and `CG`, respectively.

Perfectly conforming with the theory, we see higher correlations at low temperatures. These results, also, somewhat validates our implementations as `piRNA` was written totally independently more than 10 years ago. Moreover, by comparing Figure 9 (a) to Figure 12, and Figure 15 (b) to Figure 15 (a), we notice that the surface around the optimum value for higher temperatures becomes flatter. Figure 10, which shows the entropy of the top 30 correlation values, confirms this observation; this means the correlation values are less sensitive to a change in the weights of the base pairs as the temperature increases; this conforms with the theory because at higher temperatures, the thermodynamic entropy increases and the total score of `piRNA` becomes less sensitive to the energy released by pairings. This means that slight changes to our optimum weights at the body temperature, are less susceptible to result in different correlations than the optimum possible correlations that can be achieved by using the optimum weights.

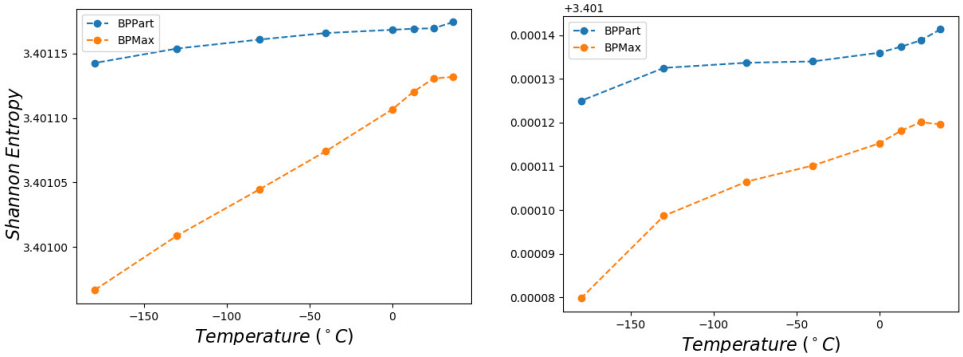


Figure 10 Shannon entropy for the top 30 Pearson (left) and Spearman's rank (right) correlation values at different temperatures for `BPPart` and `BPMax`.

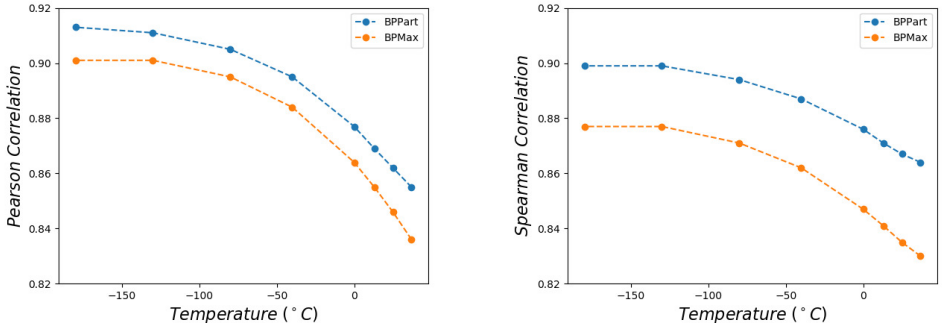


Figure 11 Pearson correlation (left) and Spearman's rank correlation (right) between `piRNA` and `BPPart` and between `piRNA` and `BPMax` at different temperatures.

Another noticeable characteristic of the optimization plots in Figures 9 (a) and 15 is the region in which the scores of both `AU` and `GU` are non-positive. This region for `BPMax` is flat because when both of these pairs are penalized (or not rewarded when their score is zero), the algorithm simply avoids making such pairs because it is trying to maximize the score. Therefore, it only tries to maximize the number of `CG` pairs, which is independent of the scores (penalty in this case) of the other two types of base pairs. This also applies to the case where one of the base pairs has a non-positive score; in that case, `BPMax` works independently of the score of that base pair. So, as soon as any of the scores becomes non-positive, `BPMax` remains constant along the corresponding axis. For `BPPart`, however, the story is different because it simply counts all the possible pairings and even if the score of a base pair becomes negative, it does not ignore counting that.

Moreover, `BPPart` has a higher correlation than `BPMax` does, which comes with the price of a 6× increase in empirical running time. Also, as Figure 10 shows, the Shannon entropy for the top 30 values is less in `BPMax` and the gap between them grows as temperature decreases; this shows that `BPPart` has a flatter region around the optimum value and its optimum correlation is less sensitive to changes in the optimum weights. Hence, we now have three choices in increasing order of computational cost: `BPMax`, `BPPart`, and `piRNA`. The

computation time increases by about 6× and 225×, respectively, from one to the next on the primary dataset. We also compared their costs on a single pair of sequences, each with a length of 100 bases. It took about 1, 6, and 1200 minuets and about 0.2 GB, 1.8 GB, and 18.5 GB of RAM for BPMax, BPPart, and piRNA, respectively, to compute the score of interaction. Note that here BPPart was about 200× faster than piRNA because the sequences had equal lengths, and the terms of degree four in the length of one of the sequences that appear in the time-complexity of piRNA (mentioned in the first section) do not make a difference here.

Given the higher correlations and less sensitivity to the optimum base-pair weights, paying the extra cost (compared to BPMax) to use BPPart seems justifiable in many applications. Another important benefit of partition functions, such as BPPart and piRNA over base-pair maximization models (e.g., BPMax) is that they can be used to compute probabilities that a base is paired or remains unpaired since we have the total counts for both cases; this property becomes necessary when working with tools such as rip [21] and biRNA [7]. Moreover, when studying the effects of SNPs and variants (e.g., the pipeline we have included in the Supplementary Material) on RNA-RNA interaction, BPMax cannot replace partition functions that are more sensitive to small perturbations.

Finally, based on the results of the experiments on both the primary dataset and the random one, we see that although the shapes of the optimization plots and the optimum weights are very similar, the correlation values are less for the random dataset. This observation is probably due to the fact that interaction regions are more complementary than the random sequences of the same size. When the genomic sequences are more complementary, the effect of the energy released by pairing becomes more significant than the energy added by an increase in the entropy on the final score of piRNA. In randomly generated sequences, however, BPPart and BPMax do not capture the increase in the entropy that leads to higher energy, which makes the interaction less desirable. With this effect, BPPart and BPMax, might overestimate the score of interaction among two non-interacting regions. It is worth mentioning that using the weighted base-pairs has diminished this effect because they are optimized to generate more similar scores to the ones from complete models that consider entropy. This hypothesis has to be thoroughly investigated in the future.

5 Conclusion

We revisited the problems of partition function and structure prediction for interacting RNAs. We simplified the energy model by ignoring the effects of entropy and reduced the full-thermodynamic model into a simple weighted base-pair counting one to obtain BPPart for the partition function. As a result, BPPart runs about 225× faster than piRNA does. Hence, we gained significant speedup by potentially sacrificing accuracy. To evaluate practical accuracy of our new model, we computed the Pearson and Spearman's Rank correlations between the results of BPPart and piRNA on 50,500 experimentally characterized RRIs in the RISE database [16]. Results highly correlate with those of piRNA. At the room and body temperatures, there is considerable correlation and therefore, significant information in the results of BPPart.

We conclude that our simpler models captures a significant portion of the thermodynamic information. Its considerable speedup and simplicity enables its use-cases in larger-scale studies which were not feasible with comprehensive models in reasonable time and resources. This approach for simplifying the full thermodynamic models can also be used together with other approximation methods that are based on thermodynamic models. Also, the information captured by BPPart can possibly be used to introduce physics-guided information

that may complement more complex prediction models in the future. We introduced a pipeline which becomes practical with our faster model and might be useful to explain how some mutations lead to some specific phenotypic consequences.

References

1 Can Alkan, Emre Karakoc, Joseph H Nadeau, S Cenk Sahinalp, and Kaizhong Zhang. Rna–rna interaction prediction and antisense rna target search. *Journal of Computational Biology*, 13(2):267–282, 2006.

2 Mirela Andronescu, Zhi Chuan Zhang, and Anne Condon. Secondary structure prediction of interacting rna molecules. *Journal of molecular biology*, 345(5):987–1001, 2005.

3 Stephan H Bernhart, Hakim Tafer, Ulrike Mückstein, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Partition function and base pairing probabilities of rna heterodimers. *Algorithms for Molecular Biology*, 1(1):1–10, 2006.

4 Doron Betel, Anjali Koppal, Phaedra Agius, Chris Sander, and Christina Leslie. Comprehensive modeling of microrna targets predicts functional non-conserved and non-canonical sites. *Genome biology*, 11(8):R90, 2010.

5 Anke Busch, Andreas S Richter, and Rolf Backofen. Intarna: efficient prediction of bacterial srna targets incorporating target site accessibility and seed regions. *Bioinformatics*, 24(24):2849–2856, 2008.

6 Song Cao and Shi-Jie Chen. Predicting rna pseudoknot folding thermodynamics. *Nucleic acids research*, 34(9):2634–2652, 2006.

7 Hamidreza Chitsaz, Rolf Backofen, and S Cenk Sahinalp. birna: Fast rna-rna binding sites prediction. In *International Workshop on Algorithms in Bioinformatics*, pages 25–36. Springer, 2009.

8 Hamidreza Chitsaz, Raheleh Salari, S.Cenk Sahinalp, and Rolf Backofen. A partition function algorithm for interacting nucleic acid strands. *Bioinformatics*, 25(12):i365–i373, 2009. Also ISMB/ECCB proceedings.

9 Ilaria Di Donato, Silvia Bianchi, Nicola De Stefano, Martin Dichgans, Maria Teresa Dotti, Marco Duering, Eric Jouvent, Amos D Korczyn, Saskia AJ Lesnik-Oberstein, Alessandro Malandrini, et al. Cerebral Autosomal Dominant Arteriopathy with Subcortical Infarcts and Leukoencephalopathy (CADASIL) as a model of small vessel disease: update on clinical, diagnostic, and management aspects. *BMC medicine*, 15(1):41, 2017.

10 Laura DiChiacchio, Michael F Sloma, and David H Mathews. Accessfold: predicting rna–rna interactions with consideration for competing self-structure. *Bioinformatics*, 32(7):1033–1039, 2015.

11 Roumen A Dimitrov and Michael Zuker. Prediction of hybridization and melting for double-stranded nucleic acids. *Biophysical Journal*, 87(1):215–226, 2004.

12 Robert M Dirks, Justin S Bois, Joseph M Schaeffer, Erik Winfree, and Niles A Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM review*, 49(1):65–88, 2007.

13 Robert M Dirks and Niles A Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *Journal of computational chemistry*, 24(13):1664–1677, 2003.

14 Ali Ebrahimpour-Boroojeny, Sanjay Rajopadhye, and Hamidreza Chitsaz. Bppart and bpmax: Rna-rna interaction partition function and structure prediction for the base pair counting model. *arXiv preprint*, 2019. `arXiv:1904.01235`.

15 Dimos Gaidatzis, Erik van Nimwegen, Jean Hausser, and Mihaela Zavolan. Inference of mirna targets using evolutionary conservation and pathway analysis. *BMC bioinformatics*, 8(1):69, 2007.

16 Jing Gong, Di Shao, Kui Xu, Zhipeng Lu, Zhi John Lu, Yucheng T Yang, and Qiangfeng Cliff Zhang. Rise: a database of rna interactome from sequencing experiments. *Nucleic acids research*, 46(D1):D194–D201, 2018.

17 Mitchell Guttman, Ido Amit, Manuel Garber, Courtney French, Michael F Lin, David Feldser, Maite Huarte, Or Zuk, Bryce W Carey, John P Cassady, et al. Chromatin signature reveals over a thousand highly conserved large non-coding RNAs in mammals. *Nature*, 458(7235):223–227, 2009.

18 Ivo L Hofacker, Walter Fontana, Peter F Stadler, L Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994.

19 Justin Bo-Kai Hsu, Chih-Min Chiu, Sheng-Da Hsu, Wei-Yun Huang, Chia-Hung Chien, Tzong-Yi Lee, and Hsien-Da Huang. mirtar: an integrated system for identifying mirna-target interactions in human. *BMC bioinformatics*, 12(1):300, 2011.

20 Hong Ming Hu, Karen O'Rourke, Mark S Boguski, and Vishua M Dixit. A novel RING finger protein interacts with the cytoplasmic domain of CD40. *Journal of Biological Chemistry*, 269(48):30069–30072, 1994.

21 Fenix WD Huang, Jing Qin, Christian M Reidys, and Peter F Stadler. Partition function and base pairing probabilities for rna–rna interaction prediction. *Bioinformatics*, 25(20):2646–2654, 2009.

22 Anne Joutel, Christophe Corpechot, Anne Ducros, Katayoun Vahedi, Hugues Chabriat, Philippe Mouton, Sonia Alamowitch, Valérie Domenga, Michaelle Cecillion, Emmanuelle Marechal, et al. Notch3 mutations in cerebral autosomal dominant arteriopathy with subcortical infarcts and leukoencephalopathy (CADASIL), a mendelian condition causing stroke and vascular dementia. *Annals of the New York Academy of Sciences*, 826(1):213–217, 1997.

23 Yuki Kato, Tatsuya Akutsu, and Hiroyuki Seki. A grammatical approach to rna–rna interaction prediction. *Pattern Recognition*, 42(4):531–538, 2009.

24 Stephanie Kehr, Sebastian Bartschat, Peter F Stadler, and Hakim Tafer. Plexy: efficient target prediction for box c/d snornas. *Bioinformatics*, 27(2):279–280, 2010.

25 Michael Kertesz, Nicola Iovino, Ulrich Unnerstall, Ulrike Gaul, and Eran Segal. The role of site accessibility in microrna target recognition. *Nature genetics*, 39(10):1278, 2007.

26 Azra Krek, Dominic Grün, Matthew N Poy, Rachel Wolf, Lauren Rosenberg, Eric J Epstein, Philip MacMenamin, Isabelle Da Piedade, Kristin C Gunsalus, Markus Stoffel, et al. Combinatorial microrna target predictions. *Nature genetics*, 37(5):495, 2005.

27 Jan Krüger and Marc Rehmsmeier. Rnahybrid: microrna target prediction easy, fast and flexible. *Nucleic acids research*, 34(suppl_2):W451–W454, 2006.

28 Almin I Lalani, Carissa R Moore, Chang Luo, Benjamin Z Kreider, Yan Liu, Herbert C Morse, and Ping Xie. Myeloid cell TRAF3 regulates immune responses and inhibits inflammation and tumor development in mice. *The Journal of Immunology*, 194(1):334–348, 2015.

29 Ronny Lorenz, Stephan H Bernhart, Christian Hoener Zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011.

30 Martin Mann, Patrick R Wright, and Rolf Backofen. Intarna 2.0: enhanced and customizable prediction of rna–rna interactions. *Nucleic acids research*, 45(W1):W435–W439, 2017.

31 NR Markham, M Zuker, and JM Keith. Unafold: software for nucleic acid folding and hybridization., pp. 3–31, 2008.

32 David H Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of rna secondary structure. *Journal of molecular biology*, 288(5):911–940, 1999.

33 John S McCaskill. The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers: Original Research on Biomolecules*, 29(6-7):1105–1119, 1990.

34 Kevin C Miranda, Tien Huynh, Yvonne Tay, Yen-Sin Ang, Wai-Leong Tam, Andrew M Thomson, Bing Lim, and Isidore Rigoutsos. A pattern-based method for the identification of microrna binding sites and their corresponding heteroduplexes. *Cell*, 126(6):1203–1217, 2006.

35 Ulrike Mückstein, Hakim Tafer, Jörg Hackermüller, Stephan H Bernhart, Peter F Stadler, and Ivo L Hofacker. Thermodynamics of rna–rna binding. *Bioinformatics*, 22(10):1177–1182, 2006.

36 Jin-Wu Nam, Olivia S Rissland, David Koppstein, Cei Abreu-Goodger, Calvin H Jan, Vikram Agarwal, Muhammed A Yildirim, Antony Rodriguez, and David P Bartel. Global analyses of the effect of different cellular contexts on microrna targeting. *Molecular cell*, 53(6):1031–1043, 2014.

37 Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

38 Ruth Nussinov, George Pieczenik, Jerrold R Griggs, and Daniel J Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied mathematics*, 35(1):68–82, 1978.

39 Dmitri D Pervouchine. Iris: intermolecular rna interaction search. *Genome Informatics*, 15(2):92–101, 2004.

40 Dmitri D Pervouchine. Iris: intermolecular rna interaction search. *Genome Informatics*, 15(2):92–101, 2004.

41 Martin Reczko, Manolis Maragkakis, Panagiotis Alexiou, Ivo Grosse, and Artemis G Hatzigeorgiou. Functional microrna targets in protein coding sequences. *Bioinformatics*, 28(6):771–776, 2012.

42 Marc Rehmsmeier, Peter Steffen, Matthias Höchsmann, and Robert Giegerich. Fast and effective prediction of microrna/target duplexes. *Rna*, 10(10):1507–1517, 2004.

43 Ángela Riffo-Campos, Ismael Riquelme, and Priscilla Brebi-Mieville. Tools for sequence-based mirna target prediction: what to choose? *International journal of molecular sciences*, 17(12):1987, 2016.

44 Elena Rivas and Sean R Eddy. A dynamic programming algorithm for rna structure prediction including pseudoknots. *Journal of molecular biology*, 285(5):2053–2068, 1999.

45 Hakim Tafer, Stephanie Kehr, Jana Hertel, Ivo L Hofacker, and Peter F Stadler. Rnasnoop: efficient target prediction for h/aca snornas. *Bioinformatics*, 26(5):610–616, 2010.

46 Brian Tjaden. Targetrna: a tool for predicting targets of small rna action in bacteria. *Nucleic acids research*, 36(suppl_2):W109–W113, 2008.

47 Shoji Tsuji, Prabhakara V Choudary, Brian M Martin, Suzanne Winfield, John A Barranger, and Edward I Ginns. Nucleotide sequence of cdna containing the complete coding sequence for human lysosomal glucocerebrosidase. *Journal of Biological Chemistry*, 261(1):50–53, 1986.

48 Sinan Uğur Umu and Paul P Gardner. A comprehensive benchmark of rna–rna interaction prediction tools for all domains of life. *Bioinformatics*, 33(7):988–996, 2017.

49 S Patrick Walton, Gregory N Stephanopoulos, Martin L Yarmush, and Charles M Roth. Thermodynamic and kinetic characterization of antisense oligodeoxynucleotide binding to a structured mrna. *Biophysical journal*, 82(1):366–377, 2002.

50 Michael S Waterman and Temple F Smith. Rna secondary structure: A complete mathematical analysis. *Mathematical Biosciences*, 42(3-4):257–266, 1978.

51 Anne Wenzel, Erdinç Akbaşli, and Jan Gorodkin. Risearch: fast rna–rna interaction search using a simplified nearest-neighbor energy model. *Bioinformatics*, 28(21):2738–2746, 2012.

52 Wenlong Xu, Anthony San Lucas, Zixing Wang, and Yin Liu. Identifying microrna targets in different gene regions. *BMC bioinformatics*, 15(7):S4, 2014.

53 Yuanji Zhang. miru: an automated plant mirna target prediction server. *Nucleic acids research*, 33(suppl_2):W701–W704, 2005.

54 Michael Zuker and Patrick Stiegler. Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981.

A Rivas-Eddy Diagrams

Here we describe the "Rivas-Eddy diagram" notation that we adopt in this paper. The main elements are:

1. A solid horizontal straight line represents a sequence; we have two sequences drawn as two parallel horizontal lines.
2. A solid curved line between two points in the same sequence is an arc; all arcs are either above the upper sequence, or below the lower one.
3. A dotted curved line with a cross in the middle, between two points in the same sequence means that those two points *do not* form an arc.
4. A dashed curved line between two points in the same sequence denotes either 2 or 3.
5. A solid line between two points in different sequences is a bond.
6. Similarly, a dotted line with a cross in the middle, between two points in different sequences means that those two points *do not* form a bond.
7. A dashed line between two points in different sequences denotes either 5 or 6.
8. A region is the space under an arc, or between bonds. When there are no additional choices of bonds/arcs in a given region, we fill it with a color (cyan); no arc or bond crosses a filled region.
9. A point in a sequence may be labeled with an index, and in general, the set of such indices are free variables used in the recursions; the index of unlabeled points before (after) labeled points is assumed to be the predecessor (successor) of the label.
10. A diagram may be labeled with the name(s) of the constituent substructures (which are eventually implemented as dynamic programming tables/variables).
11. A vanishing arc (i.e., one that starts at some index, and does not explicitly specify an end point) represents a structure whose start point is as specified, and the end point is to be determined.

B Other Results for BPPart

For the sake of comparison of how the plots in Figure 9 would look like at $-180°C$, we generated the same plots and presented them in Figure 12.

As mentioned in the paper, we performed the same optimization procedure on randomly-generated data. Figure 13 shows these optimization plots. Notice that we shapes of the plots and optimum weights are very similar, but the correlations are less. The potential reasons for this observation are disucussed in the paper.

C BPMax Algorithm

Here, we give the dynamic programming algorithm for the `BPMax` model. When explaining some of the equations, helper functions, called H, L, M, N, are used to ease the reading of the paper. To differentiate these helper functions, superscripts are used.

For a single strand of nucleotides, we define $\mathrm{S}_{i,j}$ as the maximum weighted sum of base pair scores on all possible foldings of subsequence $[i, j]$. We need to make such a table, for each of the **R** and **S** strands, and we distinguish between them by superscripts (1) and (2), respectively. We also define F_{i_1,j_1,i_2,j_2} as the maximum weighted sum of base pair scores (both intra- and inter-pairings) of subsequences $[i_1, j_1]$ from **R** and $[i_2, j_2]$ from **S**.

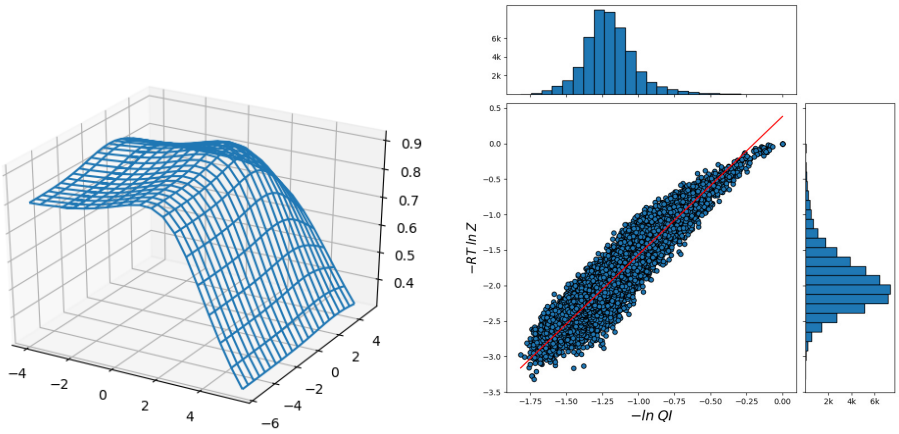


Figure 12 (a) Pearson correlation between `piRNA` and `BPPart` (vertical axis), on the primary dataset, at $-180°C$ for different weights of AU (left axis) and GU (right axis). The weight of CG pair is fixed to 3. (b) Scatter plot of the scores form `piRNA` (y-axis) and `BPPart` (x-axis) at $-180°C$. The read line is fitted to the points to minimize the Mean Squared Error (MSE).

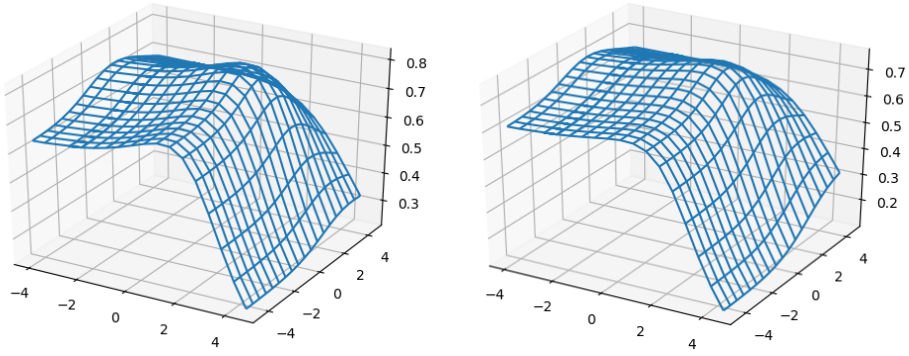


Figure 13 Pearson correlation between `piRNA` and `BPPart` (vertical axis), on the randomly generated dataset, at $-180°C$ (left) and $37°C$ (right) for different values of constant factors (weights) for AU (left axis) and GU (right axis). The weight of CG pair is fixed at 3.

The computation of $\mathrm{S}_{i,j}$ is based on the well known single RNA folding algorithm [37]. For short sequences (i.e., those whose length is strictly less than 5) the score is 0, otherwise, we use the recursion in the second case of Equation (21) shown below. It considers the case where we have an arc $i \bullet j$ and recurs on $[i+1, j-1]$, and also other cases in which the i^{th} and j^{th} bases are not paired and the $[i, j]$ is split into two smaller subsequences:

$$\mathrm{S}_{i,j} = \begin{cases} 0 & j - i < 4 \\ \max\left(\mathrm{S}_{i+1,j-1} + \mathrm{score}(i,j),\ \max_{k=i}^{j-1} \mathrm{S}_{i,k} + \mathrm{S}_{k+1,j}\right) & \text{otherwise.} \end{cases} \tag{21}$$

We define the recurrences for $\mathrm{F}_{i_1,j_1,i_2,j_2}$ similarly. When either sequence is empty, the value is simply the S of the other sequence, and for two singleton sequences, it is the score of the single bond possible. Otherwise we have three cases: (i) i_1 arcs j_1 ($i_1 \bullet j_1$) in which case the residual structure is given by a recursion on $\mathrm{F}_{i_1+1,j_1-1,i_2,j_2}$, (ii) the symmetric case of $i_2 \bullet j_2$ and $\mathrm{F}_{i_1,j_1,i_2+1,j_2-1}$, or (iii) none of these arcs, and two recursive cases of $\mathrm{F}_{i_1,k_1,i_2,k_2}$ and $\mathrm{F}_{k_1+1,j_1,k_2+1,j_2}$. They are illustrated in Figure 14, which lead to

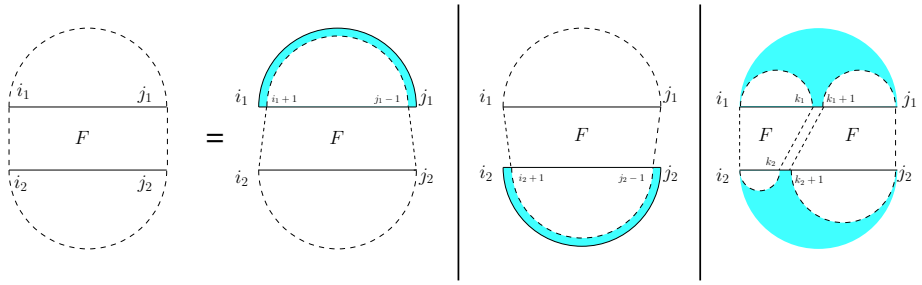


Figure 14 The four cases defining table F. Note that in the `BPMax` algorithm, the cases do not have to be mutually exclusive since we are working with the max operator, which is idempotent.

$$\mathrm{F}_{i_1,j_1,i_2,j_2} = \begin{cases} -\infty & j_1 < i_1 \text{ and } j_2 < i_2 \\ \mathrm{S}^{(1)}_{i_1,j_1} & i_1 \le j_1 \text{ and } j_2 < i_2 \\ \mathrm{S}^{(2)}_{i_2,j_2} & j_1 < i_1 \text{ and } i_2 \le j_2 \\ \mathrm{iscore}(i_1,i_2) & i_1 = j_1 \text{ and } i_2 = j_2 \\ \max\big[\, \mathrm{F}_{i_1+1,j_1-1,i_2,j_2} + \mathrm{score}(i_1,j_1), & \\ \quad \mathrm{F}_{i_1,j_1,i_2+1,j_2-1} + \mathrm{score}(i_2,j_2), & \\ \quad H_{i_1,j_1,i_2,j_2} \,\big] & \text{otherwise,} \end{cases} \tag{22}$$

$$H_{i_1,j_1,i_2,j_2} = \max_{k_1=i_1-1}^{j_1} \max_{k_2=i_2-1}^{j_2} (\mathrm{F}_{i_1,k_1,i_2,k_2} + \mathrm{F}_{k_1+1,j_1,k_2+1,j_2}). \tag{23}$$

Note that H is equivalent to

$$H_{i_1,j_1,i_2,j_2} = \max \begin{pmatrix} \mathrm{S}^{(1)}(i_1,j_1) + \mathrm{S}^{(2)}(i_2,j_2), \\ \max_{k_1=i_1}^{j_1-1} \max_{k_2=i_2}^{j_2-1} \mathrm{F}_{i_1,k_1,i_2,k_2} + \mathrm{F}_{k_1+1,j_1,k_2+1,j_2}, \\ \max_{k_2=i_2}^{j_2-1} \mathrm{S}^{(2)}(i_2,k_2) + \mathrm{F}_{i_1,j_1,k_2+1,j_2}, \\ \max_{k_2=i_2}^{j_2-1} \mathrm{F}_{i_1,j_1,i_2,k_2} + \mathrm{S}^{(2)}(k_2+1,j_2), \\ \max_{k_1=i_1}^{j_1-1} \mathrm{S}^{(1)}(i_1,k_1) + \mathrm{F}_{k_1+1,j_1,i_2,j_2}, \\ \max_{k_1=i_1}^{j_1-1} \mathrm{F}_{i_1,k_1,i_2,j_2} + \mathrm{S}^{(1)}(k_1+1,j_1) \end{pmatrix}. \tag{24}$$

In Equation (22), we compute S tables separately for each strand, according to Equation (21) with the corresponding sequence as the input, and we distinguish them by superscripts $^{(1)}$ and $^{(2)}$ above. We use the same superscript convention throughout this paper.

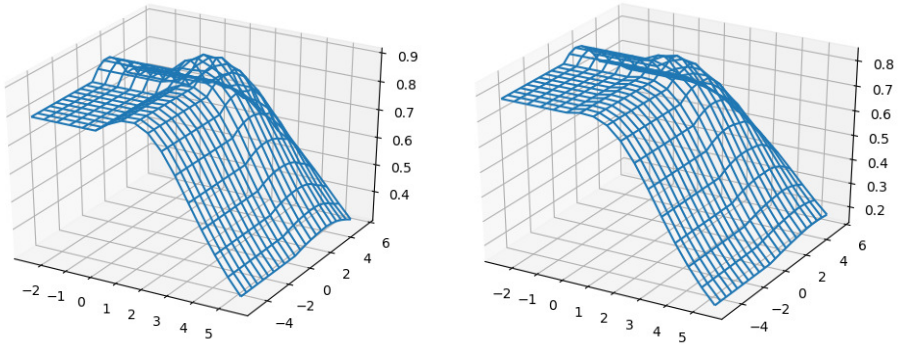


Figure 15 Pearson correlation between `piRNA` and `BPMax` (vertical axis), on the primary dataset, at $-180°C$ (left) and $37°C$ (right) for different values of constant factors (weights) for AU (left axis) and GU (right axis). The weight of CG pair is fixed at 3.

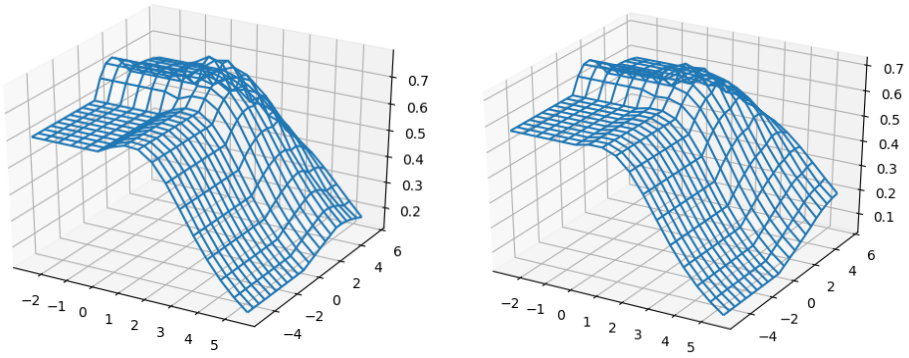


Figure 16 Pearson correlation between `piRNA` and `BPMax` (vertical axis), on the randomly generated dataset, at $-180°C$ (left) and $37°C$ (right) for different values of constant factors (weights) for AU (left axis) and GU (right axis). The weight of CG pair is fixed at 3.

C.1 Results for BPMax

The `BPMax` algorithm was about $1300\times$ faster than $piRNA$ on our primary dataset. We performed similar optimization procedure as the one explained for `BPPart` to obtain optimum weights for the base-pairs that maximize the correlations with `piRNA` scores. Figure 15 showes these optimization plots at $-180°C$ and $37°C$. We did the same analysis on randomly-generate data and presented the optimization plots in Figure 16.

D Application of BPPart in Human Biology

One of the use-cases of `BPPart` and `BPMax`, among others, is making predictions about the consequences of a slight change in the RNA sequences. This information becomes helpful for various domains and tasks, such as synthetic biology and studying the mutations. Between `BPMax` and `BPPart`, the latter is much more sensitive to small changes in the sequence, because it considers all possible structures that the two interacting sequences might form. Therefore, even a missense mutation might make a tangible difference in the computed `BPPart` score.

To verify this hypothesis, we used `BPPart` to study the effects of known missense mutations, provided by Ensembl, in the interaction regions of some RISE pairs. Given a pair of interacting RNAs in RISE for which the information about the interacting regions is provided, we retrieved the data of all the reported missense mutations of those regions from Ensembl API. Also, we got the phenotypic consequences of each mutation from Ensembl. Finally, we computed the `BPPart` score for the original sequence of one of the interacting regions and each of the mutated versions of the other sequence. Among all the generated scores for a pair, we found the outliers using the interquartile range. These outliers represent a mutation in the interacting window of one of the RNA pairs that causes a great difference in the interaction score. In the rest of this section, we almost-randomly pick and narrate two of such cases that we observed, among many discovered ones. In the arxiv version of this paper [14], we report 65 such pairs that have been discovered using this pipeline after analyzing more than one million pairs of sequences that have been generated after applying the known missense mutations to over $15,200$ pairs of interacting sequences reported in the RISE database. Further study of each of these pairs and more comprehensive study of effect of nonsense mutations on RRI would be a next step in the future.

D.1 Traces of TRAF3 in CADASIL

CADASIL is an inherited condition in which the muscle cells of small blood vessels, especially the ones in the brain, gradually die and cause many impairements, such as stroke, cognitive impairement, and mood disorders in the elderly [9]. It has been shown that mutation in NOTCH3, which resides on the reverse strand of chromosome 19, is responsible for this condition in people with this genetic disorder [22]. NOTCH3 and TRAF3 are a pair of interacting RNAs that have been reported in RISE. One of the missense mutations in NOTCH3 that has been reported to be contributing to CADASIL [22] lies within the interacting region of this gene, from loci $15,161,520$ to $15,161,543$ (according to GRCh38 assembly of human genome), with TRAF3. Interestingly, this mutation, which replaces nucletide C with G at loci $15,161,526$ of chromosome 19, causes a dramatic increase in the score of `BPPart` such that it makes it an outlier when the aformentioned procedure is followed. TRAF3 is a gene that has been reported to play a role in angiogenesis [20, 28]. A noticeable increase in the score of `BPPart` increases the chance that these two RNAs interact and cause post-transcriptional conditions that affect the translation rate of TRAF3 which possibly contributes to the phenotypic consequences of CADASIL. Further evaluation and verification of this hypothesis requires further experimental analysis.

D.2 Traces of SNORD3D in Parkinson's Disease

SNORD3D is a small nucleolar RNA which has been detected not long ago [17] with which no specific task or annotation is associated in the literature yet. According to the RISE database, one of the genes that interacts with this snoRNA is GBA. Mutations in GBA has been reported to play a role in Parkinson's disease which is a brain disorder that affects movement and often causes tremors. One of the GBA mutations that is reported to be linked with Parkinson's disease lies within the interaction region of this gene, from loci $155,239,966$ to $155,239,984$ (according to GRCh38 assembly of human genome), with SNORD3D. This specific mutation of GBA, which changes the nucleotide G to C at loci $155,239,972$ of chromosome 1, is one of the cases that is detected as an outlier using our aforementioned analysis of `BPPart` scores. This mutation, when applied to GBA, decreases its score of interaction with SNORD3D, which might cause the interaction to occur much less than the

normal case. This possibly leads to a change in the expression of GBA protein. According to KEGG, GBA is a member of Other glycan degradation, Sphingolipid metabolism, Metabolic pathways, and Lysosome pathways [47]. Therefore, we hypothesize the role of SNORD3D in some or all of those pathways, particularly, the ones that are closely related to Parkinson's disease. Further evaluation of this hypothesis requires further experimental data and analysis.

BISER: Fast Characterization of Segmental Duplication Structure in Multiple Genome Assemblies

Hamza Išerić
Department of Computer Science, University of Victoria, Canada

Can Alkan
Department of Computer Engineering, Bilkent University, Ankara, Turkey

Faraz Hach
Vancouver Prostate Centre, Canada

Ibrahim Numanagić ✉
Department of Computer Science, University of Victoria, Canada

Abstract

The increasing availability of high-quality genome assemblies raised interest in the characterization of genomic architecture. Major architectural parts, such as common repeats and segmental duplications (SDs), increase genome plasticity that stimulates further evolution by changing the genomic structure. However, optimal computation of SDs through standard local alignment algorithms is impractical due to the size of most genomes. A cross-genome evolutionary analysis of SDs is even harder, as one needs to characterize SDs in multiple genomes and find relations between those SDs and unique segments in other genomes. Thus there is a need for fast and accurate algorithms to characterize SD structure in multiple genome assemblies to better understand the evolutionary forces that shaped the genomes of today.

Here we introduce a new tool, BISER, to quickly detect SDs in multiple genomes and identify elementary SDs and core duplicons that drive the formation of such SDs. BISER improves earlier tools by (i) scaling the detection of SDs with low homology (75%) to multiple genomes while introducing further 8–24x speed-ups over the existing tools, and by (ii) characterizing elementary SDs and detecting core duplicons to help trace the evolutionary history of duplications to as far as 90 million years.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases genome analysis, fast alignment, segmental duplications, core duplicons, sequence decomposition

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.15

Supplementary Material BISER is implemented in Seq and C++ programming languages:
Software (Source Code): `https://github.com/0xTCG/biser`
archived at `swh:1:dir:0659879a5b71885efcca60618f9888ee6a5533c6`

Funding H.I. and I.N. were supported by the National Science and Engineering Council of Canada (NSERC) Discovery Grant (RGPIN-04973) and the Canada Research Chairs Program. F.H. was supported by NSERC Discovery Grant (RGPIN-05952) and Michael Smith Foundation for Health Research (MSFHR) Scholar Award (SCH-2020-0370).

Acknowledgements We thank Haris Smajlović for his invaluable comments and suggestions during the manuscript preparation.

1 Introduction

Segmental duplications (SDs), also known as low-copy repeats, are genomic segments larger than 1 Kbp that are duplicated one or more times in a given genome with a high level of homology [8]. While nearly all eukaryotic genomes harbour SDs, it is the human genome that exhibits the largest diversity of SDs. At least 6% of the human genome is covered by SDs ranging from 1 Kbp to a few megabases [8]. The architecture of human SDs also differs from other mammalian species both in its complexity and frequency [27]. For example, while most species harbour tandem SDs, the human genome is repleted with interspersed (both intra- and inter-chromosomal) SD blocks [10]. Human SDs are also often duplicated multiple times within the genome, often immediately next to, or even within an already existing SD cluster. This complex duplication architecture points to a major role that SDs play in human evolution [9, 7, 31]. Human SDs also introduce a significant level of genomic instability that results in increased susceptibility to various diseases [5, 19]. This has led to evolutionary adaptation in the shape of genes and transcripts unique to the human genome that aim to offset the effects of such instability [15]. Finally, SDs display significant diversity across different human populations and can be used as one of the markers for population genetics studies [38].

In order to understand the architecture and evolution of eukaryotic SDs, the first step is detecting all SDs within a given genome. However, SD detection is a computationally costly problem. The theoretically optimal solution to this problem – a local alignment of the entire genome to itself – is unfeasible due to large sizes of eukaryotic genomes that render the classical quadratic time algorithms such as Smith-Waterman impractical. Furthermore, the homology levels between SD copies – as low as 75% – prevent the use of the available edit distance approximations with theoretical guarantees [4, 20]. This is likely to remain so due to the sub-quadratic inapproximability of edit distance metrics [6]. The vast majority of sequence search and whole-genome alignment tools that rely on heuristics to compute the local alignments, such as MUMmer [32] and BLAST [2], also assume high levels of identity between two sequences and therefore are not able to efficiently find evolutionarily older SD regions. Even specialized aligners for noisy long reads, such as Minimap2 [30] or MashMap [24], cannot handle 75% homology that is lower than the expected noise of long reads (up to 15%, although sequencing error rates are improved recently to 5%) [3]. Finally, even if we use higher homology thresholds (such as 90%) to define an SD, the presence of low-complexity repeats and the complex SD rearrangement architecture often prevents the off-the-shelf use of the existing search and alignment tools for detecting SDs.

For these reasons, only a few SD detection tools have been developed in the last two decades, and most of them employ various heuristics and workarounds – often without any theoretical guarantees – to quickly find a set of acceptable SDs. The gold standard for SD detection, Whole-Genome Assembly Comparison (WGAC), uses various techniques such as hard-masking and alignment chunking to find SDs [8]. While its output is used as the canonical set of SDs in the currently available genomes, and as such forms the basis of the vast majority of SD analysis studies, WGAC can only find recent or highly conserved SDs (i.e. SDs with $> 90\%$ homology) within primate lineages. Furthermore, WGAC requires specialized hardware to run and takes several days to complete. Few other tools developed as a replacement for WGAC – namely SDdetector [13] and ASGART [14] – are also limited in their ability to find SDs with less than 90% homology. Currently, the only tools that are able to detect SDs with lower homology are SDquest [34] and SEDEF [33]. SEDEF exploits the unique biological properties of the SD evolutionary process and combines it with

a Poisson error model and MinHash approximation scheme, previously used for long-read alignment [24], to quickly find SDs even with 75% homology – thus enabling the study of SDs as old as 90 million years [33], – while also providing basic theoretical guarantees about the sensitivity of the search process. SDquest, on the other hand, relies on k-mer counting to find seed SD regions that are later extended and aligned with LASTZ [21].

It is important to point out that an SD is often formed by copying parts of older, more ancient SDs to a different location. This, in turn, implies that each SD can be decomposed into a set of short building blocks, where each block either stems from an ancient SD or a newly copied genomic region. Such building blocks are called "elementary SDs" [27]. Elementary SDs are often shared across related species within the same evolutionary branch. It has been proposed that the whole SD formation process is evolutionarily driven by a small subset of elementary SDs, often dubbed *seeds* or *core duplicons*, and that every SD harbours at least one such core duplicon [27]. Based on their cores, SDs can be hierarchically clustered into distinct clades. For example, the human genome SDs can be divided into 435 duplicon blocks that are further classified into 24 clades, seeded by a set of core duplicons with a total span of 2 Mbps that is often gene-rich and transcriptionally active. The prime example of a mosaic-like recombination region that is seeded by an SD core is the *LCR16* locus of the human genome that is shared with many other primates [10].

The SD evolutionary history analysis and the detection of core duplicons require a joint analysis of SDs in many related species. However, while existing SD tools are able to find SDs in single genomes in a reasonable amount of time, none of them can scale – at least not efficiently – to multiple genome assemblies. Furthermore, no publicly available tool is able to provide a deeper understanding of SD evolutionary architecture or find core duplicons across different species, mostly due to the computational complexity of such analysis because of the large number of existing SDs within different species[1]. For these reasons, only a small subset of previously reported core duplicons was analyzed in-depth (e.g. *LCR16* cores), and often so by manually focusing on narrow genomic regions to make the analysis tractable [10]. This has prevented the emergence of a clearer picture of the SD evolution across different species, especially of those SDs that preclude the primate branch of the evolutionary tree.

Here we introduce BISER (**B**risk **I**nference of **S**egmental duplication **E**volutionary st**R**ucture), a new framework implemented in Seq [36] and C++ that is specifically developed to quickly detect SDs even at low homology levels (75%) across multiple related genomes. BISER is also able to infer the elementary and core duplicons and thus allow an evolutionary analysis of all SDs in a given set of related genomes. The key conceptual advances of BISER consist of a novel linear-time algorithm that can quickly detect regions that harbour SDs in a given set of genomes, and a new method for fast SD decomposition into a set of elementary SDs based on the union-find algorithm. BISER can discover, decompose and cluster SDs in the human genome in 60 CPU minutes – an 8× speed-up over SEDEF and 25× speed-up over SDquest – and analyze all shared SDs in seven primate genomes in less than 16 CPU hours, translating to 2.5 hours on a standard 8-core laptop computer. The flexibility of BISER will make it a useful tool for SD characterizations that will open doors towards a better understanding of the complex evolutionary architecture of these functionally important genomic events.

[1] The source code that was used for older analyses [27] is not publicly available. SDquest, on the other hand, is able to detect elementary SDs but only at the single genome level.

2 Methods

2.1 Preliminaries

Consider a genomic sequence $G = g_1 g_2 g_3 \dots g_{|G|}$ of length $|G|$ and alphabet $\Sigma = \{A, C, G, T, N\}$. Let $G_i = g_i \dots g_{i+n-1}$ be a substring of G of length n that starts at position i in G. To simplify the notation, the length is assumed to be n. We will use an explicit notation $G_{i:i+n}$ for a substring of length n starting at position i when a need arises. Let $s_1 \circ s_2$ represent a string concatenation of strings s_1 and s_2.

Segmental duplications are long, low-copy repeats generated during genome evolution over millions of years. Following such an event, different copies of a repeat get subjected to different sets of mutations, causing them to diverge from each other over time. Thus, it is necessary to introduce a similarity metric between two strings in order to detect SDs in a given genome. To that end, we use the Levenshtein's [28] *edit distance* metric $\mathcal{E}$ between two strings s and s' that measures the minimum number of edit operations (i.e., substitutions, insertions, and deletions at the single nucleotide level) in the alignment of s and s'. Let ℓ be the length of such alignment; it is clear that $\max(|s|, |s'|) \leq \ell \leq |s| + |s'|$. We can also define an *edit error* $\mathrm{err}(\cdot, \cdot)$ between s and s' (or, in the context of this paper, an *error*) as the normalized edit distance: $\mathrm{err}(s, s') = \mathcal{E}(s, s')/\ell$. Intuitively, this corresponds to the sequence divergence of s and s'. Now we can formally define an SD as follows:

▶ **Definition 1.** *A segmental duplication (SD) within the error threshold ε is a tuple of paralog sequences (G_i, G_j) that satisfies the following criteria:*

1. $\mathrm{err}(G_i, G_j) \leq \varepsilon$*;*
2. $\ell \geq 1,000$ *where ℓ is the length of the optimal alignment between G_i and G_j; and*
3. *the overlap between G_i and G_j is at most $\varepsilon \cdot n$* [2]*.*

Given a set of genomes $G^1, \dots, G^\gamma$ and their mutual evolutionary relationships, our goal is to:

- find a set of valid SDs, $\mathcal{SD}^i$, within each G^i **(SD detection)**;
- find all copies of both s and s' for $(s, s') \in \mathcal{SD}^i$ in other genomes $G^j, j \neq i$, if such copies exist **(SD cross-species conservation detection)**; and
- decompose each SD from $\mathcal{SD} = \mathcal{SD}^1 \cup \dots \cup \mathcal{SD}^\gamma$ into a set of *elementary SDs* E, and determine the set of core duplicons that drive the formation of SDs in $\mathcal{SD}$ **(SD decomposition)**.

To that end, we present BISER, a computational framework that is able to efficiently perform these steps, and we describe the algorithms behind it in the following sections. For the sake of clarity, unless otherwise noted, we assume that we operate on a single genome G. Since SDs are by definition different from low-complexity repeats and transposons, we also assume that all genomes $G^1, \dots, G^\gamma$ are hard-masked and, as such, do not contain such elements.

2.1.1 SD Error Model

Different paralogs of an SD are mutated independently of each other. Therefore, the sequence similarity of paralogs is correlated with the age of the duplication event – more recent copies are nearly identical, while distant ancestral copies are dissimilar. It has been proposed that

[2] Ideally, the SD mates should not overlap; however, due to the presence of errors, we need to account for at most $\varepsilon \cdot n$ overlap.

the sequence similarity of older SDs (e.g., those shared by the mouse and human genomes) falls as low as 75% [33]. In other words, the error between different copies of an old SDs exceeds 25% (i.e., $\mathrm{err}(s, s') \geq 0.25$ for SD paralogs s and s', according the definition above).

Detection of duplicated regions within such a large error threshold is a challenging problem, as nearly any edit distance approximation technique with or without theoretical guarantees breaks down at such high levels of dissimilarity [4, 24], provided that this error is truly random. However, that is not the case: we have previously shown that the SD mutation process is an amalgamation of two independent mutation processes, namely the background point mutations (also known as *paralogous sequence variants*, or PSVs) and the large-scale block edits. As such, the overall error rate ε can be expressed as a sum of two independent error rates, ε_P (PSV mutation rate) and ε_B (block edit rate), where only ε_P is driven by a truly random mutation process.

In the case when paralogs share the 75% sequence identity, it has been shown that the random point mutations (PSVs) contribute at most 15% ($\varepsilon_P \leq 0.15$) towards the total error ε [33] (this also holds for many other mammalian genomes, as their substitution rate is often lower than the human substitution rate [16]). The remaining 10% is assumed to correspond to the block edit rate ε_B. Note that these mutations are clustered *block* errors and as such are randomly distributed across SD regions. The probability of a large block event is roughly 0.005 based on the analysis of existing SD calls.

On the other hand, we assume that PSVs between two SD paralogs s and s' follow a Poisson error model [24, 17], and that those mutations occur independently from each other. It follows that any k-mer in s' has accumulated on average $k \cdot \varepsilon_P$ mutations compared to the originating k-mer in s, provided that such k-mer was part of the original copy event. By setting a Poisson parameter $\lambda = k \cdot \varepsilon_P$, we obtain the probability of a duplication event in which a k-mer is preserved in both SD paralogs (i.e., that a k-mer is error-free) to be $e^{-k\varepsilon_P}$.

Let us return to the main problem of determining whether two strings s and s' are "similar enough" to be classified as SDs. As mentioned before, classical edit distance calculation algorithms would be too slow for this purpose. Instead, we use an indirect approach that measures the similarity of strings s and s' by counting the number of shared k-mers in their respective k-mer sets $\mathbf{K}(s)$ and $\mathbf{K}(s')$. It has been shown that Jaccard index ($\mathcal{J}(\mathbf{K}(s), \mathbf{K}(s')) = \frac{\mathbf{K}(s) \cap \mathbf{K}(s')}{\mathbf{K}(s) \cup \mathbf{K}(s')}$) is a good proxy for $\mathcal{E}(s, s')$ under the Poisson error model [24]. Thus we can combine the Poisson error model with the SD error model, and obtain the expected value of Jaccard index τ between any two strings s and s', whose edit error $\mathrm{err}(s, s')$ follows the SD error model and is lower than $\varepsilon = \varepsilon_P + \varepsilon_B$ to be [33]:

$$\tau = \mathbb{E}[\mathcal{J}(\mathbf{K}(s), \mathbf{K}(s'))] \geq \frac{1 - \varepsilon_B}{1 + \varepsilon_B} \cdot \frac{1}{2e^{k\varepsilon_P} - 1}. \tag{1}$$

As we cannot use local alignment to efficiently enumerate all SDs in a given genome due to both time and space complexity, we utilize a heuristic approach to enumerate all pairs of regions in G that are likely to harbour one or more segmental duplications. We call these pairs *putative SDs*. These pairs are not guaranteed to contain a "true" SD and must be later aligned to each other to ascertain the presence of true SDs. Nevertheless, such an approach will *filter out* the regions that do not harbour SDs, and thus significantly reduce the amount of work needed for detecting "true" SDs. The overall performance of our method, both in terms of performance and sensitivity, will depend on how well the putative SDs are chosen.

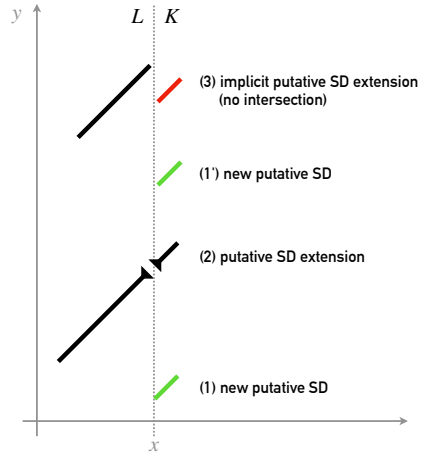
The problem of putative SD detection can be, thanks to the SD error model, easily expressed as an instance of a filtering problem: find all pairs of indices i, j in G such that $\mathcal{J}(\mathbf{K}(G_i), \mathbf{K}(G_j)) \geq \tau$, where τ is the lower bound from the Equation 1. Here we assume that the size of G_i and G_j exceeds the SD length threshold (1,000 bp), and no k-mer occurs twice in either G_i or G_j.

2.2 SD Detection

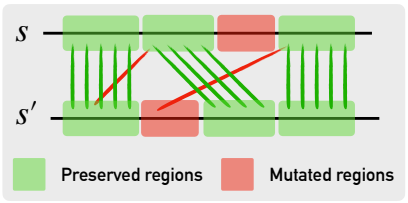
Algorithm 1 Algorithm for finding putative SDs.

Input : Genomic sequence G, its k-mer index I_G, threshold Δ.
Output : A list $\mathcal{SD}$ of putative SDs.
$L \leftarrow []$; $\mathcal{SD} \leftarrow []$;
for $x \leftarrow 1$ **to** $|G|$ **do**
  $K \leftarrow I_G[G_{x:x+k}]$; $(i_K, i_L) \leftarrow (1, 1)$;
  append (x, K_{i_K}, k) to L if L is empty;
  while $i_K \leq |K|$ **and** $i_L \leq |L|$ **do**
    $y \leftarrow K_{i_K}$; $(\ell_x, \ell_y, l) \leftarrow L_{i_L}$;
(1) **if** $y < \ell_y$ **then**
      insert (x, y, k) to L before i_L;
      advance i_K and i_L;
(2) **else if** $y \geq \ell_y + l$ **and** $\max(x - \ell_x, y - \ell_y) - l \leq \Delta$ **then**
      extend L_{i_L} to cover $G_{x:x+k}$ and $G_{y:y+k}$;
      increase counts for $\bigcup(L_{i_L})$ and $\bigcap(L_{i_L})$;
      advance i_K; $i_L \leftarrow$ `CheckJaccard` (L_{i_L})
(1') **else if** $y \leq start(L_{i_L+1})$ **then**
      insert (x, y, k) to L before i_L;
      advance i_K and i_L;
(3) **else**
      increase count for $\bigcup(L_{i_L})$;
      $i_L \leftarrow$ `CheckJaccard` (L_{i_L})
    end
  end
end

(a)



(b)



(c)

Figure 1 **(a)** A plane-sweep algorithm for finding putative SDs. **(b)** Visual guide for the algorithm. The algorithm sweeps a vertical dashed line through the set of winnowed k-mers in a genome G (represented by the x axis). At each k-mer starting at the location x, it queries the index I_G to obtain a sorted list K of k's occurrences in G (shown on the right side of the sweep line). The algorithm then scans K and the list L of putative SDs found thus far at the same time. At each step, it examines i_L-th element of L and i_K-th element of K, and decides whether to start a new putative SD (cases (1) and (1'), green k-mers on the right), extend the current putative SD with the current k-mer (case (2), black k-mer on the right), or subsume the current k-mer within the current putative SD (case (3), red k-mer). In all of these cases, the algorithm updates the counts of k-mer union $\bigcup_L$ and intersection $\bigcap_L$ for each processed putative SD (note that it only updates the counts and does not maintain the sets themselves). The `CheckJaccard` procedure marks each putative SD in L that satisfies the SD criteria as "good" and removes it from L when the extension is complete. **(c)** A visual representation of a valid k-mer matching in a valid alignment (shown by green lines). Any addition of a red matching to the set of green matchings would render the alignment invalid as red matchings are not co-linear with the green matchings (in other words, they "cross" the existing matchings).

As we cannot use local alignment to efficiently enumerate all SDs in a given genome due to both time and space complexity, we utilize a heuristic approach to enumerate all pairs of regions in G that are likely to harbour one or more segmental duplications. We call these pairs *putative SDs*. These pairs are not guaranteed to contain a "true" SD and must be later aligned to each other to ascertain the presence of true SDs. Nevertheless, such an approach

will *filter out* the regions that do not harbour SDs, and thus significantly reduce the amount of work needed for detecting "true" SDs. The overall performance of our method, both in terms of performance and sensitivity, will depend on how well the putative SDs are chosen.

The problem of putative SD detection can be, thanks to the SD error model, easily expressed as an instance of a filtering problem: find all pairs of indices i, j in G such that that $\mathcal{J}(\mathbf{K}(G_i), \mathbf{K}(G_j)) \geq \tau$, where τ is the lower bound from the Equation 1. Here we assume that the size of G_i and G_j exceeds the SD length threshold (1,000 bp) and no k-mer occurs twice in either G_i or G_j.

The filtering approach has already been successfully used in other software packages and forms the backbone of both SEDEF (SD detection tool) and MashMap (Nanopore read aligner).

However, both methods maintain the k-mer sets $\mathbf{K}(s)$ and $\mathbf{K}(s')$ to calculate the Jaccard index between the sequences s and s'.

As these methods dynamically grow s and s' (as the length n is not known in advance), the corresponding sets $\mathbf{K}(s)$ and $\mathbf{K}(s')$ are constantly being updated, necessitating a costly recalculation of $\mathbf{K}(s) \cap \mathbf{K}(s')$ on each update. A common trick is to use the MinHash technique to reduce the sizes of $\mathbf{K}(s)$ and $\mathbf{K}(s')$, and thus the frequency of such updates. However, the frequent recalculation of the Jaccard index still remains a major bottleneck even in the MinHash-based approaches.

Here we note that the Jaccard index calculation can be significantly simplified by not having to maintain the complete k-mer sets $\mathbf{K}(s)$ and $\mathbf{K}(s')$. The need for keeping such sets arises from the fact that calculation of $\mathbf{K}(s) \cap \mathbf{K}(s')$ allows any k-mer in $\mathbf{K}(s')$ to match any k-mer in $\mathbf{K}(s)$. However, such a loose intersection requirement is not only redundant for approximation of edit distance under the SD error model but is even undesirable as such intersections can introduce a cross-over k-mer matches that are not possible in the edit distance metric space (see Figure 1c for an example of valid and invalid matchings).

By disallowing such cross-over cases, we can significantly optimize the calculation of the Jaccard index. Let us show how to do that without sacrificing sensitivity. Let us first introduce $s \circledast s'$ as an alternative way of measuring the k-mer similarity between strings s and s'.

For that purpose, let us introduce a notion of a *co-linear k-mer matching* between s and s' as a set of index pairs (i, j) $(1 \leq i \leq |s|, 1 \leq j \leq |s'|)$ such that the k-mers that start at i and j in s and s' respectively are equal, and such that all pairs (i, j) in a matching are co-linear (i.e. for each (i, j) and (i', j'), either $i < i'$ and $j < j'$, or $i > i'$ and $j > j'$).

A $\circledast$ operation describes the size of a maximum co-linear matching of k-mers between s and s'. In other words, we want to select a maximal set of matching k-mers in $\mathbf{K}(s)$ and $\mathbf{K}(s')$ such that no two k-mer matchings cross-over each other (see Figure 1c for an example of cross-over, or non-co-linear, matchings). We can replace $\mathbf{K}(s) \cap \mathbf{K}(s')$ with $s \circledast s'$ and introduce an *ordered Jaccard index* $\hat{\mathcal{J}}(s, s')$, formally defined as:

$$\hat{\mathcal{J}}(s, s') = \frac{s \circledast s'}{\mathbf{K}(s) \cup \mathbf{K}(s')}.$$

The following lemma allows us to use an ordered Jaccard index $\hat{\mathcal{J}}$ in lieu of classical Jaccard index $\mathcal{J}$:

▶ **Lemma 2.** *The ordered Jaccard index $\hat{\mathcal{J}}(s, s')$ of two strings s and s' is equal to the Jaccard index $\mathcal{J}(\mathbf{K}(s), \mathbf{K}(s'))$ (under the assumptions of SD error model, namely the separation of ϵ_B and ϵ_P), assuming that s and s' only share k-mers that have not been modified by PSVs following the originating copy event.*

Proof. It is sufficient to prove that the size of $|\mathbf{K}(s) \cap \mathbf{K}(s')|$ always corresponds to the size of maximal co-linear matching between s and s'.

To show that $s \circledast s' \leq |\mathbf{K}(s) \cap \mathbf{K}(s')|$, it is enough to note that matched k-mers in any matching are by definition identical, and thus belong to $\mathbf{K}(s) \cap \mathbf{K}(s')$.

We will prove that $s \circledast s' \geq |\mathbf{K}(s) \cap \mathbf{K}(s')|$ by contradiction. First, note that the string s is equal to s' immediately after the duplication event (i.e. before the occurrence of PSVs) and that all k-mers are co-linear in their maximal matching because s contains no repeated k-mers (an assumption made by the SD error model). Now, suppose that there is a cross-over in $\mathbf{K}(s) \cap \mathbf{K}(s')$. That implies either a cross-over between s and s' before PSVs occurred – contradicting the previous observation – or a cross-over after it, contradicting the assumption that any matched k-mer pair was matched before the occurrence of PSVs. Hence $\mathbf{K}(s) \cap \mathbf{K}(s')$ cannot contain any cross-overs, and $s \circledast s' = |\mathbf{K}(s) \cap \mathbf{K}(s')|$. ◀

If the conditions of Lemma 2 are satisfied, we can calculate $s \circledast s'$ in linear time by a simple scan through s and s' at the same time. A linear calculation of $s \circledast s'$, together with the fact that the lower bound τ in Equation 1 equally holds for $\hat{\mathcal{J}}$ as well (a consequence of Lemma 2), allows us to use a plane sweep technique to select all pairs of substrings (s, s') in G whose ordered Jaccard distance $\hat{\mathcal{J}}(s, s')$ exceeds τ, and as a result, select all putative SDs in G (see Figure 1 for details).

We begin by creating a k-mer index I_G that connects each k-mer in G to an ordered list of its respective locations in G. Then we sweep a vertical line in G from left to right while maintaining a sorted list L of putative SDs found thus far. For each location x in G encountered by a sweep line, we query I_G to obtain a sorted list K containing loci of $G_{x:x+k}$'s copies in G. Then, for any y in K, we check if it either (1) begins a new potential putative SD that maps x to y, (2) extends an existing putative SD, or (3) is covered by existing putative SD in L. If a putative SD in L is too distant from y, it is promoted to the final list of putative SD regions if it satisfies the ordered Jaccard index threshold τ and the other SD criteria from the Definition 1. Note that we do not allow a k-mer to extend a putative SD if the distance between it and the SD exceeds the user-defined threshold Δ (set to 250 by default). It takes $|L| + |K|$ steps to process each k-mer in G because both L and K are sorted. However, because the size of $|L|$ is kept low by the distance criteria, and because $|K|$ is low enough in practice, [3], the time complexity of Algorithm 1 is $O(|G|)$ for constructing the index I_G, and linear in terms of the genome size for plane sweeping.

The key assumption in Lemma 2 – that two paralogs only share the k-mers that have not been mutated since the copy event – does not always hold in practice on real data. As such, the Algorithm 1 might occasionally under-estimate the value of $\hat{\mathcal{J}}$, leading potentially to some false negatives. We control that by using Δ – the same parameter that controls the growth of putative SDs by limiting the maximum distance of neighbouring k-mers in $s \circledast s'$ (Figure 1) – to limit the growth of under-estimated SDs and thus start the growth of potentially more successful SDs earlier. This heuristic might cause a large SD to be reported as a set of smaller disjoint SD regions. For that reason, we post-process the set of putative SDs upon the completion of Algorithm 1 and merge together any two SDs that are close to each other if their union satisfies the ordered Jaccard index criteria. We also extend each putative SD by 5 Kbp both upstream and downstream to account for the small SD regions that might have been filtered out during the search process. This parameter is user-defined and might be adjusted for different genome assemblies.

[3] The average size of L in our experiments was 370, and the average size of K is 30.

The performance of the plane sweep technique can be further improved by winnowing the set of k-mers used for the construction of I_G [24]. Instead of indexing all k-mers in G, we only consider k-mers in a *winnowing fingerprint* $W(G)$ of G. $W(G)$ is calculated by sliding a window of size w through G and by taking in each window a lexicographically smallest k-mer (the rightmost k-mer is selected in case of a tie). The expected size of $W(G)$ for a random sequence G is $2|G|/(w+1)$ [35]. The main benefit of winnowing is that it can significantly speed up the search step (up to an order of magnitude) without sacrificing sensitivity. The winnow $W(G)$ can be computed in a streaming fashion in linear time using $O(w)$ space with the appropriate data structures [11].

Following the discovery of putative SDs, we locally align paralogs from each putative SD and only keep those regions whose size satisfies the SD criteria mentioned above. BISER uses a two-tiered local chaining algorithm described previously in SEDEF that uses a seed-and-extend approach and efficient $O(n \log n)$ chaining method following by a SIMD-parallelized sparse dynamic programming algorithm to calculate the boundaries of the final SD regions and their alignments [1, 30, 39].

2.3 SD Decomposition

Once the set of final SDs $\mathcal{SD} = \{(s_1, s'_1), \dots\}$ is discovered and the precise global alignment of each paralog pair $(s, s') \in \mathcal{SD}$ is calculated, we proceed by decomposing the set $\mathcal{SD}$ into a set of evolutionary building blocks called *elementary SDs*. More formally, we aim to find a minimal set of elementary SDs $E = \{e_1, \dots, e_{|E|}\}$, such that each SD paralog s is a concatenation of $\hat{e}^s_1 \circ \cdots \circ \hat{e}^s_{n_s}$. Each $\hat{e}_i$ either belongs to E or there is some $e_j \in E$ such that $\mathrm{err}(\hat{e}_i, e_j) \leq \varepsilon$. An example of such a decomposition is given in Figure 2.

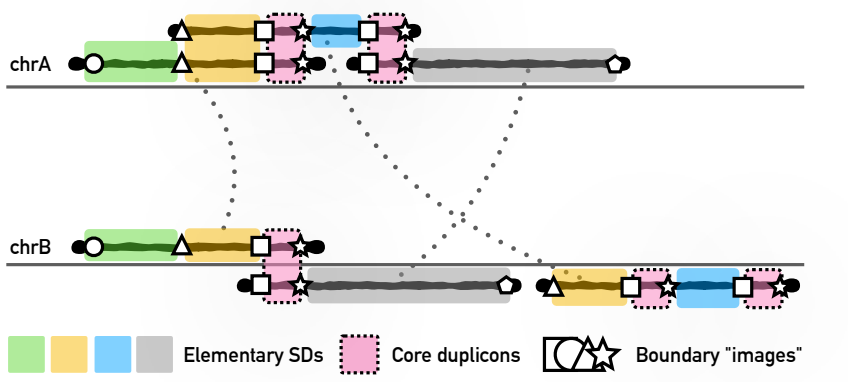


Figure 2 A decomposition of three partially overlapping SDs into a set of elementary SDs. Each paralog pair is indicated by a pair of two thick black lines linked by a dashed line. Each elementary SD is represented as a coloured box. The boxes of core duplicons – elementary SDs shared by all SDs – are depicted with a dashed border. Note that a boundary of each elementary SD is induced by a boundary of an existing SD. Different boundaries are represented by different shapes, and their images (paralog copies) also share the same shape. For the sake of simplicity, we only show the identifiers (shapes) for locations that define elementary SD boundaries.

Note that each locus covered by an SD paralog is either copied to another locus during the formation of that SD (in other words, it is "mirrored" by its paralog) or belongs to an alignment gap. As SD events can copy over the regions that already form an existing SD, a single locus might "mirror" a large number of existing locations. In order to find all locations that a locus i mirrors, we use a modification of Tarjan's union-find disjoint set

algorithm [41] to link together all mirrored locations. This algorithm begins by giving each locus in a genome covered by an existing SD a distinct identifier (represented by a distinct shape in Figure 2). It then iterates over the set of SDs, and for each pair of SD paralogs (s, s') uses the global alignment between s and s' to link the identifiers of any two loci that are mirrored by the SD event. Linking merges the two identifiers into a single identifier. Upon the completion of the algorithm, all copies ("mirrors") of each locus will share the same identifier.

Note that the boundaries of SD paralogs, or their images, correspond to the boundaries of elementary SDs, as each paralog by definition starts and ends with an elementary SD (Figure 2). However, as the set of elementary SDs should be minimal, it is not only necessary but sufficient to focus only on the identifiers that belong to a boundary of an existing SD paralog in order to construct the set of elementary SDs E. These identifiers describe a set of locations in G that form the boundaries of elementary SDs. We can obtain the final set E by iterating over G and checking if a locus is identified with a boundary-covering identifier.

In practice, SD boundaries and SD alignments are highly uneven, and SDs themselves exhibit a complex mosaic structure that often introduces "mirror loops" [34] that can collapse multiple unrelated loci into a single identifier. BISER handles these cases by discarding any mirrored loci that lie within a close distance of an already existing elementary SD boundary.

After decomposing SDs into the set of elementary SDs E, we select some of them as *core duplicons*. We define these duplicons as the minimal set of elementary SDs that cover all existing SDs (an SD is covered by an elementary if either paralog is composed of that elementary SD). We use a classical set-cover approximation algorithm [12] to determine a set of core duplicons from E.

2.4 Multiple Genomes

The above method can be efficiently scaled to γ distinct genomes $G^1, \ldots, G^\gamma$ by constructing a composite k-mer index $I_{G^1 \cup \ldots \cup G^\gamma}$ and by running the search procedure on each G^i in parallel. We only need to ensure that the SD overlap criteria are enforced only if a k-mer belongs to the same genome that is currently being searched. Note that the size of the genome index grows sub-linearly with the number of genomes, as most genomes – especially mammalian ones – share the large number of common k-mers. Also, note that this method can be trivially extended to search for reverse-complemented SD copies by adding an additional iteration over $\bar{G}$, where $\bar{G}$ is a reverse complement of G.

The detection of cross-species conserved SD regions happens automatically when using a composite index $I_{G^1 \cup \ldots \cup G^\gamma}$. However, as such procedure does not distinguish between conserved SDs and other conserved regions, BISER uses additional checks to ensure that every reported conservation is also a valid SD in at least one of the provided genomes.

3 Results

We have evaluated all stages of BISER for speed and accuracy on both simulated and real-data datasets. All results were obtained on a multi-core Intel Xeon 8260 CPU (2.40GHz) machine with 1 TB of RAM. The run times are rounded to the nearest minute and are reported for both single-core as well as multi-core (8 CPU cores) modes when ran in parallel via GNU Parallel [40]. All real-data genomes were hard-masked, and all basepair coverage statistics are provided with respect to the hard-masked genomes.

In our experiments, we used $k = 14$ when searching for putative SDs and $k = 10$ during the alignment step (note that both parameters are user-adjustable). The size of the winnowing window was set to 16. We found that the lower values of k significantly impact the running time without providing any visible improvement to the detection sensitivity, while higher values of k significantly lower the detection sensitivity.

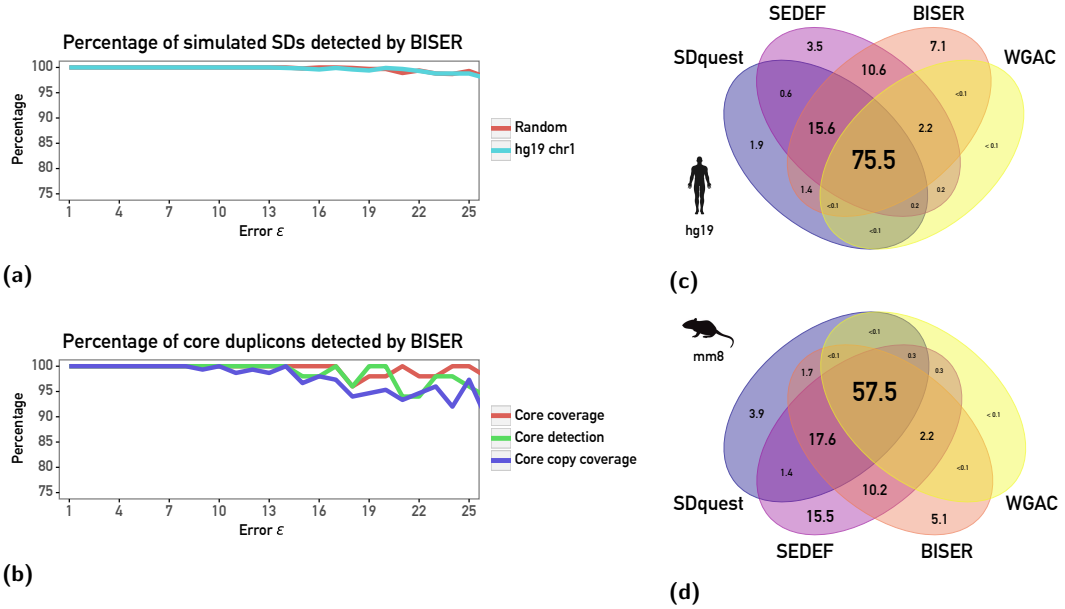


Figure 3 **(a)** Performance of BISER's algorithm on simulated SDs. x-axis represents the simulated SD error rate ε, while y axis represents the percentage of correctly detected SDs. Note that the plot area is cropped as BISER detects more than 98% of simulated SDs for any $\varepsilon \leq 0.25$. **(b)** Performance of BISER's core duplicon detection on simulated cores. The red line shows the percentage of ancestral core locations covered by a detected SD; the green line shows the percentage of cores identified as such; while the blue line shows the coverage of later core copies covered by a detected SD. Note that the plot area is also cropped. **(c)** Venn diagram depicts the SD coverage of the BISER, WGAC, SEDEF and SDquest (in Mbp) on the hard-masked human genome (hg19). **(d)** Venn diagram depicts the SD coverage of the BISER, WGAC, SEDEF and SDquest (in Mbp) on the hard-masked mouse genome (mm8). Note that nearly all bases out of ≈17 Mbp bases that are shown to be unique to SEDEF (and not covered by BISER) map to gaps and low-copy repeats and should be therefore treated as noise (not true SDs).

3.1 Simulations

3.1.1 SD detection

The accuracy of using the strong Jaccard index together with the SD error model as a function of error parameter ε, as well as the overall sensitivity of BISER's SD detection pipeline, was evaluated on a set of 1,000 simulated segmental duplications ranging from 1 to 100Kbp. All sequences and mutations were randomly generated with uniform distribution according to the SD error model with $\varepsilon \in \{0.01, 0.02, \ldots, 0.25\}$ (i.e., we allowed the overall error rate to reach 25%). We consider a simulated SD as being "covered" if BISER found an SD that covers more than 90% of the original SD's basepairs. As shown in Figure 3a, the overall sensitivity is around 99% even for $\varepsilon = 0.25$.

We performed the same experiment on human (hg19) chromosome 1 (Figure 3a), where we selected uniformly at random 10,000 sequences of various lengths and duplicated them within the chromosome. Each duplication was followed by introducing random PSVs according to the SD error model while varying the values of ε as described above. Even in this case, BISER's performance stays the same, and only a handful of very small SDs (of size ≈1,000) were missed.

3.1.2 Core duplicon detection

We also devised a simulation experiment to measure the power of BISER's core duplicon detection module. We began with a random DNA sequence of size 10 Mbp. Then we simulated an evolutionary process by introducing a set of novel SDs for 50 generations. In each generation, we introduced a novel core duplicon that does not overlap with an existing core, and we introduce a random number of SD events according to the SD error model that contains at least one core duplicon introduced thus far. In each iteration, we made sure that the difference between paralogs does not exceed $\varepsilon \in \{0.01, 0.02, \ldots, 0.25\}$ regardless of their age. We finally used BISER to analyze the final sequence to predict SDs and their elementary decomposition, as well as the core duplicons. BISER was able to successfully cover all ancestral cores, and properly decompose them into the elementary SDs and finally identify them as core duplicons (Figure 3b). Furthermore, BISER covered 95% or more of the more recent copies of the ancestral cores. We note that the combination of large block indels leads BISER to occasionally report a single ancestral core as a set of 2 or more core duplicons or to report an elementary SD that covers less than 90% of a core. These cases are rare and happen less than 8% of the time at the highest levels of ε. The sensitivity of core duplicon detection is further described in Figure 3b.

Table 1 Running time performance of BISER (single-core and 8-core mode) on Intel Xeon 8260 CPU at 2.40 GHz for single genomes (hg19 and mm8).

	Single genome (hg19)			
	Total	Putative SDs	Alignment	Decomposition
1 core	**1h 8m**	44m	24m	<1m
8 cores	**10m**	6m	4m	<1m
	Single genome (mm8)			
	Total	Putative SDs	Alignment	Decomposition
1 core	**1h 26m**	40m	46m	<1m
8 cores	**13m**	5m	8m	<1m

Table 2 Running time performance of BISER (single-core and 8-core mode) on Intel Xeon 8260 CPU at 2.40 GHz for seven genomes.

	Seven genomes (see below)			
	Total	Putative SDs	Alignment	Decomposition
1 core	**30h 23m**	11h 58m	18h 19m	6m
8 cores	**4h 30m**	1h 54m	2h 29m	6m

3.2 Single-genome results

We have run BISER on the *H.sapiens* hg19 genome and *M.musculus* mm8 genome, and compared it to the published WGAC [8][4], SEDEF [33] and SDquest [34] SD calls. We also compared the runtime performance of BISER to that of SEDEF and SDquest. Note that we were not able to run WGAC due to the lack of hardware necessary for its execution. We did not compare BISER to other SD detection tools – namely SDdetector [13], MashMap2 [25] and ASGART [14] – as it has been previously shown that these tools underperform when compared to SEDEF or SDquest, and require an order of magnitude more resources than either SEDEF or SDquest do (see [33] for the detailed comparisons with these tools). For the same reason, we did not compare BISER to whole-genome aligners such as Minimap2 [30] and MUMmer/nucmer [32], as well as DupMasker [26], as none of these tools were designed to detect *de novo* SDs in a genome.

BISER was able to find and align all SD regions in hg19 in 10 minutes on 8 cores (or ≈one hour on a single core) (Table 1). To put this into perspective, BISER is nearly 8× faster than SEDEF, 24× faster than SDquest, and an order of magnitude faster than WGAC that takes days to find human SDs (personal communication; we were not able to run WGAC pipeline ourselves due to legacy hardware requirements). As a side note, BISER has the same memory requirements as SEDEF or SDquest and needs around 5 GB of RAM per core. Since SEDEF by default operates on a genome that is not hard-masked, we also ran SEDEF on a hard-masked genome to measure its theoretical speed (note that SEDEF was not designed for hard-masked genomes; thus, the basepair analysis is omitted). SEDEF took 21 minutes on 8 CPU cores to process a hard-masked hg19, leaving it still >2× slower than BISER. Similar performance gains were observed on the mm8 genome as well.

In terms of sensitivity, BISER discovers about 1 GB of putative SD regions. After the alignment step, BISER reports 112 Mb of final SD regions within the 75% edit distance in hg19. That is 34 Mbp more than WGAC and 17 Mbp more than SDquest. The total coverage of SEDEF and BISER are similar to each other, differing by 3 Mbp uniquely detected by SEDEF and 7 Mbp uniquely covered by BISER (Table 3). BISER misses a few Mbp of SD regions unique to SDquest and a negligible amount unique to WGAC (Figure 3). On the mm8 genome, we can observe similar trends. However, we also observed that SEDEF covers roughly 17 Mbp not covered by BISER (Figure 3). When SEDEF is run on a hard-masked genome, it does not cover these bases; further analysis showed that nearly all bases (≥16.3 Mbp) originally reported as unique to SEDEF actually map either to alignment gaps, soft-masked repeat elements, or small islands (<200bp) between the low-copy repeats and as such do not constitute "true" SDs.

BISER found roughly ≈67,000 elementary SDs that describe hg19 SD calls. Of those, 2,759 were identified as core duplicons. BISER's core duplicons cover all 100 of the core duplicons reported in the earlier work [27], including the cores from the *LCR16* cluster. Note that many previously identified core duplicons in the hg17 version of the human genome turned out to be short tandem repeats in the hg19 version. The whole decomposition process took less than a minute on the final set of ≈58,000 SDs.

[4] `http://humanparalogy.gs.washington.edu`

Table 3 SD coverage of the human and mouse genomes (hg19 and mm8) and the runtime performance of BISER, SEDEF and SDquest. "Missed" and "Extra" columns are calculated with respect to the WGAC SD calls. All running times are reported on 8 CPU cores. We could not run WGAC as we do not have access to the legacy hardware needed for its execution; the reported runtime is from [33].

hg19

Tool	**Covered** (MBp)	**Missed** (MBp)	**Extra** (MBp)	**Time**
WGAC (gold standard)	78.2			days
BISER	112.4	0.4	34.6	10m
SEDEF	108.4	0.1	30.3	1h 15m
SDquest	95.2	2.5	19.5	3h 56m

mm8

Tool	**Covered** (MBp)	**Missed** (MBp)	**Extra** (MBp)	**Time**
WGAC (gold standard)	60.6			days
BISER	94.4	0.8	34.6	13m
SEDEF	105.2	0.1	44.7	1h 24m
SDquest	82.5	2.7	24.5	6h 06m

3.3 Multi-genome results

In addition to running BISER on a single genome, we also ran BISER on the following seven related genomes:

- *C.jacchus* (marmoset, version calJac3),
- *M.mulatta* (macaque, version rheMac10),
- *G.gorilla* (gorilla, version gorGor6),
- *P.abelii* (orangutan, version ponAbe3),
- *P.troglodytes* (chimpanzee, version panTro6),
- *H.sapiens* (human, version hg19), and
- *M.musculus* (mouse, version mm8).

These genomes were analyzed in the previous work [10] with the sole exception of *M.musculus* that is novel to this analysis.

BISER took around four and a half hours to complete the run on 8 cores. Of that, it took around 2 hours to find putative SDs within the same species (42 minutes for in-species detection and 71 minutes for detecting SDs conserved across different species). The remaining time (2h 29m) was spent calculating the final alignments for all reported SDs (Table 2). The vast majority of alignment time (1h 37m minutes out of 2h 29m) was spent only on aligning putative SDs from calJac3 genome. We presume that this is due to the high presence of unmasked low-complexity regions in this particular assembly.

The SD decomposition and core duplicon detection took slightly less than 6 minutes to complete on a set of nearly 2,407,000 SDs. BISER found ≈116,000 elementary SD sets seeded by ≈13,000 cores. All cores from [27] and [10] were covered by BISER's cores as well.

4 Conclusion

More than a decade ago, the Genome 10K Project Consortium proposed to build genome assemblies for 10,000 species [18]. Due to the lack of high-quality long-read sequencing data, this aim was not immediately realized. However, the Genome 10K Project spearheaded

the development of other large-scale many-genome sequencing projects such as the Earth BioGenome Project [29] and Vertebrate Genomes Project[5]. Recent developments in generating more accurate long-read sequencing data, coupled with better algorithms to assemble genomes now promise to make the aforementioned and similar projects feasible.

Analyzing the recently and soon-to-be generated genome assemblies to understand evolution requires the development of various algorithms for different purposes, from gene annotation [37] to orthology analysis [22] and the selection and recombination analysis [23]. Although a handful of tools such as SEDEF and SDquest are now available to characterize segmental duplications in genome assemblies, they cannot perform multi-species SD analysis, and they suffer from computational requirements. We developed BISER as a new segmental duplication characterization algorithm to be added to the arsenal of evolution analysis tools.

We demonstrate that (1) BISER is substantially faster than earlier tools; (2) it can characterize SDs in multiple genomes to delineate the evolutionary history of duplications; and (3) it can identify elementary SDs and core duplicons to help understand the mechanisms that give rise to SDs. We believe that BISER will be a powerful and common tool and will contribute to our understanding of SD evolution when thousands of genome assemblies become available in the next few years. The next step in this line of research would consist of interpreting BISER's results, biological analysis of the reported core duplicons, and applying BISER to a larger set of available mammalian genomes to infer the evolutionary history of ancient duplications.

References

1 Mohamed Ibrahim Abouelhoda and Enno Ohlebusch. Multiple genome alignment: Chaining algorithms revisited. In Ricardo Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Combinatorial Pattern Matching*, pages 1–16. Springer Berlin Heidelberg, 2003.

2 S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990. doi:10.1016/S0022-2836(05)80360-2.

3 Shanika L. Amarasinghe, Shian Su, Xueyi Dong, Luke Zappia, Matthew E. Ritchie, and Quentin Gouil. Opportunities and challenges in long-read sequencing data analysis. *Genome Biology*, 21:30, 2020. doi:10.1186/s13059-020-1935-5.

4 A. Andoni, R. Krauthgamer, and K. Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proc. IEEE 51st Annual Symp. Foundations of Computer Science*, pages 377–386, October 2010. doi:10.1109/FOCS.2010.43.

5 Francesca Antonacci, Jeffrey M Kidd, Tomas Marques-Bonet, Brian Teague, Mario Ventura, Santhosh Girirajan, Can Alkan, Catarina D Campbell, Laura Vives, Maika Malig, Jill A Rosenfeld, Blake C Ballif, Lisa G Shaffer, Tina A Graves, Richard K Wilson, David C Schwartz, and Evan E Eichler. A large and complex structural polymorphism at 16p12.1 underlies microdeletion disease risk. *Nat Genet*, 42(9):745–750, September 2010. doi:10.1038/ng.643.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746612.

7 J. A. Bailey, J. M. Kidd, and E. E. Eichler. Human copy number polymorphic genes. *Cytogenet Genome Res*, 123(1-4):234–243, 2008. doi:10.1159/000184713.

8 J. A. Bailey, A. M. Yavor, H. F. Massa, B. J. Trask, and E. E. Eichler. Segmental duplications: organization and impact within the current human genome project assembly. *Genome Res*, 11(6):1005–1017, June 2001. doi:10.1101/gr.187101.

[5] https://vertebrategenomesproject.org/

9 Jeffrey A Bailey and Evan E Eichler. Primate segmental duplications: crucibles of evolution, diversity and disease. *Nat Rev Genet*, 7(7):552–564, July 2006. doi:10.1038/nrg1895.

10 Stuart Cantsilieris, Susan M. Sunkin, Matthew E. Johnson, Fabio Anaclerio, John Huddleston, Carl Baker, Max L. Dougherty, Jason G. Underwood, Arvis Sulovari, PingHsun Hsieh, Yafei Mao, Claudia Rita Catacchio, Maika Malig, AnneMarie E. Welch, Melanie Sorensen, Katherine M. Munson, Weihong Jiang, Santhosh Girirajan, Mario Ventura, Bruce T. Lamb, Ronald A. Conlon, and Evan E. Eichler. An evolutionary driver of interspersed segmental duplications in primates. *Genome biology*, 21:202, 2020. doi:10.1186/s13059-020-02074-4.

11 Keegan Carruthers-Smith. Sliding window minimum implementations, 2013. last accessed 28 January 2021. URL: SlidingWindowMinimumImplementations.

12 Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

13 Jean-Félix Dallery, Nicolas Lapalu, Antonios Zampounis, Sandrine Pigné, Isabelle Luyten, Joëlle Amselem, Alexander H. J. Wittenberg, Shiguo Zhou, Marisa V. de Queiroz, Guillaume P. Robin, Annie Auger, Matthieu Hainaut, Bernard Henrissat, Ki-Tae Kim, Yong-Hwan Lee, Olivier Lespinet, David C. Schwartz, Michael R. Thon, and Richard J. O'Connell. Gapless genome assembly of colletotrichum higginsianum reveals chromosome structure and association of transposable elements with secondary metabolite gene clusters. *BMC genomics*, 18:667, 2017. doi:10.1186/s12864-017-4083-x.

14 Franklin Delehelle, Sylvain Cussat-Blanc, Jean-Marc Alliot, Hervé Luga, and Patricia Balaresque. ASGART: fast and parallel genome scale segmental duplications mapping. *Bioinformatics*, 34:2708–2714, 2018. doi:10.1093/bioinformatics/bty172.

15 Max L. Dougherty, Jason G. Underwood, Bradley J. Nelson, Elizabeth Tseng, Katherine M. Munson, Osnat Penn, Tomasz J. Nowakowski, Alex A. Pollen, and Evan E. Eichler. Transcriptional fates of human-specific segmental duplications in brain. *Genome research*, 28:1566–1576, 2018. doi:10.1101/gr.237610.118.

16 John W. Drake, Brian Charlesworth, Deborah Charlesworth, and James F. Crow. Rates of spontaneous mutation. *Genetics*, 148(4):1667–1686, 1998. arXiv:https://www.genetics.org/content/148/4/1667.full.pdf.

17 Huan Fan, Anthony R Ives, Yann Surget-Groba, and Charles H Cannon. An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data. *BMC genomics*, 16:522, July 2015. doi:10.1186/s12864-015-1647-5.

18 Genome 10K Community of Scientists. Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. *J Hered*, 100(6):659–674, 2009. doi:10.1093/jhered/esp086.

19 Santhosh Girirajan, Megan Y. Dennis, Carl Baker, Maika Malig, Bradley P. Coe, Catarina D. Campbell, Kenneth Mark, Tiffany H. Vu, Can Alkan, Ze Cheng, Leslie G. Biesecker, Raphael Bernier, and Evan E. Eichler. Refinement and discovery of new hotspots of copy-number variation associated with autism spectrum disorder. *Am J Hum Genet*, 92(2):221–237, February 2013. doi:10.1016/j.ajhg.2012.12.016.

20 Hiroyuki Hanada, Mineichi Kudo, and Atsuyoshi Nakamura. On practical accuracy of edit distance approximation algorithms. *arXiv preprint arXiv:1701.06134*, 2017. arXiv:1701.06134v1.

21 Robert S. Harris. *Improved Pairwise Alignment of Genomic Dna*. PhD thesis, Pennsylvania State University, University Park, PA, USA, 2007. AAI3299002.

22 Xiao Hu and Iddo Friedberg. SwiftOrtho: A fast, memory-efficient, multiple genome orthology classifier. *GigaScience*, 8, October 2019. doi:10.1093/gigascience/giz118.

23 Martin Hölzer and Manja Marz. PoSeiDon: a Nextflow pipeline for the detection of evolutionary recombination events and positive selection. *Bioinformatics*, July 2020. doi:10.1093/bioinformatics/btaa695.

24 Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. In S. Cenk Sahinalp, editor, *Proceedings of 21st Annual International Conference on Research in Computational Molecular Biology (RECOMB 2017)*, volume 10229, pages 66–81, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-56970-3_5.

25 Chirag Jain, Sergey Koren, Alexander Dilthey, Adam M Phillippy, and Srinivas Aluru. A fast adaptive algorithm for computing whole-genome homology maps. *Bioinformatics*, 34(17):i748–i756, 2018.

26 Zhaoshi Jiang, Robert Hubley, Arian Smit, and Evan E. Eichler. Dupmasker: a tool for annotating primate segmental duplications. *Genome research*, 18:1362–1368, August 2008. doi:10.1101/gr.078477.108.

27 Zhaoshi Jiang, Haixu Tang, Mario Ventura, Maria Francesca Cardone, Tomas Marques-Bonet, Xinwei She, Pavel A Pevzner, and Evan E Eichler. Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. *Nature genetics*, 39:1361–1368, November 2007. doi:10.1038/ng.2007.9.

28 Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

29 Harris A. Lewin, Gene E. Robinson, W. John Kress, William J. Baker, Jonathan Coddington, Keith A. Crandall, Richard Durbin, Scott V. Edwards, Félix Forest, M. Thomas P. Gilbert, Melissa M. Goldstein, Igor V. Grigoriev, Kevin J. Hackett, David Haussler, Erich D. Jarvis, Warren E. Johnson, Aristides Patrinos, Stephen Richards, Juan Carlos Castilla-Rubio, Marie-Anne van Sluys, Pamela S. Soltis, Xun Xu, Huanming Yang, and Guojie Zhang. Earth BioGenome Project: Sequencing life for the future of life. *Proceedings of the National Academy of Sciences of the United States of America*, 115:4325–4333, April 2018. doi:10.1073/pnas.1720115115.

30 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics (Oxford, England)*, 34:3094–3100, September 2018. doi:10.1093/bioinformatics/bty191.

31 Tomas Marques-Bonet, Jeffrey M Kidd, Mario Ventura, Tina A Graves, Ze Cheng, LaDeana W Hillier, Zhaoshi Jiang, Carl Baker, Ray Malfavon-Borja, Lucinda A Fulton, Can Alkan, Gozde Aksay, Santhosh Girirajan, Priscillia Siswara, Lin Chen, Maria Francesca Cardone, Arcadi Navarro, Elaine R Mardis, Richard K Wilson, and Evan E Eichler. A burst of segmental duplications in the genome of the African great ape ancestor. *Nature*, 457(7231):877–881, February 2009. doi:10.1038/nature07744.

32 Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy, Rachel Coston, Steven L. Salzberg, and Aleksey Zimin. MUMmer4: A fast and versatile genome alignment system. *PLoS computational biology*, 14:e1005944, January 2018. doi:10.1371/journal.pcbi.1005944.

33 Ibrahim Numanagić, Alim S Gökkaya, Lillian Zhang, Bonnie Berger, Can Alkan, and Faraz Hach. Fast characterization of segmental duplications in genome assemblies. *Bioinformatics*, 34:i706–i714, September 2018. doi:10.1093/bioinformatics/bty586.

34 Lianrong Pu, Yu Lin, and Pavel A Pevzner. Detection and analysis of ancient segmental duplications in mammalian genomes. *Genome research*, 28:901–909, June 2018. doi:10.1101/gr.228718.117.

35 Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM, 2003.

36 Ariya Shajii, Ibrahim Numanagić, Riyadh Baghdadi, Bonnie Berger, and Saman Amarasinghe. Seq: A high-performance language for bioinformatics. *Proc. ACM Program. Lang.*, 3, October 2019. doi:10.1145/3360551.

37 Alaina Shumate and Steven L. Salzberg. Liftoff: accurate mapping of gene annotations. *Bioinformatics*, December 2020. doi:10.1093/bioinformatics/btaa1016.

38 Peter H Sudmant, Jacob O Kitzman, Francesca Antonacci, Can Alkan, Maika Malig, Anya Tsalenko, Nick Sampas, Laurakay Bruhn, Jay Shendure, 1000 Genomes Project, and Evan E Eichler. Diversity of human copy number variation and multicopy genes. *Science*, 330(6004):641–646, October 2010. doi:10.1126/science.1197005.

39 Hajime Suzuki and Masahiro Kasahara. Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC bioinformatics*, 19(1):33–47, 2018.

40 O. Tange. GNU Parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, February 2011. doi:10.5281/zenodo.16303.

41 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979. doi:10.1016/0022-0000(79)90042-4.

Flow Decomposition with Subpath Constraints

Lucia Williams
School of Computing, Montana State University, Bozeman, MT, USA

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland

Brendan Mumey
School of Computing, Montana State University, Bozeman, MT, USA

Abstract

Flow network decomposition is a natural model for problems where we are given a flow network arising from superimposing a set of weighted paths and would like to recover the underlying data, i.e., *decompose* the flow into the original paths and their weights. Thus, variations on flow decomposition are often used as subroutines in multiassembly problems such as RNA transcript assembly. In practice, we frequently have access to information beyond flow values in the form of *subpaths*, and many tools incorporate these heuristically. But despite acknowledging their utility in practice, previous work has not formally addressed the effect of subpath constraints on the accuracy of flow network decomposition approaches. We formalize the *flow decomposition with subpath constraints* problem, give the first algorithms for it, and study its usefulness for recovering ground truth decompositions. For finding a minimum decomposition, we propose both a heuristic and an FPT algorithm. Experiments on RNA transcript datasets show that for instances with larger solution path sets, the addition of subpath constraints finds 13% more ground truth solutions when minimal decompositions are found exactly, and 30% more ground truth solutions when minimal decompositions are found heuristically.

2012 ACM Subject Classification Theory of computation → Network flows; Applied computing → Computational transcriptomics

Keywords and phrases Flow decomposition, subpath constraints, RNA-Seq

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.16

Supplementary Material *Software (Source Code)*: `https://github.com/msu-alglab/coaster`

Funding This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO), the Academy of Finland (grants No. 322595, 328877), the US Fulbright program, the Fulbright Finland Foundation, the Helsinki Institute for Information Technology (HIIT), as well as the US National Science Foundation (grant DBI-1759522).

Acknowledgements Computational efforts were performed on the Hyalite High Performance Computing System, operated and supported by University Information Technology Research Cyberinfrastructure at Montana State University.

1 Introduction

Flow networks are useful models in many domains, from transportation planning to computational biology. In some cases, the flow on a graph arises as the superposition of some set of weighted paths, such as trips through a road network, routing of information through a communication network, or paths in a graph encoding mixed reads sequenced from several biological sequences, as in the case of RNA transcripts through a splice graph.

In many such applications, we are actually presented with the inverse problem: given a flow in a graph, we need to recover the initial paths that made up the flow. This problem is also referred to as the *flow decomposition (FD) problem*. In computational biology, this is a common subroutine in multiassembly problems, such as RNA transcript assembly or viral quasispecies assembly. Prioritizing parsimonious solutions proved to be an accurate

assembly method, but it can suffer when there are multiple parsimonious solution to choose from. As such, in this paper we consider a natural generalization of the flow decomposition problem, by assuming that extra information about the initial paths is available in the form of *subpath constraints*. These are paths in the network graph that must be followed by at least one path in the flow decomposition; thus, we are looking for flow decompositions with the property that every constraint is a subpath of some decomposition path. We call the resulting problem *flow decomposition with subpath constraints (FDSC)*.

Biological setting. Algorithms that solve variations of the flow decomposition problem are at the heart of most RNA transcript assembly software, including IsoLasso [16], Traph [25], FlipFlop [9], Scallop [21] and StringTie [18]. More recently, flow decomposition methods were used for another multi-assembly problem, namely strain-aware genome assembly, with applications to viral quasispecies assembly [4, 2]. Briefly, flow decomposition methods for sequence assembly work by using reads and their abundances to first construct a flow network whose vertices may represent exons (in the case of an RNA splice graph) or k-mers (in the case of a de Bruijn graph). Edges in the network are present if there is read evidence that some sequence followed the edge (e.g. two exons are consecutive in some transcript). Furthermore, each edge is weighted by the number of reads that support it. With perfect data, we might expect the weights to directly provide a flow in the network; however in practice some adjustment to the weights may be needed to achieve a flow. One such method uses a minimum-cost flow approach for this adjustment [25]. Another approach [28] models the input as an *inexact flow* network in which edge flows belong to intervals, that are estimated from the data. In all cases, we seek a path decomposition for the flow network that minimizes the number of paths.

In Kloster et al. [13] it was shown that in the case of RNA transcripts, most of the time the "true" transcripts also provide a minimum flow decomposition of the splice graph. However, there can often be more than one solution to the minimum flow decomposition problem; indeed, Kloster et al. found that, when the number of true transcripts is seven, the minimum flow decomposition found corresponds to the true paths in only 80% of the instances of that size, with lower accuracies as the number of true paths increases. In fact, practical methods for RNA assembly methods also have a precision of 50%-60% on some human datasets [23, 18]. Adding subpath constraints to the flow decomposition problem may further restrict the solution space, thus improving the RNA assembly accuracy.

In practice, the subpath constraints can be derived from reads overlapping three or more nodes of the flow graph. Long RNA-Seq reads naturally have this property in many cases; however, also short reads can exhibit this behavior in the case of short exons. As we review below, other possible sources of such constraints exist in practice as well, such as from partial assemblies, or *super-reads* [18] constructed from short reads that can be uniquely extended.

Finally, most of the RNA assembly tools cited above work in a so-called *genome-guided* setting in which also a reference genome of the studied species is available. This makes the splice graph acyclic (i.e. a *DAG*). While both the original flow decomposition problem and our variant with subpath constraints can be defined in flow networks with cycles (which would correspond to a *de novo assembly* setting), in this paper we focus on DAGs only.

Related work. Finding a flow decomposition with the minimum number of weighted paths is a well-studied problem in computer science. Even when restricted to DAGs, the minimum FD problem is NP-hard [27], and thus various practical approaches to it exist: approximation algorithms [11, 24, 19, 17, 5, 6], FPT algorithms [13], greedy algorithms [27, 22]. By taking

the set of subpaths constraints to be empty (or to correspond to all edges of the graph with non-zero flow), it follows that also finding a solution to the FDSC problem with a *minimum* number of paths is NP-hard.

The idea of improving RNA assembly by multi-edge subpath information is in fact used by several flow-based tools, such as [21] and StringTie [18]. However, both approaches integrate subpaths in a heuristic manner, with no overall formulation of the computational problem they are solving. The same holds also for the viral quasispecies assembler [4]. Recently, the method TransBorrow [29] uses partial assemblies from different RNA assembly tools, and works by heuristically extending the subpaths they correspond to in a splice graph.

Moreover, our FDSC problem generalizes a related problem on DAGs. Recall that in the *minimum path cover (MPC)* problem, we are looking for a minimum-cardinality set of path that together cover all nodes of a DAG (e.g. "explain" all exons of a splice graph). The problem is behind early RNA assembly methods such as Cufflinks [26], and early virus quasispecies assembly methods such as ShoRAH [30]. The MPC problem has been extended to include subpath constraints as well [20, 15, 8, 14], by analogously requiring that each constraint is a subpath of some solution path. While these generalizations are polynomially-time solvable, they (together with the initial MPC formulations) are usually unsatisfactory since they ignore the weights of the graph (i.e. the abundances of the reads) – recall that most state-of-the-art RNA assembly methods cited above are flow-based. Moreover, MPCs and MPCs with subpath constraints correspond to restricted classes of flows in some DAG [7, 20], and thus the minimum FDSC problem is a strict generalization of the MPC problem with subpath constraints.

Contributions. In this work, we initiate the formal study of the FDSC problem. This is a natural model for multiassembly problems, as seen by the abundance of methods and tools that incorporate subpath information for improving RNA and viral quasispecies assemblies. However, because finding a *minimum* solution to the FDSC problem is NP-hard, these methods and tools have focused on either heuristic approaches or a polynomial-time solvable particular version of the problem (MPC) that ignores valuable edge weight information. Here, we make two advances that bring us closer to being able to use the complete version of the problem in practical tools. On the theoretical side, we formalize the problem and give the first algorithm to determine whether an instance is feasible (Theorem 17), and produce a solution if it is. The algorithm works via a reduction to the standard flow decomposition problem where any solution must translate to a solution in the original graph that satisfies all of the subpath constraints. Additionally, we give an FPT algorithm for the minimum FDSC problem (Theorem 19), extending the one of Kloster et al. [13].

We implement both of these algorithms, and perform a proof-of-concept study of their usefulness in RNA assembly. We experiment on a dataset developed by Shao et al. [22] to study their heuristic for the minimum FD problem. The same dataset was then used by Kloster et al. [13], who focused on studying the usefulness of standard minimum flow decompositions in RNA assembly, as explained above. We find that our FDSC algorithms increase our ability to uncover the ground truth RNA transcripts, as more and longer subpath constraints are included in the input. This holds both when minimality is enforced, through our FPT algorithm, and when it is only heuristically sought, through the flow decomposition reduction and an associated path reduction heuristic. For example, when there are seven ground truth transcripts, we increase the accuracy by 13% when an optimal solution is found (via FPT) and 30% when a heuristic solution is found.

When no edge appears in more subpath constraints than its flow value, our FDSC algorithm runs in polynomial time (i.e. always finds a solution to the FDSC problem, not necessarily minimum). Though we desire a minimal such path decomposition, algorithms that guarantee such solutions in general may be too slow to be used in practice. Despite a lack of minimality guarantees in our heuristic FDSC algorithm, our experiments show that the addition of subpath constraints yields solutions that approach the accuracy of a minimum decomposition without subpath constraints; thus, our results show that heuristic FDSC is a practical substitute for minimum FD without subpath constraints. On the other hand, when for some edges the number of subpath constraints exceed its flow value, our algorithm takes exponential time even to decide whether an FDSC solution exists (not necessarily minimum). We leave open the complexity of the FDSC problem (e.g. bring it in P, or classify it as NP-complete). This is an interesting question especially since finding one standard flow decomposition can be done in polynomial time [1]. Since flow decompositions are basic concepts in flow networks with a large body of research, we believe that also its natural generalization as the FDSC problem may lead to further developments in network flow theory and in its various applications.

2 Preliminaries

Since flow weights represent read counts, we restrict attention to integral flow networks and flow decompositions.

▶ **Definition 1.** *A* flow network $G = (V, E, f)$ *is a directed acyclic graph (DAG) comprised of a set of vertices* V *containing a source* s *and sink* t*, a set of directed edges* E*, and a flow function* $f : E \to \mathbb{N}$*, such that for* $v \in V \setminus \{s, t\}$*,*

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w). \tag{1}$$

Finally, for each $v \in V$*, there is an* s*-*t *path in* G *that includes* v*.*

▶ **Definition 2** (Flow decomposition)**.** *A* flow decomposition *for a flow network* $G = (V, E, f)$ *consists of set of* s*-*t *paths* $\mathcal{P} = (P_1, \ldots, P_k)$ *and associated integral flow weights,* $w = (w_1, \ldots, w_k)$ *with* $w_i \in \mathbb{N}$ *such that for each edge* $e \in E$*,*

$$\sum_{i:e\in P_i} w_i = f(e). \tag{2}$$

We define several problems concerning finding decompositions of flow networks into paths.

▶ **Problem 3** (MFD)**.** *Given a flow network* $G = (V, E, f)$*, the* minimum flow decomposition problem *is to find a decomposition* $(\mathcal{P}, w)$ *such that* $|\mathcal{P}|$ *is minimized.*

▶ **Definition 4** (Flow decomposition with subpath constraints)**.** *Let* $G = (V, E, f)$ *be a flow network.* Subpath constraints *are defined to be a set of simple paths* $\mathcal{R} = \{R_1, \ldots, R_\ell\}$ *in* G *such that no* $R_i \subseteq R_j$*. A flow decomposition* $(\mathcal{P}, w)$ *satisfies the subpath constraints if and only if*

$$\forall R_i \in \mathcal{R}\ \exists P_j \in \mathcal{P} \text{ such that } R_i \text{ is a subpath of } P_j \text{ (in short, } R_i \in P_j\text{).} \tag{3}$$

Figure 1 shows an example of a flow network with subpath constraints.

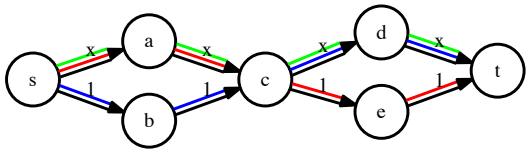


Figure 1 An example FDSC flow network with the flow values of the edges being 1 or x; the colored paths indicate the subpath constraints. If $x = 1$, then the instance is infeasible. If $x = 2$, then the instance is feasible and requires three paths to decompose (whereas the associated FD instance without subpath constraints can be decomposed with two paths in both cases).

▶ **Problem 5** (FDSC). *Given a flow network $G = (V, E, f)$ and subpath constraints $\mathcal{R}$, the* flow decomposition with subpath constraints *problem is to determine if there exists, and if so, find a flow decomposition $(\mathcal{P}, w)$ satisfying (3).*

▶ **Problem 6** (MFDSC). *Given a flow network $G = (V, E, f)$ and subpath constraints $\mathcal{R}$, the* minimum flow decomposition with subpath constraints *problem is to determine if there exists, and if so, find a flow decomposition $(\mathcal{P}, w)$ satisfying (3) such that $|\mathcal{P}|$ is minimized.*

3 Algorithms

3.1 FDSC reduces to FD

We now describe a reduction from the FDSC problem to the FD problem. The idea is to convert subpath constraints into edges in an FD instance so that any path decomposition of the FD instance translates into a path decomposition for the FDSC instance that covers the subpath constraints.

Given a flow network $G = (V, E, f)$ with subpath constraints $\mathcal{R}$, we define the *overdemand* of an edge as

$$d_o(e) = \max(0, |\{i : e \in R_i\}| - f(e)), \tag{4}$$

and say that e is *overdemanded* if $d_o(e) > 0$. The FDSC problem $(G, \mathcal{R})$ may be feasible if multiple subpaths covering e are satisfied by a single path in a path decomposition. If no edges are overdemanded, we can give a simple reduction from FDSC to FD by transforming all subpath constraints in the FDSC instance into edges in the FD instance.

▶ **Lemma 7.** *Let $G = (V, E, f)$ be a flow network with subpath constraints $\mathcal{R}$ such that no edge is overdemanded. Let $G' = (V, E', f')$ be the flow network that results from converting every subpath constraint $R_i = [v_1, v_2, \ldots, v_{|R_i|}]$ to a* bridge edge *$e_i = (v_1, v_{|R_i|})$ with $f'(e_i) = 1$ and subtracting one from the flow values on the edges it covers. That is, for all $e \in E$, $f'(e) = f(e) - |\{i : e \in R_i\}|$. G' is a flow network.*

Proof. Consider building G' from G iteratively by converting each subpath constraint into a new edge and subtracting its flow from the edges it covers. At each step, conservation of flow holds. Thus, after the final step, f' is a flow on G'. Additionally, because no edge is overdemanded, all flow values in f' are nonnegative. Thus, G' is a flow network. ◀

▶ **Lemma 8.** *Let G and G' be as described in Lemma 7. Let $(\mathcal{P}', w)$ be a size k solution to the FD problem on G'. There exists a size k solution to the FDSC problem on G.*

Proof. We show how to construct a size k solution to FDSC on G from $(\mathcal{P}', w)$. For each path $P' \in \mathcal{P}'$, create a new path P by replacing all bridge edges e'_i with the original sequence of nodes R_i. By construction, R_i must form a path from the start node of e_i to the end node of e_i in P, and so P is a valid path from s to t in G. We take $\mathcal{P}$ to be the set of all k such paths P. We now must show that $(\mathcal{P}, w)$ forms a flow decomposition with subpath constraints for G.

Let e be any edge in G and let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of subpath constraints containing e. We can divide the paths in $\mathcal{P}$ that cover e into two disjoint sets: $\mathcal{P}_B$, those that covered bridge edges $e_i : R_i \in \mathcal{R}'$, and $\mathcal{P}_O$, those those that covered the original edge e in G'. Because $(\mathcal{P}', w)$ is a flow decomposition for G', each path in $\mathcal{P}_B$ must have unit weight. Thus, paths in $\mathcal{P}_B$ contribute $|\{i : R_i \in \mathcal{R}'\}|$ to e's flow. On the other hand, paths in $\mathcal{P}_O$ must cover e's flow in G', which is $f(e) - |\{i : R_i \in \mathcal{R}'\}|$. Thus, paths from $\mathcal{P}_B$ and $\mathcal{P}_O$ together cover e with exactly $f(e)$ units of flow. Additionally, $\mathcal{P}_B$ must satisfy all of the subpath constraints $\mathcal{R}'$, so together $\mathcal{P}_B$ and $\mathcal{P}_O$ do as well. ◀

Because any FDSC instance without overdemanded edges can be solved by reduction to FD, it follows that all FDSC instances without overdemanded edges are feasible.

▶ **Corollary 9.** *Let $G = (V, E, f)$ be a flow network with subpath constraints $\mathcal{R}$. A sufficient condition for a flow decomposition to exist is that no edge is overdemanded.*

When an FDSC instance has an overdemanded edge, the reduction given above fails, because any overdemanded edge would have a negative flow value after subtracting all of its demands from its original flow. However, if the FDSC instance $(G, \mathcal{R})$ is feasible, it is possible to first transform $(G, \mathcal{R})$ to an FDSC instance $(G, \mathcal{R}^*)$, where no edge is overdemanded and any path decomposition for $(G, \mathcal{R}^*)$ also provides a feasible path decomposition for $(G, \mathcal{R})$. By Lemma 7, $(G, \mathcal{R}^*)$ can be solved via reduction to an FD instance.

▶ **Lemma 10.** *Let $(G, \mathcal{R})$ be a feasible FDSC instance with overdemanded edge e and $(\mathcal{P}, w)$ be a path decomposition for $(G, \mathcal{R})$. Let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of subpath constraints that contain e. There must be some $P \in \mathcal{P}$ such that $|\{R_i : R_i \in \mathcal{R}', R_i \in P\}| \geq 2$.*

Proof. Suppose not. That is, suppose $(\mathcal{P}, w)$ is a path decomposition for $(G, \mathcal{R})$ but no path in $\mathcal{P}$ covers two or more subpath constraints in $\mathcal{R}'$ completely. This means that every subpath constraint in $\mathcal{R}'$ must be satisfied by a different path; call this set of paths $\mathcal{P}'$ and let the total weight assigned to these paths be $w' \geq |\mathcal{P}'| = |\mathcal{R}'| = |\{i : e \in R_i\}|$. As e is overdemanded, we have $|\{i : e \in R_i\}| > f(e)$. But then $w' > f(e)$, contradicting the fact that $(\mathcal{P}, w)$ is a path decomposition for $(G, \mathcal{R})$. ◀

▶ **Definition 11** (Compatible Subpaths)**.** *Two subpaths $R_i, R_j \in \mathcal{R}$ are* compatible *if and only if R_i and R_j have a suffix-prefix overlap (so that $R_i \cup R_j$ forms a simple path in G).*

▶ **Definition 12** (Directly Compatible Subpaths)**.** *Two subpaths $R_i, R_j \in \mathcal{R}$ are* directly compatible *if and only if R_i and R_j are compatible and there does not exist a subpath R_k such that R_i and R_k are compatible and R_k and R_j are compatible.*

We remark that the directly compatible relation is just the transitive reduction of the compatible relation.

▶ **Lemma 13.** *Let $(G, \mathcal{R})$ be an FDSC instance with some overdemanded edge e. Then $(\mathcal{P}, w)$ is a solution for $(G, \mathcal{R})$ if and only if there exist directly compatible subpaths R_i and R_j, each containing e, such that $(\mathcal{P}, w)$ is a solution for $(G, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\})$.*

Proof. ($\rightarrow$) Let $(G, \mathcal{R})$ be a feasible FDSC instance with overdemanded edge e. Let $(\mathcal{P}, w)$ be a path decomposition for $(G, \mathcal{R})$. Let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of subpath constraints that contain e. By Lemma 10, there exists a $P \in \mathcal{P}$ and $R_i, R_j \in \mathcal{R}'$ such that $R_i \neq R_j$ and $R_i, R_j \in P$. Since R_i and R_j both belong to P and overlap (since they each contain e), it follows that they are compatible. If R_i and R_j are not directly compatible, there must exist some R_k such that R_i and R_k both contain e and are directly compatible. In this case, take R_j to be R_k. Furthermore, the path P satisfies the subpath constraint $R_i \cup R_j$, so $(\mathcal{P}, w)$ is a feasible solution for $(\mathcal{G}, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\})$.

($\leftarrow$) Let R_i and R_j be directly compatible subpaths that both contain e. Let $(\mathcal{P}, w)$ be a feasible solution to $(\mathcal{G}, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\})$. It follows that there exists a path $P \in \mathcal{P}$ that covers $R_i \cup R_j$. Clearly, P also covers R_i and R_j, so $(\mathcal{P}, w)$ is also a feasible solution for $(G, \mathcal{R})$. ◀

▶ **Corollary 14.** *Let $(G, \mathcal{R})$ be an FDSC instance. If there are no compatible subpaths R_i and R_j containing some overdemanded edge e, then $(G, \mathcal{R})$ is infeasible.*

▶ **Definition 15** (Total Overdemand)**.** *Let $(G, \mathcal{R})$ be an FDSC instance. The* total overdemand *is defined as*

$$D_o(G, \mathcal{R}) = \sum_{e \in E} d_o(e). \tag{5}$$

▶ **Lemma 16.** *Let $(G, \mathcal{R})$ be an FDSC instance with directly compatible constraints subpaths R_i and R_j, each containing some overdemanded edge e. Then, $D_o(\mathcal{G}, \mathcal{R} \cup \{R_i \cup R_j\} \setminus \{R_i, R_j\}) < D_o(G, \mathcal{R})$.*

Proof. All overdemanded edges in $R_i \cap R_j$ have their overdemand decreased by one. Since $e \in R_i \cap R_j$, there is at least one such edge. The overdemand of all remaining edges is unchanged. ◀

▶ **Theorem 17.** *Let $(G, \mathcal{R})$ be an FDSC instance with $|\mathcal{R}| = \ell$ and at least one overdemanded edge. In $O(\ell^{2 \cdot \min(D_o(G,\mathcal{R}),\ell)}|E| + \ell^3)$ time, we can determine whether $(G, \mathcal{R})$ is feasible, and if so, reduce $(G, \mathcal{R})$ to an FD instance on a modified network with at most ℓ additional edges.*

Proof. Lemma 13 suggests a strategy for processing FDSC instances with overdemanded edges: pick an overdemanded edge e and recursively test each possible union of directly compatible pairs of subpath constraints that contain e. If there are no compatible subpath constraints for some overdemanded edge, then by Corollary 14, that branch of the recursion cannot lead to a feasible solution. On the other hand, if we reach an FDSC instance with no overdemanded edges, then we can solve this instance using the reduction to FD described in Lemma 7.

The branches of this recursive algorithm form a tree with maximum branching factor $\binom{\ell}{2}$ (since we must try each pair of directly compatible constraint subpaths containing an overdemanded edge). By Lemma 16, the total overdemand is reduced by at least one at one at each new level in the recursion tree. We also note that the number of subpath constraints is reduced by one at each new level. It follows that the maximum depth of recursion is at most $\min(D_o(G, \mathcal{R}), \ell)$. The non-recursive work can be made to take $O(|E|)$ time by precomputing the set of edges in $R_i \cap R_j$ for each directly compatible pair (R_i, R_j) as well as the inverse relationship, namely the set of directly compatible pairs (R_i, R_j) whose intersection contains a given edge. We can also efficiently represent constraint subpath direct compatibility using a separate graph data structure G^c, where each node represents a constraint subpath and edges

indicate direct compatibility. Then, maintaining the current "unioned" constraint subpaths along a given recursion path corresponds to maintaining a set of node-disjoint paths in this graph. These paths can be represented efficiently by tracking which subpaths are tails/heads of current paths and only permitting unions of a current head with a current tail (initially each node is in a separate path and is both a tail and a head). This can be checked and maintained in $O(1)$ time per directly compatible pair considered. When a directly compatible pair (R_i, R_j) containing e is found to union, we decrement the overdemand of all edges in $R_i \cap R_j$ by one; this takes $O(|E|)$ time. The initial work of building the constraint graph G^c can be done by checking each pair of subpath constraints for compatibility ($O(\ell^2|E|)$ time), and then finding the transitive reduction of this relation ($O(\ell^3)$ time using Aho's algorithm with standard matrix multiplication). Then for each edge $e \in G$, we can build a list of directly compatible constraint pairs that contain e in their intersection. ◀

3.2 A heuristic algorithm for MFDSC

In practice, we can run an MFD heuristic algorithm to determine a solution to the FD instance found via the reduction in the previous section. We use greedy-width, first proposed in [27], which greedily chooses the heaviest ("widest") paths in order to decompose the flow. As G' is a DAG, a greedy-width path can found in $O(|V| + |E| + \ell)$ time, by standard dynamic programming. In [27] it is shown that at most $|E| - |V| + 2$ greedy-width paths can be found, so the total time to find an FD solution is $O(|E|(|V| + |E| + \ell))$. Translating the FD solution back to the original graph (following Lemma 8) yields a path decomposition for the FDSC problem. However, in applications, we are often interested in finding solution to the MFDSC problem, i.e. finding a solution with the minimum number of paths. The introduction of bridge edges in the reduction described above may lead to more paths being required to decompose the reduced FD instance than the original FDSC instance. This is because we now must find paths through bridge edges, as well as in the original flow network. For this reason, we apply a *bridge reweighting* heuristic before decomposing the network in order to reduce the number of paths. For some arbitrary ordering of the bridge edges, we do the following:

1. For each bridge edge, find the minimum flow $f_{\min}$ on the edges of its corresponding subpath constraint. Since the FDSC is feasible, $f_{\min} \geq 0$.
2. Subtract $f_{\min}$ from each of the subpath constraint edges, and add $f_{\min}$ to the bridge edge.

Since the bridge edge starts at the first node of the subpath constraint and ends at the last, flow conservation holds and the mapping of the bridge paths back to the original network again provides a solution to the FDSC instance. Figure 2 demonstrates the reduction to an FD instance and the bridge reweighting step on an example FDSC instance.

3.3 An FPT scheme for MFDSC

In this section, we describe an extension of Toboggan [13], an FPT algorithm for decomposing DAG flows, to also handle subpath constraints. Toboggan is able to find a k-path decomposition for a flow network $G = (V, E, f)$, if one exists, in $2^{O(k^2)} \cdot (|V| + \lambda))$ time, where λ is the logarithm of the largest flow value present. To solve MFD, Toboggan just tests increasing k values until a solution is found. We briefly describe Toboggan's approach and then discuss how to modify the algorithm so that it can also check if an FD solution also satisfies subpath constraints.

Toboggan considers the vertices of G in topological order and computes a table T_i for each vertex v_i using dynamic programming. Table entries are of the form (g, L), where g indicates how paths from the previous table T_{i-1} are extended, and L is a linear system

(a) An input FDSC instance with one subpath constraint, modified (for compactness) from the test dataset.

(b) The resulting FD instance that is solved using greedy-width. The pink edge is a *bridge edge* for the corresponding subpath constraint. Weights in parentheses are the weights before bridge reweighting.

Figure 2 Demonstration of reduction and bridge reweighthing procedure used in the heuristic MFDSC algorithm.

indicating how the weights of these paths are constrained to satisfy the flow requirements on all edges encountered so far. This linear system can be written as $\mathbf{Aw} = \mathbf{b}$, where $\mathbf{A}$ is a binary matrix of k columns representing whether each row's edge is covered by each column's path, $\mathbf{w}$ is the length k solution vector, and $\mathbf{b}$ is the flow on the row's edge. Because there are k weights and all coefficients are integers, each linear system can be reduced to k linearly independent rows. As noted in [13], testing an integer linear system L for feasibility and finding a solution can be done in $O(k^{2.5k+o(k)}|L|)$ time, where $|L|$ is shown to be $k^{O(1)}\lambda$.

When the final vertex in the order is reached, these linear systems indicate the path flow constraints on all edges in G, and so, if a particular system is feasible, the corresponding paths and weights provide an FD solution.

To modify Toboggan to also consider the subpath constraints, for the final table $T_{|V|}$, we will add a second linear system to simultaneously satisfy of the form $\mathbf{Aw} \geq \mathbf{b}$, where $\mathbf{A}$ is an $\ell \times k$ binary matrix and $\mathbf{b}^T = (d_1, \ldots, d_\ell)$. Here $A(i,j) \in \{0,1\}$, indicates whether path P_j contains R_i. We give an updated version of a lemma [13, Lemma 5] that bounds the number of distinct linear systems in the final table.

▶ **Lemma 18.** *The final table has at most* $\frac{4^{k^2+\frac{k\ell}{2}}}{k!k^k}$ *distinct linear systems.*

Proof. We follow the proof of [13, Lemma 5]. Since $\mathbf{A}$ is an $\ell \times k$ binary matrix, there are $2^{k\ell}$ possible systems of the second form. We must multiply this by the number of flow matching systems which was bounded ([13, Lemma 5]) by $\frac{4^{k^2}}{k!k^k}$. So, the total number of possible combined linear systems is $2^{k\ell}\frac{4^{k^2}}{k!k^k} = \frac{4^{k^2+\frac{k\ell}{2}}}{k!k^k}$. ◀

▶ **Theorem 19.** *Let $(G, \mathcal{R})$ be an FDSC instance with $|\mathcal{R}| = \ell$ and λ is the logarithm of the largest flow value in the input. Modifying the Toboggan algorithm as described provides an FPT algorithm for MFDSC with running time $2^{O(k^2)}|V| + 2^{O(k^2+k\ell)}(k+\ell)^{O(1)}\lambda$.*

Proof. Kloster et al. prove ([13, Lemma 4]), that in any table T_i, the number of distinct g values present is at most $\sqrt{k}(0.649k)^k$. This implies (following [13, Theorem 7]) that there are at most

$$\frac{4^{k^2+\frac{k\ell}{2}}}{k!k^k} \cdot \sqrt{k}(0.649k)^k = \sqrt{k}\frac{4^{k^2+\frac{k\ell}{2}}0.649^k}{k!}$$

final linear systems L to check for integer solutions. The encoding size of a linear system L is now bounded by $(k+\ell)^{O(1)}\lambda$, where λ is the logarithm of the largest flow value in the input. Checking feasibility and finding a solution for L can now be done in $O(k^{2.5k+o(k)}(k+\ell)^{O(1)}\lambda)$

time, so the total time needed to check all such linear systems is at most

$$\sqrt{k}\frac{4^{k^2+\frac{k\ell}{2}}0.649^k}{k!}\cdot O(k^{2.5k+o(k)}(k+\ell)^{O(1)}\lambda) \leq O(4^{k(k+\frac{\ell}{2})}k^{1.5k}1.765^k k^{o(k)}(k+\ell)^{O(1)}\lambda),$$

using the fact that $\frac{k^k}{k!} \leq e^k$. The total running time of the algorithm becomes $2^{O(k^2)}|V| + 2^{O(k^2+k\ell)}(k+\ell)^{O(1)}\lambda$. ◀

4 Experiments

The algorithms described in Section 3 were implemented in Python[1] in a package called *Coaster*.[2] We refer to the algorithm of Section 3.2 as *heuristic MFDSC* and Section 3.3 as *FPT MFDSC*. Experiments were performed on a high performance research cluster, where each run was executed on a single Intel Xeon Ivy Bridge E (3.4 Ghz) or similar CPU. For all runs, a memory limit of 20 Gb was set and instances were timed out if they ran longer than 30 seconds. For fairness of comparison, we report only on graph instances that ran to completion for all algorithm and parameter combinations, unless otherwise mentioned.

Datasets. As in previous studies on flow decomposition methods for RNA-Seq assembly [13, 22, 28], we use a simulated RNA-Seq dataset from [22] where each instance is a flow network generated by simulating RNA transcripts and their abundances with Flux-Simulator [10]. The original dataset includes human, mouse, and zebrafish genes, but we restrict our attention to instances in the human dataset, which contains 100 independently generated transcriptomes. As in [13, 28], we use only instances with at least two ground truth paths (since a single ground truth path is trivial to decompose). We also restrict the dataset to instances with 10 ground truth paths or fewer, yielding a total of 528,544 instances. Because the transcripts and abundances are known, we have ground truth paths and weights for each splice graph instance. We measure accuracy as the proportion of instances for which the algorithm returns the ground truth set of paths and weights exactly.

Simulating subpath constraints. In order to simulate subpath constraints, we take subpaths of the ground truth paths according to two parameters: the number of subpaths ℓ, and a fixed length for all subpaths $|R|$. As noted in [13, Lemma 8], we can simplify the graph by bypassing any vertex with out-degree or in-degree equal one. We set $|R|$ as the length of subpaths in this contracted graph. To generate subpath constraints that are consistent across experiments, we fix an arbitrary ordering for the ground truth paths for each instance, and take the first $|R|$ edges of the first ℓ (contracted) paths as the subpaths. We note that the method of generating subpath constraints described here does not yield any overdemanded edges.

Accuracy results. To study the effect of the subpath constraints on the accuracy of the RNA-Seq assembly, we vary ℓ and $|R|$ independently, letting $\ell \in [0, 4]$ and $|R| \in \{3, 4\}$. Because instances become more difficult to solve correctly as the number of ground truth paths increases, we separate results by the number of ground truth paths, which we denote by k. Accuracy results for both algorithms are reported in Table 1. For each k value, we

[1] Based on the codebase for Toboggan [12].
[2] Available at `https://github.com/msu-alglab/coaster`.

also report the percentage of instances that completed for all parameter combinations tested. As already shown by Kloster et al. [13], the MFD solutions found by Coaster for $\ell = 0$ (for them, Toboggan) do correspond to the the ground truth paths and weights most of the time. However, for larger k values, we can see that FPT MFD solutions (without subpath constraints) do not necessarily recover the correct set of paths and weights. For $k = 7$, for example, only 81% of the optimal decompositions produced by Coaster are the ground truth decomposition that we are seeking. Similarly, we see that FPT MFDSC solutions tend to be correct, with accuracy decreasing as k increased. However, FPT MFDSC has higher accuracy for all parameter combinations than FPT MFD at the same k value. For $k = 7$, when we add four subpath constraints of length four each, the ground truth decomposition is found 91% of the time, a 13% increase over FPT MFD.

When $\ell = 0$, our heuristic MFDSC algorithm is equivalent to the often-used greedy-width heuristic for MFD; our results show that adding subpath constraints to greedy-width increases its accuracy considerably for larger k values, for example, by 30% when $k = 7$. The increased accuracy of heuristic MFDSC is also good news for the use of MFDSC in practical methods, since heuristic MFD methods are already commonly used in RNA-Seq tools. In fact, the inclusion of many long subpath constraints makes heuristic MFDSC more accurate than FPT MFD for k values up 5, which account for 95.6% of the full dataset studied (all $k \geq 2$).

Part of the success of the heuristic MFDSC can be attributed to the fact that it finds optimal solutions in most cases. Without subpath constraints, heuristic MFDSC (i.e. greedy-width MFD) finds an optimal solution in 98.0% and the ground truth solution in 95.3% of instances in our dataset. With two subpath constraints of length 4, that increases to 99.0% and 98.1%, respectively. (For $\ell > 2$, small k values are excluded, so results are not comparable with $\ell = 0$ experiments.)

With and without subpath constraints, the vast majority of incorrectly predicted path decompositions are due to the algorithm returning an optimal decomposition of the same size as the ground truth one, but different from it, rather than a too-small optimal decomposition. As found in [13], in nearly all instances, the ground truth path decomposition is also an optimal decomposition. (They find that 0.043% of instances of all ground truth k that ran to completion in 50 seconds had non-optimal ground truth decompositions; we find that 0.100% of instances that completed in 30 seconds for all parameter combinations and had ground truth k less than 9 had non-optimal ground truth decompositions.) However, most instances are solved correctly, so it could be the case that the few instances that are not solved correctly are those that had non-optimal ground truths. This tends not to be the case. Overall, only 0.027% of instances for which the FPT for MFD yields incorrect solutions have non-optimal ground truth path decompositions. This is dominated by the $k = 2$ instances, however, for which no instance had a non-optimal ground truth; for $k = 3$ through $k = 8$, between 0.1% and 0.3% of instances that were predicted incorrectly had non-optimal ground truths. With many and longer subpath constraints ($|R| = 4$ and $\ell = 4$), it is still only a very small number – 0.052% – of incorrect solutions that have non-optimal ground truth path decompositions. Thus, this implies that the addition of subpath constraints restricts the solution space, allowing the algorithm to return the correct one more frequently and explaining the increase in accuracy when they are included.

Effect of the bridge reweighting. To confirm the effectiveness of the bridge reweighting heuristic for MFDSC, as opposed to simply using a path decomposition found by the method of Lemma 8, we measured the accuracy of the FDSC algorithm without bridge reweighthing on the same dataset studied above. In that case, the addition of subpath constraints in our

Table 1 Accuracy results using heuristic MFDSC (odd rows) and FPT MFDSC (even rows). For $k = 2$ through $k = 8$, we only report on instances that completed for every parameter combination; the "pc" column reports the percentage of instances that completed for all runs for each k value. For $k = 9$ and $k = 10$, less than 40% and 1% respectively of instances completed for the FPT runs, so we include them only for heuristic MFDSC, which ran to completion for all instances. The italicized values are the ones reported in [13, Figure 3], with some slight differences due to the fact that we restrict to the human dataset (they studied two additional datasets) and timeout differences.

				$\lvert R\rvert = 3$				$\lvert R\rvert = 4$			
k	n	pc	$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
2	291734	100%	0.992	0.999	0.999			1.000	1.000		
			0.991	0.999	0.999			1.000	1.000		
3	130867	100%	0.961	0.977	0.983	0.986		0.985	0.993	0.994	
			0.969	0.983	0.990	0.994		0.986	0.996	0.998	
4	58167	100%	0.901	0.926	0.941	0.948	0.958	0.942	0.964	0.974	0.979
			0.934	0.952	0.964	0.974	0.983	0.958	0.976	0.987	0.995
5	25933	100%	0.822	0.853	0.873	0.887	0.9	0.876	0.911	0.93	0.944
			0.892	0.913	0.928	0.940	0.953	0.922	0.944	0.962	0.976
6	11774	99.6%	0.727	0.763	0.784	0.805	0.816	0.787	0.831	0.862	0.883
			0.849	0.870	0.885	0.898	0.911	0.881	0.906	0.928	0.944
7	5095	94.6%	0.617	0.659	0.692	0.706	0.729	0.681	0.738	0.775	0.802
			0.810	0.835	0.855	0.871	0.883	0.845	0.872	0.894	0.912
8	2109	83.7%	0.495	0.523	0.558	0.589	0.611	0.545	0.607	0.664	0.702
			0.787	0.808	0.822	0.833	0.845	0.819	0.840	0.863	0.884
9	1323	100%	0.388	0.423	0.452	0.485	0.512	0.447	0.497	0.555	0.608
10	699	100%	0.310	0.333	0.349	0.367	0.393	0.351	0.383	0.418	0.454

experiments reduces the accuracy of the path decompositions returned, as shown in Table 2. Bridge reweighting allows the maximum flow that can cover a subpath constraint to do so, without introducing extra weight-one paths.

Performance results. We measured runtime and peak memory use of the implementation for both algorithms using all instances $k = 2$ through $k = 10$. For heuristic MFDSC on all instances, even those that timed out during FPT runs, the average, minimum, and maximum runtimes in seconds were 0.0059, 0.00096, and 0.977. For FPT instances that completed, they were 0.076, 0.001, and 30 (the timeout limit set for the experiments – instances would take longer if allowed). As mentioned in the Table 1 caption, for FPT MFDSC, less than half and almost no instances with $k = 9$ and $k = 10$ completed in 30 seconds respectively,

Table 2 Accuracy values for FD heuristic without bridge reweighthing. If all bridges are kept at weight one, subpath constraints reduce the accuracy of the path decomposition, though less if they are longer.

	$\lvert R\rvert = 3$				$\lvert R\rvert = 4$			
$\ell = 0$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$
0.980	0.940	0.816	0.588	0.348	0.958	0.919	0.798	0.637

indicating as expected that there is a steep increase in runtime as k gets large. However, for the heuristic MFDSC, no instance took longer than one second, even those with $k = 10$, showing that the heuristic MFDSC algorithm can be practical in applications. We measured the peak memory usage for batches of 2000 instances (batched according to the original order of instances in the dataset). Thus, the memory use for all instances is included, including those that timed out, and minimum and average measurements are over the batches, not individual instances. For heuristic MFDSC, the average, minimum, and maximum peak memory usage in MB was 52, 36, and 60. For FPT MFDSC, they were 77, 51, and 811.

5 Discussion

In this work, we initiate the formal study of the MFDSC problem, which is used as a model in applications such as RNA sequencing and viral quasispecies assembly. We give both a heuristic algorithm, based on a novel reduction to flow decomposition, and an FPT algorithm, which extends the FPT MFD algorithm of Kloster et al. [13]. Through experiments on a previously-studied simulated transcriptomics dataset, we verify the base assumption underlying the use of MFDSC in practical RNA-Seq tools: that the minimum-size path decomposition should correspond to the ground truth set of paths and weights. Additionally, we show that the use of subpath constraints increases accuracy when compared to MFD without subpath constraints. We also find that our heuristic algorithm is practical, completing in less than 1 second for all instances studied, and achieves accuracy levels near those of FPT MFDSC. This is an encouraging result, because while RNA sequencing data tends toward very small ground truth path sets, other multiassembly problems such as viral quasispecies assembly may not – for example, some benchmarking datasets of [3] contain 10 and 15 strains, meaning that MFDSC (or even MFD without subpath constraints) would be intractable without a heuristic.

The research presented here suggests a number of future directions. One is characterizing the complexity of determining whether an FDSC instance is feasible. The heuristic MFDSC algorithm we give begins with this step, but takes time exponential in the total overdemand and the number of subpath constraints if any edge is overdemanded. Another is to develop MFDSC algorithms for graphs containing cycles. Though splice graphs for RNA assembly are usually DAGs, graphs for de novo assembly of viral or other genomes would likely contain cycles due to repeated sequences. Finally, we find that as the size of ground truth gets large, accuracy decreases because there are multiple optimal solutions, even with the maximum length and number of subpath constraints that we tested. To increase accuracy, either more subpath constraints are needed (which may be possible, depending on the domain), or additional optimality criteria could be used.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

2 Jasmijn A. Baaijens, Bastiaan Van der Roest, Johannes Köster, Leen Stougie, and Alexander Schönhuth. Full-length de novo viral quasispecies assembly through variation graph construction. *Bioinform.*, 35(24):5086–5094, 2019. doi:10.1093/bioinformatics/btz443.

3 Jasmijn A Baaijens, Amal Zine El Aabidine, Eric Rivals, and Alexander Schönhuth. De novo assembly of viral quasispecies using overlap graphs. *Genome research*, 27(5):835–848, 2017.

4 Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *International Conference on Research in Computational Molecular Biology*, pages 221–222. Springer, 2020.

5 Georg Baier, Ekkehard Köhler, and Martin Skutella. On the k-splittable flow problem. In *European Symposium on Algorithms*, pages 101–113. Springer, 2002.

6 Georg Baier, Ekkehard Köhler, and Martin Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

7 Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs Theory, Algorithms and Applications*. Springer-Verlag, Berlin, 1st edition, 2000.

8 Ergude Bao, Tao Jiang, and Thomas Girke. Branch: boosting RNA-Seq assemblies with partial or related genomic sequences. *Bioinformatics*, 29(10):1250–1259, 2013.

9 Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, 2014. doi:10.1093/bioinformatics/btu317.

10 Thasso Griebel, Benedikt Zacher, Paolo Ribeca, Emanuele Raineri, Vincent Lacroix, Roderic Guigó, and Michael Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic acids research*, 40(20):10073–10083, 2012.

11 Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *2012 Proceedings IEEE INFOCOM*, pages 828–836. IEEE, 2012.

12 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. Toboggan: Version 1.0, June 2017. doi:10.5281/zenodo.821634.

13 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical FPT algorithm for flow decomposition and transcript assembly. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, 2018.

14 Anna Kuosmanen, Tuukka Norri, and Veli Mäkinen. Evaluating approaches to find exon chains based on long reads. *Briefings in bioinformatics*, 19(3):404–414, 2018.

15 Anna Kuosmanen, Ahmed Sobih, Romeo Rizzi, Veli Mäkinen, and Alexandru I. Tomescu. On using longer RNA-Seq reads to improve transcript prediction accuracy. In James P. Gilbert, Haim Azhari, Hesham H. Ali, Carla Quintão, Jan Sliwa, Carolina Ruiz, Ana L. N. Fred, and Hugo Gamboa, editors, *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - Volume 3: BIOINFORMATICS, Rome, Italy, February 21-23, 2016*, pages 272–277. SciTePress, 2016. doi:10.5220/0005819702720277.

16 Wei Li, Jianxing Feng, and Tao Jiang. Isolasso: A LASSO regression approach to RNA-Seq based transcriptome assembly. *Journal of Computational Biology*, 18(11):1693–1707, 2011. doi:10.1089/cmb.2011.0171.

17 Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity balancing path flow decomposition and routing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.

18 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. Stringtie enables improved reconstruction of a transcriptome from RNA-Seq reads. *Nature Biotechnology*, 33(3):290–295, 2015.

19 Krzysztof Pieńkosz and Kamil Kołtyś. Integral flow decomposition with minimum longest path length. *European Journal of Operational Research*, 247(2):414–420, 2015.

20 Romeo Rizzi, Alexandru I. Tomescu, and Veli Mäkinen. On the complexity of minimum path cover with subpath constraints for multi-assembly. *BMC Bioinform.*, 15(S-9):S5, 2014. doi:10.1186/1471-2105-15-S9-S5.

21 Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature biotechnology*, 35(12):1167–1169, 2017.

22 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2017.

23 Tamara Steijger, Josep F Abril, Par G Engstrom, Felix Kokocinski, The RGASP Consortium, Tim J Hubbard, Roderic Guigo, Jennifer Harrow, and Paul Bertone. Assessment of transcript

reconstruction methods for RNA-Seq. *Nat Meth*, 10(12):1177–1184, December 2013. doi:10.1038/nmeth.2714.

24 Vorapong Suppakitpaisarn. An approximation algorithm for multiroute flow decomposition. *Electronic Notes in Discrete Mathematics*, 52:367–374, 2016. INOC 2015 – 7th International Network Optimization Conference. doi:10.1016/j.endm.2016.03.048.

25 Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14(S-5):S15, 2013. Proceedings paper from RECOMB-seq: Third Annual RECOMB Satellite Workshop on Massively Parallel Sequencing Beijing, China. 11-12 April 2013. doi:10.1186/1471-2105-14-S5-S15.

26 Cole Trapnell, B.A. Williams, G. Pertea, Ali Mortazavi, G. Kwan, M.J. van Baren, S.L. Salzberg, B.J. Wold, and L. Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28:511–515, 2010.

27 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008.

28 Lucia Williams, Gill Reynolds, and Brendan Mumey. RNA transcript assembly using inexact flows. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914, 2019.

29 Ting Yu, Zengchao Mu, Zhaoyuan Fang, Xiaoping Liu, Xin Gao, and Juntao Liu. Transborrow: genome-guided transcriptome assembly by borrowing assemblies from different assemblers. *Genome Research*, 30(8):1181–1190, 2020. doi:10.1101/gr.257766.119.

30 Osvaldo Zagordi, Arnab Bhattacharya, Nicholas Eriksson, and Niko Beerenwinkel. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. *BMC Bioinformatics*, 12(1):119+, 2011.

Conflict Resolution Algorithms for Deep Coalescence Phylogenetic Networks

Marcin Wawerka
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Dawid Dąbkowski
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Natalia Rutecka
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Agnieszka Mykowiecka
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Paweł Górecki
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

We address the problem of inferring an optimal tree displayed by a network, given a gene tree G and a tree-child network N, under the deep coalescence cost. We propose an $O(|G||N|)$-time dynamic programming algorithm (DP) to compute a lower bound of the optimal displayed tree cost, where $|G|$ and $|N|$ are the sizes of G and N, respectively. This algorithm has the ability to state whether the cost is exact or is a lower bound. In addition, our algorithm provides a set of reticulation edges that correspond to the obtained cost. If the cost is exact, the set induces an optimal displayed tree that yields the cost. If the cost is a lower bound, the set contains pairs of conflicting edges, that is, edges sharing a reticulation node. Next, we show a conflict resolution algorithm that requires $2^{r+1} - 1$ invocations of DP in the worst case, where r is a number of reticulations. We propose a similar $O(2^k|G||N|)$-time algorithm for level-k networks and a branch and bound solution to compute lower and upper bounds of optimal costs. We also show how our algorithms can be extended to a broader class of phylogenetic networks. Despite their exponential complexity in the worst case, our solutions perform significantly well on empirical and simulated datasets, thanks to the strategy of resolving internal dissimilarities between gene trees and networks. In particular, experiments on simulated data indicate that the runtime of our solution is $\Theta(2^{0.543k}|G||N|)$ on average. Therefore, our solution is an efficient alternative to enumeration strategies commonly proposed in the literature and enables analyses of complex networks with dozens of reticulations.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Applied computing → Computational genomics

Keywords and phrases Phylogenetic Network, Gene Tree, Species Tree, Deep Coalescence, Reticulation, Optimal Displayed Tree

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.17

Funding The support was provided by National Science Centre grant #2019/33/B/ST6/00737.

1 Introduction

Evolutionary networks are mathematical models of evolutionary processes with reticulate events such as hybridization, recombination, or horizontal gene transfer [1, 24]. Hybridization is a common phenomenon in plants and is often used in agriculture to create new breeds [18]. Recombination and reassortment are two shuffling processes in which variants of genetic material are created from pairs of highly similar DNA sequences. For example, many viruses have segmented genomes, including influenza viruses and rotaviruses [32], while horizontal

gene transfer is common in bacteria [3]. In the last decades, mathematical and computational properties of phylogenetic networks have been intensively studied (see books [20, 24]). One of the most classic notions is a *tree displayed by a network*, obtained from a network by removing a set of *reticulation edges*. Displayed trees may represent an evolutionary history of a gene family if the incongruence is due to a reticulation event [24]. Alternative approaches include embedding a gene tree into a displayed tree [27, 31, 43] or using a parental species tree as a generalization of a displayed tree [25, 43, 49].

The pioneering work by Maddison [28] introduced the deep coalescence (DC) cost, which measures the *extra* gene lineages of a gene tree when embedded into a species tree. When a gene is embedded into its species tree, each edge of the species contains several mapped gene lineages. In the perfect situation, when both trees have the same topology, this number is one. Therefore, the cost is defined by the number of *extra* gene lineages required to embed a gene tree into a species tree. DC and general coalescent-based methods are popular in classical problems of computational biology, e.g. estimation of species trees [33, 41, 48], tree reconciliation [15, 10, 44, 48], or gene tree error correction [8].

There are two main general approaches to embed a gene tree into a network using the parsimony principle: (1) choosing the tree displayed by the network with the lowest cost, i.e. solving the *optimal displayed tree (ODT)* problem, in which a reticulation node can be reached only from one fixed parent, or (2) *a direct tree-network* embedding, without the above constraint. These approaches are present in relevant articles concerning inferences of networks under Robinson-Foulds (RF) embedding cost [31], the duplication-loss cost [43], the deep coalescence cost [27], including the general parsimony framework using the concept of parental species trees [25]. Alternative studies are based on minimizing deep coalescence criterion [47] or probabilistic models on coalescent histories [46]. Model-based approaches are usually computationally demanding since they often require enumeration of all possible coalescence histories [42, 46]. Finally, perhaps one of the most prominent applications of the above methods is the problem of network inference (e.g., [24, 25, 31, 43, 45] and tools e.g., [42]).

From the theoretical point of view, ODT under DC corresponds to NP-hard problems: (1) best switching (i.e., choosing the set of reticulation edges) for the duplication-loss model [43], and (2) the computation of RF-embedding cost [31]. In [43] the problem is solved in $O(|N| + p2^k|G|)$ time, where G is a gene tree and p is the number of biconnected components in a level-k network N. [31] proposed an $O(2^r|N|)$-time optimized algorithm to compute RF-embedding cost. Another relevant contribution is from [27] with an $O(4^k|G||N|^2)$-time tree vs. level-k network reconciliation algorithm under DC events. However, the latter cannot be directly compared to ours since we solve a different problem. In all of the above contributions, the complexity coming from 2^r (or 2^k) is reached due to exhaustive enumeration strategies. In this article, we show how to avoid such strategies by proposing an efficient in practice method to infer optimal displayed trees despite the theoretical intractability of ODT in general.

Our contribution: We address the problem of inference of an optimal tree displayed by a network (ODT), given a gene tree G and a network N under the deep coalescence cost (DC). We propose a novel approach in which we define scenarios for embedding G into N using sets of reticulation edges from N, with a property that the score of a scenario approximates the displayed tree cost. In particular, we prove that the score of a scenario is a lower bound of the cost of the optimal displayed tree. In a particular case, when a scenario induces a non-conflicting set of reticulation edges, we provide the correspondence between a score of

this scenario and a cost of a displayed tree. Next, we propose an $O(|G||N|)$ time dynamic programming (DP) algorithm to compute an optimal scenario. We show that an optimal scenario with no conflicts corresponds to a solution of ODT. Based on DP, we design a recursive algorithm to ODT by resolving conflicts in sets of reticulation edges. This algorithm has exponential time complexity $O(2^r|G||N|)$, where r is the number of reticulation nodes in N. We propose a similar $O(2^k|G||N|)$-time algorithm for level-k networks. We also show how the algorithms can be extended to a broader class of phylogenetic networks. Finally, we show experimental studies on random, simulated, and empirical datasets. We show that our algorithm has significantly improved runtime on simulated datasets by reducing the exponent from r to $0.543r$ on average.

2 Definitions

A *network* on a set of species X is a directed acyclic graph $N = (V(N), E(N))$ with a single root such that: (1) its leaves, i.e., nodes of indegree 1 and outdegree 0, are labeled by the species from X, and (2) there is a path from the root to any other vertex. A network is *binary* if its leaves, root, and the remaining nodes have degrees 1, 2 and 3, respectively. A node is called a *reticulation* if it has indegree two and outdegree one, and a *tree node* if it has indegree at most one and outdegree two. A network is *semi-binary*, if additionally, it may contain *semi-binary nodes* of indegree at most one and outdegree one, which includes the root having exactly one child. We can *contract* a semi-binary node v of indegree one as follows: (1) remove v, (2) remove both edges incident with v, and (3) insert a new edge connecting the unique parent of v with the only child of v. Similarly, if v has indegree zero we remove v, and the child of v becomes a new root.

If $\langle v, w\rangle \in E(N)$, then v is a parent of w and w is a child of v, denoted $w.\mathsf{parent} = v$ if w is a non-root tree node or a leaf. We write $v.\mathsf{sibling} = w$ if $v \neq w$ have the same parent. We write $v \succeq w$ if there is a directed path from v to w, and $v \succ w$ if $v \succeq w$ and $v \neq w$. The set of all leaves in a network is denoted $L(N)$, by $R(N) \subset V(N)$ we denote the set of reticulation nodes in N, and by $E_R(N) \subset E(N)$ we denote the set of all reticulation edges in N, that is, edges $\langle v, r\rangle \in E(N)$ with $r \in R(N)$. We say that a reticulation edge e is a *sibling* of a reticulation edge e' if they share the same bottom reticulation node. By $\mathrm{outdeg}_N(v)$ we denote the outdegree of v in N.

A *phylogenetic network* is a binary network on X in which the leaves are labeled one-to-one with the species from X[1]. A *species tree* is a phylogenetic network without reticulation nodes. Similarly, a *gene tree*, or in short *a tree*, is a binary network without reticulation nodes. Note that the leaf labeling in a gene tree does not have to be one-to-one.

From now on, all phylogenetic networks are *tree-child* [4, 5, 6, 35] (except for Section 5.4 and 6.3), i.e., each non-leaf node has a child that is either a tree node or a leaf.

2.1 Deep Coalescence Cost: embedding a tree into a (displayed) tree

Given a gene tree G and a species tree S on X, the *lca-mapping* $\mathsf{M}\colon V(G) \to V(S)$ is defined as follows: (1) if g is a leaf labeled $x \in X$ then $\mathsf{M}(g)$ is the unique leaf labeled x in S, and (2) if g has two children g' and g'', then $\mathsf{M}(g)$ is the lowest common ancestor of $\mathsf{M}(g')$ and $\mathsf{M}(g'')$ in S. Embedding G into S is performed by mapping each edge $\langle v, w\rangle \in E(G)$ to a

[1] Note that in a binary network two leaves can be labeled by the same label, which is not allowed in a phylogenetic network.

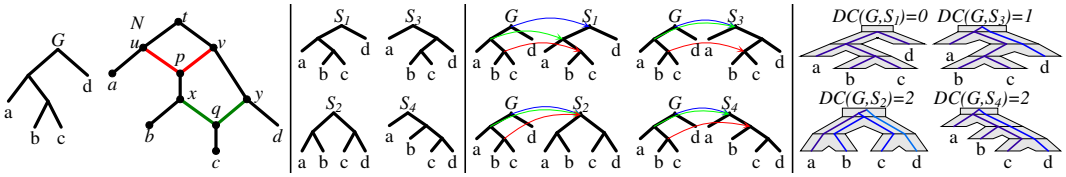


Figure 1 From the left: a gene tree G and a phylogenetic network N with two reticulations, four trees displayed by N, lca-mappings between G and S_i's, visualization of embeddings of G into S_i's. Here, S_1 is the optimal tree displayed by N with the DC cost of 0.

path connecting $\mathsf{M}(v)$ and $\mathsf{M}(w)$ in S. We say that the gene edge *visits* edges from that path. Let $||v, w||$ denote the number of edges on the path connecting v and w. Then, the visited edges contribute to the deep coalescence cost, denoted $\mathsf{DC}(G, S)$, as follows:

$$\mathsf{DC}(G, S) = \sum_{\langle v,w\rangle \in E(G)} (||\mathsf{M}(v), \mathsf{M}(w)|| - 1). \tag{1}$$

A set $Y \subseteq E_R(N)$ is called *perfect* if, for each $r \in R(N)$, Y contains exactly one edge whose bottom node is r. Given a perfect Y, the graph denoted N/Y, obtained from N by removing all edges from $E_R(N) \setminus Y$ is a semi-binary tree, i.e., semi-binary network with no reticulations. Further, contracting all semi-binary nodes yields a species tree on X, denoted N_Y. We say that a species tree T is *displayed* by a phylogenetic network N, if $N_Y = T$ for some perfect Y. We also say that the perfect set Y is induced by a tree T displayed by N if $N_Y = T$.

We now define the *Optimal Displayed Tree* problem (ODT) in the parsimony framework:

▶ **Problem 1** (ODT). *Given a tree G and a phylogenetic network N. Find an optimal tree S^* displayed by N that minimizes $\mathsf{DC}(G, S)$ in the set of all trees S displayed by N.*

The *cost* of an optimal displayed tree, we denote $\mathsf{DC}(G, N)$. While the complexity of ODT remains unknown for the class of tree-child networks, we claim that the problem is NP-hard in a general class of networks. The proof is similar to the NP-hardness proof of the best switching problem from [43]. See also [31] for the related problem of RF-embedding. Fig. 1 depicts an example of DC costs.

2.2 Scenarios between gene trees and phylogenetic networks

In the previous Section, we showed how a gene tree is embedded into a species tree. Here, we propose to embed a gene tree into a phylogenetic network using a more general approach than embedding through a displayed tree. We start with the notion of *unfolded network* (see also [22, 23]), then we define *scenarios* between gene trees and unfolded networks.

For a phylogenetic network N on X with k reticulations, the *unfolded network* $\hat{N}$ is the tree N_k obtained from N by a sequence of k unfolding operations defined on pairs (N_i, σ_i), such that N_i is a semi-binary network on X and $\sigma_i \colon V(N_i) \to V(N)$ defines the origin of a node from N_i. Let (N_0, σ_0) be a pair such that $N_0 = N$ and $\sigma_0(v) = v$ for each $v \in V(N)$. Then, for a sequence of all reticulation nodes $r_1, r_2, \ldots, r_k$ from N in a reversed topological order, (N_i, σ_i) is obtained from (N_{i-1}, σ_{i-1}) by unfolding the reticulation r_i as follows:

- Let S_i be a copy of the subtree of N_{i-1} rooted at r_i.
- $V(N_i) := V(N_{i-1}) \cup V(S_i)$ and $E(N_i) := (E(N_{i-1}) \setminus \{\langle p, r_i\rangle\}) \cup E(S_i) \cup \{\langle p, r_i'\rangle\}$, where p is an arbitrary parent of r_i and r_i' is the root of S_i.
- $\sigma_i(v)$ is $\sigma_{i-1}(v)$ if $v \in V(N_{i-1})$; otherwise, it is $\sigma_{i-1}(t)$, if v is a copy of t from N_{i-1}.

Informally, for each reticulation node, we copy its subtree, detach the original subtree from one parent, and attach the copy to the same parent, without changing the labels. To avoid using k directly, we set σ to be σ_k.[2] Fig. 2 depicts an unfolded network.

▶ **Lemma 2** (Correctness of unfolding). *The unfolded network $\hat{N}$ of N is a semi-binary tree.*

Let a *leaf-root* path be a directed path connecting a leaf with the root in a network.

▶ **Theorem 3** (Unfolding Soundness). *There is a one-to-one correspondence between leaf-root paths in N and leaf-root paths in $\hat{N}$.*

It follows from Thm. 3 that N and $\hat{N}$ have the same structure of leaf-root paths. *A scenario* for G and N is a function $\xi\colon L(G) \to L(\hat{N})$ that preserves the leaf labeling: for every $g \in L(G)$, the labels of g and $\xi(g)$ are equal. A scenario ξ can be extended to the lca-mapping $\mathsf{M}_\xi\colon V(G) \to V(\hat{N})$ such that for $g \in V(G)$, $\mathsf{M}_\xi(g)$ is the lowest node v in $\hat{N}$ such that $\xi(g') \preceq v$, for each leaf $g' \preceq g$. Note that $M_\xi(g)$ is either a leaf or a tree node with two children.

2.3 Deep Coalescence Score of Scenarios

Having the lca-mapping determined by a scenario, we are ready to define the deep coalescence *score*, denoted $\tilde{\mathsf{DC}}$, to approximate deep coalescence events induced by scenarios in phylogenetic networks. Our goal is to deduce properties allowing us to approximate the DC cost to solve ODT. In particular, our approach differs from the approaches from [25, 27, 43], e.g., in the way how a cost of a path is defined, although the general concept of mapping a gene tree into a network is analogous.

For a scenario ξ for G and N, we say that $\langle v, w\rangle \in E(G)$ *visits* $\langle a, b\rangle \in E(\hat{N})$ if $\mathsf{M}_\xi(v) \succeq a \succ b \succeq \mathsf{M}_\xi(w)$. Then, $\langle a, b\rangle$ has exactly one of the following *types*.

- Type I: $\mathsf{M}_\xi(v) = a$, i.e., it is the first edge.
- Type II: $\mathsf{M}_\xi(v) \succ a$, $\mathrm{outdeg}_{\hat{N}}(a) = 2$ and $\sigma(b.\mathsf{sibling}) \notin R(N)$;
- Type III: $\mathsf{M}_\xi(v) \succ a$, $\mathrm{outdeg}_{\hat{N}}(a) = 2$ and $\sigma(b.\mathsf{sibling}) \in R(N)$; we say that ξ *bypasses* the reticulation edge $\sigma(\langle a, b.\mathsf{sibling}\rangle)$
- Type IV: $\mathrm{outdeg}_{\hat{N}}(a) = 1$ (only if $\sigma(a) \in R(N)$).

In the above definition, type (I) is only for the first (i.e., the closest to the root) edge visited by a given edge from G, while for the remaining visited edges from $\hat{N}$ an edge has: Type (II) if the sibling of its bottom node is a tree node, Type (III) if the sibling of its bottom node is a reticulation, and Type (IV) if the top node of the edge is a reticulation.

By $\kappa_\xi(v, w)$ we denote the set of all edges of Type I or II visited by $\langle v, w\rangle$. Then, the *deep coalescence score* for G, N and a scenario ξ is

$$\tilde{\mathsf{DC}}(G, N, \xi) = \sum_{\langle v,w\rangle \in E(G)} (|\kappa_\xi(v, w)| - 1). \tag{2}$$

Examples of scenarios and $\tilde{\mathsf{DC}}$ scores are depicted in Fig. 2. Finally, we can define *Optimal Scenario Inference* problem, DC-MinRec.

▶ **Problem 4** (DC-MinRec). *Given a gene tree G and a phylogenetic network N. Find an optimal scenario ξ^* that minimizes $\tilde{\mathsf{DC}}(G, N, \xi)$ in the set of all scenarios ξ for G and N.*

[2] We also use σ with edges, e.g., $\sigma(\langle v, w\rangle)$ denotes $\langle\sigma(v), \sigma(w)\rangle$.

In Section 4, we propose a dynamic programming algorithm that solves DC-MinRec in $O(|G||N|)$ time. Note that the complexity depends on the size of N (not on the potentially exponential size of $\hat{N}$).

In a trivial case, the solution to DC-MinRec is induced by the classical DC cost.

▶ **Lemma 5.** *If N is a phylogenetic network with no reticulation node then there is only one scenario ξ for G and N. Moreover, $\widetilde{\mathsf{DC}}(G, N, \xi) = \mathsf{DC}(G, N)$.*

3 Scores of scenarios vs. costs of displayed trees

In this Section, we present several theoretical results connecting our scoring functions. Note that the notion of a *cost* will be used only with the DC cost defined in (1) for trees and for phylogenetic networks in Problem 1, while for scenarios, we will use the notion of a ($\widetilde{\mathsf{DC}}$) *score*. To establish the correspondence, we first show that each perfect set Y determines a scenario. Recall that N_Y is obtained from N/Y by contracting semi-binary nodes. Let $\hat{N}_Y$ be the graph obtained from $\hat{N}$ by removing all edges e such that $\sigma(e) \in E_R(N) \setminus Y$ and all subtrees whose root is the bottom node of e.

▶ **Lemma 6.** *$\hat{N}_Y$ and N/Y are isomorphic, and the isomorphism is established by $\sigma|_{V(\hat{N}_Y)}$.*

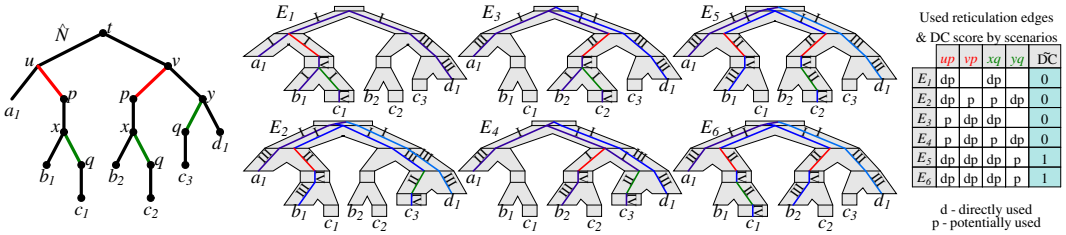


Figure 2 *Left:* The unfolded network $\hat{N}$ of N from Fig. 1 is shown with σ values attached to nodes, where for the leaves, the index is inserted to distinguish leaves with the same labels/mappings. *Middle:* 6 scenarios for $G = ((a, (b, c)), d)$ shown as embeddings of G to $\hat{N}$. Numbers I-IV denote the type of a visited edge. Only E_1 is regular, while $E_1 - E_4$ are optimal. *Right:* DC score and types of used reticulation edges for each scenario ($\Upsilon_{[.]}$).

For a perfect set Y, we define a scenario ξ_Y such that for each gene leaf g labeled x, $\xi_Y(g)$ is the only leaf in $L(\hat{N}_Y) \subseteq L(\hat{N})$ labeled x. Correctness follows from Lemma 6. For example, in Fig. 2, if $Y = \{\langle u, p\rangle, \langle x, q\rangle\}$, then Y is perfect and $N_Y = S_1$ from Fig. 1. Moreover, for $G = ((a, (b, c)), d)$, ξ_Y maps a to a_1, b to b_1 c to c_1 and d to d_1 as depicted in E_1.

We say that $e \in E_R(N)$ is *directly used* by scenario ξ if there is a visited edge e' of Type I or II such that $\sigma(e') = e$. Similarly, we say that reticulation edge e is *potentially used* by ξ if the sibling edge of e is bypassed by ξ. By $\Upsilon_\xi \subseteq E_R(N)$ we denote the set of reticulation edges used directly or potentially by ξ (see Fig. 2).

We say that $Y \subseteq E_R(N)$ has a *conflict* if Y contains two sibling edges. We say that ξ is *regular* if Υ_ξ has no conflict. For instance, Υ_{E_4} for E_4 from Fig. 2 has two possible conflicts in N. Observe that Υ_ξ may not be perfect in general, even if ξ is regular. For instance, if $G = (c, d)$ and ξ maps c to c_3 in the network from Fig. 2, then $\Upsilon_\xi = \{\langle y, q\rangle\}$.

Now, we can state the crucial proposition that establishes a correspondence between regular scenarios and embedding to trees displayed by a network.

▶ **Proposition 7** (Scenario-Displayed Tree Correspondence)**.** *A scenario ξ is regular, if and only if for every perfect set Y such that $\Upsilon_\xi \subseteq Y$, $\widetilde{\mathsf{DC}}(G, N, \xi) = \mathsf{DC}(G, N_Y)$.*

In the following proposition, we show that the cost of a tree displayed by a network using a perfect set is bounded from below by the cost of its corresponding scenario.

▶ **Proposition 8.** *If Y is perfect, then* $\mathsf{DC}(G, N_Y) \geq \tilde{\mathsf{DC}}(G, N, \xi_Y)$.

Finally, we show that the equality between the score and the cost holds only if the induced scenario is regular.

▶ **Proposition 9.** *If Y is perfect, then* $\mathsf{DC}(G, N_Y) = \tilde{\mathsf{DC}}(G, N, \xi_Y)$ *if and only if ξ_Y is regular.*

The next theorem states that the cost of an optimal tree displayed by a network is bounded from below by the score of an optimal scenario.

▶ **Theorem 10** (Lower Bound Property)**.** *If S^* is an optimal tree displayed by N, and ξ^* is an optimal scenario of N then* $\mathsf{DC}(G, S^*) \geq \tilde{\mathsf{DC}}(G, N, \xi^*)$.

In our example from Fig. 1 and Fig. 2, the cost of S_1 and the score of E_1 are equal. However, in general, a regular scenario may not exist. For instance, if $G = (a, d)$, there is only one scenario ξ for N from Fig. 2, where a and d are mapped to a_1 and d_1, respectively. Then, ξ is not regular, and $0 = \tilde{\mathsf{DC}}(G, N, \xi) < \mathsf{DC}(G, N) = 1$ (for S_1 or S_2).

Finally, we present a crucial theoretical property used to solve ODT using solutions to instances of DC-MinRec.

▶ **Theorem 11** (Regularity)**.** *Let d be the score of an optimal scenario of N. A tree S displayed by N with* $\mathsf{DC}(G, S) = d$ *exists, if and only if there is an optimal regular scenario of N.*

4 Dynamic Programming (DP) Algorithm to solve DC-MinRec

Dynamic programming algorithms are commonly used in tree reconciliation, including models based on directed acyclic graphs (DAGs) [10, 16, 27, 37, 43], where a gene tree is mapped to a tree or a DAG through the lca-mapping or general mapping based on concepts close to our scenarios. Such approaches often lead to polynomial time solutions with square time complexity in the best case. Here, we present a dynamic programming solution to Problem 4 by providing formulas to compute the score of an optimal scenario.

Additional notation: By v' and v'', we denote the children of a tree node v, and by r' the child of a reticulation node r. For simplicity, instead of $\sigma(M_\xi(g))$ for a gene tree node g, we write ξ_g (i.e., $\xi_{[.]}$ is a mapping from G to N).

By $G|g$, we denote the subtree of G rooted at g. The main component of dynamic programming is δ such that for $g \in V(G)$ and $s \in V(N)$, $\delta(g, s)$ is the minimum score for $G|g$ in the set of all scenarios ξ between $G|g$ and $\hat{N}$ such that $\xi_g = s$. For simplicity, we ignore -1 from the $\tilde{\mathsf{DC}}$ formula in the partial costs in δ as this yields a constant term dependent on the size of G. Let $\tau(s)$ be equal 0 if s is a reticulation, and 1 otherwise. Then, we have the following dynamic programming formula that solves DC-MinRec:

$$\delta(g, s) = \begin{cases} \delta^f(g', s) + \delta^f(g'', s) & g \text{ and } s \text{ are tree nodes,} \quad (3) \\ \delta(g', s) + \delta(g'', s) & g \text{ is a tree node and } s \text{ is a leaf,} \quad (4) \\ 0 & g \text{ and } s \text{ are leaves labeled by the same species,} \quad (5) \\ +\infty & \text{otherwise,} \quad (6) \end{cases}$$

$$\delta^f(g, s) = \min(\delta(g, s), \tau(s') + \delta^{\uparrow}(g, s'), \tau(s'') + \delta^{\uparrow}(g, s'')), \quad (7)$$

$$\delta^{\uparrow}(g,s) = \begin{cases} \min(\delta(g,s), \tau(s')\tau(s'') + \min(\delta^{\uparrow}(g,s'), \delta^{\uparrow}(g,s''))) & s \text{ is a tree node,} \quad (8) \\ 1 + \delta^{\uparrow}(g,s') & s \in R(N), \quad (9) \\ \delta(g,s) & s \text{ is a leaf.} \quad (10) \end{cases}$$

In the next Lemma, we express properties satisfied by the above formulas.

▶ **Lemma 12.** *Let $g \in V(G)$, $s \in V(N)$ and all scenarios below are for G and N.*

D1 *$\delta(g,s)$ is equal to minimum number of Type I/II edges visited by edges from $E(G|g)$ among scenarios ξ satisfying $\xi_g = s$.*

D2 *If c is a child of g and t is not a reticulation. Then, $\delta^{\uparrow}(c,t)$ is equal to minimum number of Type I/II edges visited by edges from $E(G|c)$ plus the number of edges $e' = \langle a, b\rangle$ of Type II visited by $\{\langle c, g\rangle\}$ with $t \succeq \sigma(a)$ among scenarios ξ such that $\xi_g = s \succ t \succeq \xi_c$.*

D3 *If c is a child of g and s is a tree node. Then, $\delta^f(c,s)$ is equal to minimum number of Type I/II edges visited by edges from $E(G|c) \cup \{\langle c, g\rangle\}$ among scenarios ξ satisfying $\xi_g = s$.*

The optimal score is given by the following theorem, whose proof follows immediately from the definitions of δ, $\widetilde{\mathsf{DC}}$ and Lemma 12.

▶ **Theorem 13.** *Given a gene tree G and a phylogenetic network N. The score of an optimal scenario ξ^* is $\widetilde{\mathsf{DC}}(G, N, \xi^*) = -|E(G)| + \min_{s \in V(N)} \delta(G.\mathsf{root}, s)$.*

To infer an optimal scenario, we apply standard backtracking based on values of δ array. Since there are three arrays, each of size $|G||N|$ and every cell of an array can be computed in $O(1)$ time, DP has $O(|G||N|)$ time and space complexity. Note that in implementation δ^f can be embedded into δ computation. Thus, the space may be reduced to two arrays.

4.1 Inferring used reticulations edges from DP

An optimal scenario can be inferred from DP formulas using standard backtracking. However, this scenario may not be perfect. To further utilize the results of DP, we infer the set of used reticulation edges. For two nodes v and w, let $\rho(v,w) = \{\langle v, w\rangle\}$ denote the one-element set with $\langle v, w\rangle$ if the edge is a reticulation edge in N, and $\rho(v,w) = \emptyset$ otherwise. Similarly, by $\bar{\rho}(v,w)$ we denote the one-element set with the sibling edge of $e = \langle v, w\rangle$ if e is a reticulation edge in N, and $\bar{\rho}(v,w) = \emptyset$, otherwise. Then, DP components δ, δ^f and $\delta^{\uparrow}$ are associated with reticulation edge usage rules u, u^f, and $u^{\uparrow}$, resp., as follows:

$$u(g,s) = \begin{cases} u^f(g',s) \cup u^f(g'',s) & \text{in (3),} \\ \emptyset & \text{in (4)-(6),} \end{cases}$$

$$u^f(g,s) = \begin{cases} u(g,s) & \text{if } \delta^f(g,s) = \delta(g,s) \text{ in (7),} \\ u^{\uparrow}(g,c) \cup \rho(s,c) & \text{if } \delta^f(g,s) = \tau(c) + \delta^{\uparrow}(g,c) \text{ for some } c \in \{s', s''\} \text{ in (7),} \end{cases}$$

$$u^{\uparrow}(g,s) = \begin{cases} u(g,s) & \text{if } \delta^{\uparrow}(g,s) = \delta(g,s) \text{ in (8) or (10),} \\ u^{\uparrow}(g,c) & \text{in (9),} \\ u^{\uparrow}(g,c) \cup \rho(s,c) \cup \bar{\rho}(s, c.\mathsf{sibling}) & \text{if } \delta^{\uparrow}(g,s) = \tau(s')\tau(s'') + \delta^{\uparrow}(g,c) \\ & \text{for some } c \in \{s', s''\} \text{ in (8).} \end{cases}$$

The correctness of above formulas follows from the next lemma.

▶ **Lemma 14.** *If the backtracking of DP results in a scenario ξ, then $\Upsilon_\xi = u(G.\mathsf{root}, \xi_{G.\mathsf{root}})$.*

5 Inferring Optimal Displayed Trees

In this Section, we propose algorithms to solve ODT and its variants. We also answer whether the problem can be analogously solved for level-k networks and for broader classes of phylogenetic networks.

5.1 Solution to ODT

Thm. 11 motivates the following general branching algorithm to solve ODT. Suppose DP returns a solution with a conflict. Then, such a conflict can be resolved by branching and solving two sub-instances of the problem with phylogenetic networks induced from the input phylogenetic network by removing exactly one edge from the conflict. Let N_e be the phylogenetic network obtained from $N/\{e\}$ by contracting all semi-binary nodes[3]. Alg. 1 details the procedure to infer an optimal tree displayed by a given network. Here, branching occurs when there is a conflict in the set of used reticulation edges. Thus, if the number of conflicts is low, e.g., when G and N are similar, we expect a small number of DP invocations.

Algorithm 1 Optimal displayed tree inference: a solution to ODT.

1: **Input:** a gene tree G and a phylogenetic network N.
2: **Output:** an optimal tree displayed by a network N
3: Compute Υ_{ξ^*} (see Thm. 14) for some optimal scenario ξ^* for G and N inferred by DP.
4: Optimization: **Return** the empty tree with $+\infty$ score if the score of ξ^* is larger or equal to the currently known best score of a regular scenario (if already computed in a recursive call).
5: **If** Υ_{ξ^*} has no conflict **Then Return** N_Y for some perfect set $Y \supseteq \Upsilon_{\xi^*}$.
6: **Otherwise** Υ_{ξ^*} has a conflict, i.e., there are two sibling edges, say e and e' in Υ_{ξ^*}.
7: Recursively compute two optimal trees displayed by networks N_e and $N_{e'}$, resp.
8: **Return** the tree with the lower cost.

Correctness of Algorithm 1 follows from Thm. 11 and the following theorem.

▶ **Theorem 15.** *If e, $e' \in E_R(N)$ are sibling edges then* $\mathsf{DC}(G,N) = \min\{\mathsf{DC}(G,N_e), \mathsf{DC}(G,N_{e'})\}$. *Moreover, T is an optimal tree displayed by N if and only if T is an optimal tree displayed by a network N_e or $N_{e'}$ with minimum cost.*

In the worst case, we need to branch for every reticulation twice, which gives $2^{r+1}-1$ invocations of DP. Thus, Alg. 1 has time complexity $O(2^r|G||N|)$ in the worst case. However, as mentioned previously, we expect Alg. 1 to behave better than worst complexity in practice. See also our experimental evaluation in Section 6.

5.2 Lower and upper bounds of the optimal cost of a displayed tree

In applications where only the optimal cost is needed, for instance, in problems of network inference, we can use the Lower Bound Theorem 10. As the cost of an optimal displayed tree is bounded below by the score from DP, we can also compute the upper bound using regular scenarios returned from multiple invocations of DP. See details in Alg. 2.

▶ **Lemma 16.** *For G and N, Alg. 2 returns l and u such that $l \leq \mathsf{DC}(G,N) \leq u$.*

[3] Recall that N/X is the network obtained from N by removing all edges from X.

Algorithm 2 Branch and bound algorithm to approximate $\mathsf{DC}(G, N)$.

Input: a gene tree G, a phylogenetic network N, the maximal depth of recursion $d \geq 0$.
Output: a pair: lower and upper bounds of the cost of an optimal displayed tree for G and N.
Compute Υ_{ξ^*} and the optimal score c^* for some optimal scenario ξ^* for G and N.
If Υ_{ξ^*} has no conflict **Then Return** $\langle c^*, c^* \rangle$ *// Regular scenario; report the exact cost*
If the depth d of recursive invocations is reached **Then Return** $\langle c^*, +\infty \rangle$ *// Lower bound only*

Recursively compute bounds $\langle l, u \rangle$ and $\langle l', u' \rangle$ for G and N_e and $N_{e'}$, respectively, where e and e' are two conflicted sibling edges from Υ_{ξ^*}.
Return $\langle \min(l, l'), \min(u, u') \rangle$ *// Combine results from two invocations*

5.3 Inferring optimal trees displayed by level-k networks

Our results can also be extended to level-k networks. The definition and properties are adopted from [9, 12, 43]. A level-k network is a phylogenetic network in which every biconnected component has at most k reticulation nodes [9]. If B is a biconnected component of N, then by $B.\mathsf{root}$ we denote the unique node in B with no ancestors in B. Using the notation from [43], by $bc(N)$ we denote the tree obtained from N by contracting all its biconnected components. Let $\mathsf{Lab}(N)$ denote the set of species present in N as leaf labels.

Algorithm 3 Optimal displayed tree for level-k network.

Input: a gene tree G, a phylogenetic network N.
Output: the cost of optimal tree displayed by N.
$c = 0$ *// The cost variable*
For each biconnected component B from $bc(N)$ in postfix order:
  If $\mathsf{Lab}(G) \subseteq \mathsf{Lab}(B)$ Then **Return** $c + \mathsf{DC}(G, B)$ *// The case of root and the top component*
  Let F be a collection of all maximal subtrees of G rooted at g such that $\mathsf{Lab}(G|g) \subseteq \mathsf{Lab}(B)$.
  $c := c + \sum_{G' \in F} \mathsf{DC}^{\uparrow}(G', B)$ *// $\mathsf{DC}^{\uparrow}$ is a "$\delta^{\uparrow}$" variant of DC; see def. in Section 5.3*
  Let s_B be a new species label representing the component B.
  Replace B in N, and every $G' \in F$, by a leaf labeled s_B. *// Contract B and subtrees of G*
Return c

In Alg. 3, edges visited by subtrees of G have to be connected in the embedding. Therefore, for each non-root component B in $b(N)$, we minimize the score using the additional costs of a path to the root of B. Formally, $\mathsf{DC}^{\uparrow}(G, N)$ is the minimum value of $\mathsf{DC}(G, S) + ||M(G.\mathsf{root}), S.\mathsf{root}||$ in the set of all displayed trees S of N. Computing the value (almost) does not require modification of our algorithms. Here, instead of the formula from Thm. 13, we compute $\mathsf{DC}^{\uparrow}(G, N)$ using $-|E(G)| + \delta^{\uparrow}(G.\mathsf{root}, S.\mathsf{root})$. The correctness follows from Lemma 12 case D2. The formula can be easily embedded into Alg. 1. The time complexity of Alg. 3 is $O(2^k|G||N|)$.

5.4 Beyond tree-child networks

DP can be extended to analyse a broader class of networks, which is more beneficial from a practical point of view. Assume that instead of a tree-child network condition, our class of networks satisfies a relaxed condition: *a node has at most one reticulation child*. This assumption admits the child of reticulation to be a reticulation, which is not allowed in tree-child networks. Then, in DP, we have the following modification in (9): $\tau(s') + \delta^{\uparrow}(g, s')$, and in usage rules in the 2nd case of $u^{\uparrow}$ referring to (9): $u^{\uparrow}(g, s) \cup \rho(s, c)$, which is needed when the child is also a reticulation. Under this modification, Alg. 1 returns a correct optimal displayed tree. We omit details for brevity.

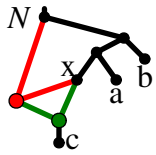


Figure 3 A non tree-child network.

We also analysed a general class of binary networks, i.e., in which a tree node may have two reticulation children. However, DP cannot correctly analyse such networks. When embedding a gene tree (a, b) into the network N from Fig. 3, we see that the optimal displayed tree is $S = ((a, b), c)$ with the cost 0. Here, S is constructed by removing a node x and all three incident edges, and a tree node $x.\mathsf{parent}$ with two children is also contracted. In the current DP, when a gene edge $\langle a.\mathsf{parent}, a\rangle$ from G visits $x.\mathsf{parent}$ DP will increase the cost. Therefore, the lower bound property is not satisfied in this case unless a solution in which such removed tree nodes are detected is implemented. It remains open whether it can be done in polynomial time without checking all variants of displayed trees.

6 Experimental evaluation

In this Section, we present the experimental evaluation using our prototype implementation of DP, Alg. 1 and Alg. 2 called `EmbRetNet` written in Python 3. The algorithms were extended to analyse the class of networks described in Section 5.4. The software package is available from `https://bitbucket.org/pgor17/embretnet`.

Performance of inferring optimal displayed trees

Our Alg. 1 is exponential in the worst case; however, we expect better performance when the gene tree is similar to the network, which is expected for empirical data. In this Section, we summarize the results of several experiments to compare the performance of our implementation of Alg. 1 to the naïve implementation in which all trees displayed by a given network are generated, and then, the costs are computed using a linear time solution from [48]. Note that both algorithms have exponential time complexity; however, the naïve algorithm always has the same number of steps, proportional to $2^r(|G| + |N|)$, where r is the number of reticulations in N. Experiments were conducted on a Ubuntu server with Intel(R) Xeon(R) CPU E5-2698 v4@2.20GHz (80 cores) and 500 GB of RAM.

6.1 Evaluation on random datasets

Data preparation: To generate random tree-child phylogenetic networks, we used an algorithm from [26] and its implementation in Python from GitHub with a slight modification to generate only binary networks. Random gene trees with one-to-one labeling of leaves were generated using the Yule-Harding model. Then, we generated datasets R1, R2, and R3, each consisting of $2 \cdot 10 \cdot 100$ pairs of random gene trees and networks. For each $n \in \{12, 20\}$ and $r \in \{1, 2, \ldots, 10\}$, we generated 100 pairs $\langle G, N\rangle$, such that N is a network with n leaves and r reticulations and in dataset R1 $|L(G)| = n$, in R2 the number of leaves in G is sampled uniformly from the interval $[2, n]$, and in R3 G is a randomly chosen tree displayed by N.

Discussion: In the two left panels of Fig. 4, we summarize the experimental evaluation results of R1 and R3, where G and N were generated independently. Since these datasets represent extremes, in which G and N are highly different, our algorithm frequently infers

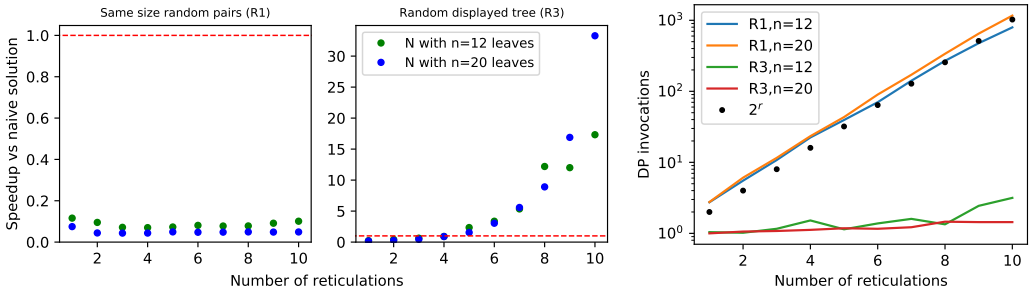


Figure 4 *Left:* Performance of Alg. 1 vs naïve approach for random datasets R1 and R3. Each dot represents the average speedup computed from the runtimes of 100 pairs of gene trees and phylogenetic networks. *Right:* Average number of DP invocations necessary to calculate an answer for random datasets R1 and R3. Since the results of R2 are similar to R1, we moved them to Appendix A.1.

conflicted sets of reticulation edges by visiting almost every possible scenario, thus achieving nearly the pessimistic exponential complexity. In consequence, it is noticeably slower than the naïve one for R1. On the contrary, with data from R3 the algorithm rarely branches, and its average runtime matches the complexity of DP, i.e., $O(|G||N|)$. Even with a larger constant factor, we outperform the naïve algorithm for $r > 4$, achieving > 15 times speedup for $r = 10$.

6.2 Evaluation on simulated datasets

Our simulation procedure can be divided into three major phases: (i) simulating tree-child phylogenetic networks, (ii) simulating gene trees, and (iii) introducing errors to the gene trees. The selection of the parameters in all three phases is mainly based on the simulation study conducted by Molloy and Warnow [34], which uses parameters derived from a fungal dataset presented by Rasmussen and Kellis [36]. See Appendix A.2 for more details.

(i) Simulating tree-child phylogenetic networks. First, we simulated species trees using a general sampling approach implemented in R package TreeSim version 2.4 [21] with the parameters from [34]. The number of leaves was set to 12 or 20. Next, we inferred a network with r reticulations from each of the simulated species trees, where r was uniformly sampled from $[1, 10]$. We followed a popular study by Solis-Lemus and Ané [39]. To add r reticulations, we started by randomly choosing r pairs of edges in the species tree. We then subdivided both edges in a pair, making two new vertices. Finally, we added a reticulation edge between the vertices. This method generates networks belonging to the tree-based class [14]. Similarly to [31], the networks were constrained to be tree-child.

(ii) Simulating gene trees. For each phylogenetic network, we randomly chose one of its displayed trees, obtaining one of the possible trees, along which gene families evolve. We simulated one gene tree per displayed tree using SimPhy version 1.0.2 [30]. Similarly to [34], we used three rates of duplication/loss (DL) $\{10^{-10}, 2 \cdot 10^{-10}, 5 \cdot 10^{-10}\}$ and two values of the effective population size $\{10^7, 5 \cdot 10^7\}$, corresponding respectively to a low and a medium level of incomplete lineage sorting (ILS). Altogether, we used six sets of simulation parameters, which allowed us to obtain a diversified set of gene trees. Additionally, we set the minimum number of leaves to 3.

(iii) Simulating sequences and estimating gene trees. To introduce errors to the generated gene trees, we simulated sequences along with them and estimated gene trees from multiple sequence alignments using the maximum-likelihood method (MLE). DNA sequences were simulated by INDELible v1.03 [13] and SimPhy [30]. Again, we followed the parameters proposed in [34]. The alignment length was set to 1000 bp. To estimate gene trees, we used a true alignment returned by INDElible. Then, we inferred ML-trees by PhyML v.3.1 [19] using GTR+Γ model. Finally, to obtain a rooted gene tree from an unrooted ML-tree, we conducted midpoint-plateau rooting implemented in URec [17] using the corresponding displayed tree inferred in step (ii) of our pipeline.

Finally, for each set of parameters of duplication-loss rates, population sizes, reticulation values, and leaf-set sizes we simulated 100 networks and 100 corresponding gene trees. The simulations were run in parallel on 10 cores and the total simulation time was under 8 hours. The algorithm took 2 hours to process all datasets, and it took, on average, 45 seconds to run 100 instances with 20 leaves and 10 reticulations for low ILS and low DL.

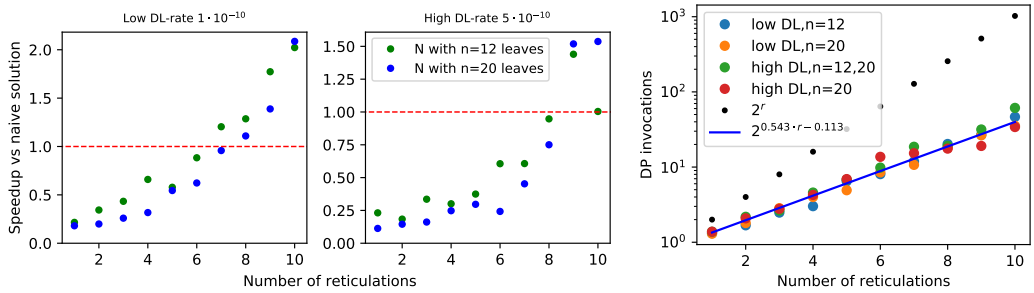


Figure 5 *Left:* Performance of Alg. 1 vs. naïve approach for datasets with low ILS ($1 \cdot 10^7$). See Appendix A.3 for a complete set. *Right:* Average number of DP invocations necessary to calculate an answer for datasets with low ILS. The blue line is calculated using least squares linear regression, that best approximates results for all simulated datasets.

Discussion. In Fig. 5, we present diagrams showing the results of evaluations for two datasets. The results for the remaining datasets are similar (please refer to Appendix). The way we simulated data makes trees and networks more similar to each other. Thus, we can see significant improvements vs. random datasets. Regardless of parameter choices, we start to outperform the naïve solution for $r > 9$. For simulated data closest to reality (low ILS, low DL), we achieved better results for $r > 7$. The results suggest a hybrid approach in Alg. 1: enumerate all displayed trees to compute DC costs directly if the network has a low number of reticulations (e.g., $r < 9$).

To estimate the average runtime of our algorithm, we first calculated the depths d of the recursive calls as $\log_2$ of the number of DP invocations from each experiment. Then, we found that for all points $\langle r, d\rangle$ from our experiments, $d = 0.543r - 0.1135$ is the fitted least squares regression line having the standard error of .011 (see the rightmost diagram in Fig. 5). We conclude that, despite the worst case theoretical complexity, i.e., $O((2^{r+1}-1)|G||N|)$, the real runtime of our implementation on simulated data is proportional to $2^{0.543r}|G||N|$ and outperforms the naïve approach starting from small r's. We claim that a similar statement holds for the algorithm with level-k networks. In other words, it is possible to analyse empirical networks even with $r = k = 40$, since the exponent can be reduced by half.

6.3 Empirical tests

Our final experiment was conducted using real data. We revisited research presented in [29] concerning coronavirus (CoV) phylogeny. In the cited paper, the authors investigated the origins of the SARS-CoV-2 virus which causes a severe respiratory disease. They validated the hypothesis that the appearance of this new coronavirus is a consequence of several recombination events that occurred between some evolutionarily close CoV species. The results showed that both intergenic and intragenic recombination played a significant role in the SARS-CoV-2 evolution.

The goal of our study was to test whether scenarios with the lowest DC cost inferred for the phylogenetic network from [29] and individual gene trees confirm recombinations identified in the cited paper. In our experiment, we focused on intergenic recombinations, in which a whole gene is transferred from one species and integrated into another species genome.

Data preparation: In the gene trees inference step, we followed the cited work. We selected 15 out of 25 examined coronavirus species, omitting a few species to avoid multifurcations. As representatives of SARS-Cov-2, two variants were used. One was sampled from a patient from Wuhan (Hu-Wuhan), the origin of the pandemic spread of coronavirus, and the other was collected in Italy (Hu-Italy). Other selected species were RaTG13 bat CoV from *R. affinis* which, at first, was considered the only close relative of SARS-CoV-2, bat CoV ZC45 and ZXC21 strains from Zhejiang province of China (Bat-CoVZC45, and Bat-CoVZXC21), bat coronaviruses collected from species found in several provinces of China and from Bulgaria (Rf1, HKU3-12, BatCoV273, BatCoV279, and BM48-31 BGR), two CoV strains from Guangdong and Guangxi pangolins (Guangdong-Png, Guangxi-Png-P2V), and three SARS CoV related species (SARS, SARS-BJ182-4, and Rs3367). For the list of full names please refer to Appendix A.4. Coronavirus sequences were obtained from GenBank [2] at `https://www.ncbi.nlm.nih.gov/` and GISAID [38] databases. In the studied phylogenetic network, recombinations were found in the case of the genes M, ORF1ab, ORF3a, ORF6, ORF8, and ORF10; therefore, our research was focused on this set of genes. Multiple sequence alignments for the gene families were performed with MUSCLE [11] and corrected by GBlocks [7] with less stringent correction option. The ML gene trees were inferred using RAxML [40] with parameters described in [29]. All species were present in all gene families except ORF8, which lacks the BM48-31-BGR species.

Phylogenetic networks: The coronavirus tree with marked intergenic recombinations ($H1$-$H7$) identified in [29] is depicted in Fig. 6. Since the direction of three out of seven recombinations was not certain, we prepared 8 networks corresponding to all combinations of the directions of gene transfers. Each network is named with three letters L/R responding to the direction of $H5$, $H6$ and $H7$, respectively, i.e. in the LRL network $H5$ and $H7$ are directed left and $H6$ is directed right. Please note that the inferred networks are not tree-child and therefore in this experiment we use the extended version of our algorithm described in Section 5.4.

Discussion: The results of the experiment are depicted in Fig 6. For each gene family, we checked whether the expected reticulation edge was used by the inferred scenario with the lowest DC cost. We can distinguish three possibilities for reticulation edge e: 1. e was used by the expected gene, 2. e was used by one or more extra genes, and 3. the expected gene didn't use e. We were able to confirm most of the reticulations except two: $H2$ that was reported in [29] with the lowest support wasn't confirmed by any of the networks, and $H7$

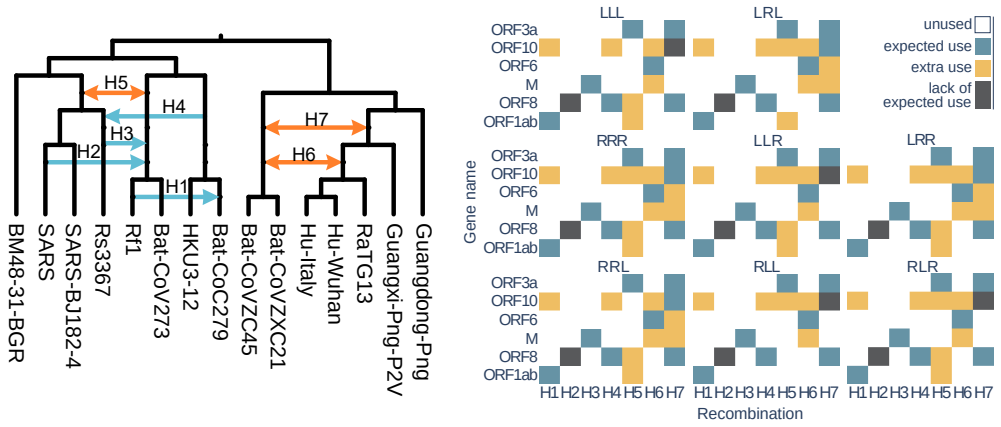


Figure 6 *Left:* Coronavirus species tree with recombinations $H1$-$H7$ reported in [29]. Genes-recombinations assignment from [29]: $H1$:ORF1ab, $H2$:ORF8, $H3$:M, $H4$:ORF8, $H5$:ORF3a, $H6$:ORF6, $H7$:ORF3a, ORF8, ORF10. *Right:* Results showing which recombination edges were used by each gene tree in the scenarios with the lowest DC cost for each network variant.

was confirmed only by networks with $H6$ directed right. The most extra uses were found for $H5$ and gene $ORF10$ which gene tree had low support values. Further research might be performed for these cases. The least extra uses were present in $RLL(5)$ and $LLL(6)$, which may be some lead when investigating the direction of $H6$ and $H7$.

We observed that, for a fixed gene tree, the set of optimal displayed trees inferred by our algorithm and their cost, is independent of the network variant. This phenomenon needs further theoretical investigation. Costs for each gene tree have the following values: ORF3a: 0, ORF10: 5, ORF6: 1, M: 3, ORF8: 2, ORF1ab: 2. This observation may lead to discovering some important property and can be a subject for further investigation.

7 Conclusions

In this work, we have investigated the problem of inferring an optimal tree displayed by a network under the deep coalescence cost and proposed a new score to approximate the cost. We have shown that the score has nice mathematical and computational properties allowing us to bound the cost of an optimal displayed tree from below. We have proposed a polynomial-time dynamic programming (DP) algorithm to compute the score together with the set of used reticulation edges that yielded the score. Then, we have proposed a new way to infer a displayed tree by a recursive procedure resolving conflicts detected in multiple invocations of DP. In the worst case, our algorithm to infer an optimal tree requires 2^{r+1}-1 DP invocations, where r is the number of reticulations. However, numerous tests on simulated data have indicated that the exponent may be reduced by half on average. This phenomenon is explained by similarity, i.e., we expect a low number of conflicts if a gene tree is more congruent with its network. In other words, the average runtime of $\Omega(2^{0.543r}|G||N|)$ can compete on empirical datasets with exhaustive enumeration strategies (either on the level of a whole network or each biconnected component independently) commonly used in alternative approaches to scoring tree-network pairs [25, 27, 43]. We also claim that the statement holds for level-k networks by replacing r by k in the formula. We conclude that our conflict resolution algorithm enables analyses of complex networks with dozens of reticulation events. We also claim that resolving conflicts returned by dynamic programming is a new alternative towards designing efficient algorithms that utilize internal similarities of empirical datasets.

Future Outlooks. Resolving conflicts in the usage of reticulation edges can be naturally generalized to other cost functions, e.g., gene duplication cost. Also, it is not difficult to extend DP to analyze unrooted gene trees. Another critical question is whether the runtime exponent can be further reduced, e.g., by choosing optimal scenarios with the smallest possible sets of conflicted reticulation edges. Furthermore, we would like to test the efficiency and accuracy of the branch and bound algorithm to approximate the optimal cost. Also, we plan to apply the methods in computationally demanding problems of network inference from sets of gene trees, which may require reimplementation in a low-level programming language (e.g., C/C++).

References

1 Eric Bapteste, Leo van Iersel, Axel Janke, Scot Kelchner, Steven Kelk, James O. McInerney, David A. Morrison, Luay Nakhleh, Mike Steel, Leen Stougie, and James Whitfield. Networks: expanding evolutionary thinking. *Trends in Genetics*, 29(8):439–441, 2013.

2 Dennis A Benson, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic Acids Research*, 39(suppl_1):D32–D37, 2010.

3 Luis Boto. Horizontal gene transfer in evolution: facts and challenges. *Proceedings of the Royal Society B: Biological Sciences*, 277(1683):819–827, November 2009.

4 Gabriel Cardona, Francesc Rosselló, and Gabriel Valiente. Comparison of tree-child phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(4):552–569, 2008.

5 Gabriel Cardona, Francesc Rossello, and Gabriel Valiente. Comparison of tree-child phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(4):552–569, October 2009.

6 Gabriel Cardona and Louxin Zhang. Counting and enumerating tree-child networks and their subclasses. *Journal of Computer and System Sciences*, 114:84–104, 2020.

7 Jose Castresana. Selection of conserved blocks from multiple alignments for their use in phylogenetic analysis. *Molecular Biology and Evolution*, 17(4):540–552, 2000.

8 Ruchi Chaudhary, J Gordon Burleigh, and Oliver Eulenstein. Efficient error correction algorithms for gene tree reconciliation based on duplication, duplication and loss, and deep coalescence. In *BMC Bioinformatics*, volume 13, pages 1–10. BioMed Central, 2012.

9 Charles Choy, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Computing the maximum agreement of phylogenetic networks. *Theoretical Computer Science*, 335(1):93–107, 2005.

10 Beatrice Donati, Christian Baudet, Blerina Sinaimeri, Pierluigi Crescenzi, and Marie-France Sagot. EUCALYPT: efficient tree reconciliation enumerator. *Algorithms for Molecular Biology*, 10(1):3, 2015.

11 Robert C Edgar. Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):1–19, 2004.

12 Mareike Fischer, Leo Van Iersel, Steven Kelk, and Celine Scornavacca. On computing the maximum parsimony score of a phylogenetic network. *SIAM Journal on Discrete Mathematics*, 29(1):559–585, 2015.

13 William Fletcher and Ziheng Yang. Indelible: a flexible simulator of biological sequence evolution. *Molecular Biology and Evolution*, 26(8):1879–1888, 2009.

14 Andrew R. Francis and Mike Steel. Which phylogenetic networks are merely trees with additional arcs? *Systematic Biology*, 64(5):768–777, June 2015.

15 Paweł Górecki, Oliver Eulenstein, and Jerzy Tiuryn. Unrooted tree reconciliation: A unified approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(2):522–536, 2013.

16 Paweł Górecki and Jerzy Tiuryn. DLS-trees: A model of evolutionary scenarios. *Theoretical Computer Science*, 359(1-3):378–399, 2006.

17 Paweł Górecki and Jerzy Tiuryn. Urec: a system for unrooted reconciliation. *Bioinformatics*, 23(4):511–512, 2007.

18 Benjamin E. Goulet, Federico Roda, and Robin Hopkins. Hybridization in plants: Old ideas, new techniques. *Plant Physiology*, 173(1):65–78, November 2016.

19 Stéphane Guindon, Jean-François Dufayard, Lefort Vincent, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: Assessing the performance of phyml 3.0. *Systematic Biology*, 59(3):307–321, 2010.

20 Dan Gusfield. *ReCombinatorics: the Algorithmics of Ancestral Recombination Graphs and Explicit Phylogenetic Networks*. MIT Press, Boston, 2014.

21 Klaas Hartmann, Dennis Wong, and Tanja Stadler. Sampling trees from evolutionary models. *Systematic Biology*, 52(4):465–476, 2010.

22 Katharina T. Huber and Vincent Moulton. Phylogenetic networks from multi-labelled trees. *Journal of Mathematical Biology*, 52(5):613–632, 2006.

23 Katharina T. Huber, Vincent Moulton, Mike Steel, and Taoyang Wu. Folding and unfolding phylogenetic trees and networks. *Journal of Mathematical Biology*, 73(6-7):1761–1780, 2016.

24 Daniel H. Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic Networks: Concepts Algorithms and Applications*. Cambridge University Press, New York, 2010.

25 Leo Van Iersel, Mark Jones, and Celine Scornavacca. Improved maximum parsimony models for phylogenetic networks. *Systematic Biology*, 67(3):518–542, December 2017.

26 Remie Janssen and Yukihiro Murakami. Linear time algorithm for tree-child network containment. In *International Conference on Algorithms for Computational Biology*, pages 93–107. Springer, 2020.

27 Matthew LeMay, Ran Libeskind-Hadas, and Yi-Chieh Wu. A polynomial-time algorithm for minimizing the deep coalescence cost for level-1 species networks. *bioRxiv*, November 2020.

28 Wayne P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

29 Vladimir Makarenkov, Bogdan Mazoure, Guillaume Rabusseau, and Pierre Legendre. Horizontal gene transfer and recombination analysis of SARS-CoV-2 genes helps discover its close relatives and shed light on its origin. *BMC Ecology and Evolution*, 21(1):1–18, 2021.

30 Diego Mallo, Leonardo De Oliveira Martins, and David Posada. Simphy: Phylogenomic simulation of gene, locus, and species trees. *Systematic Biology*, 65(2):334–344, 2015.

31 Alexey Markin, Tavis K. Anderson, Venkata SKT Vadali, and Oliver Eulenstein. Robinson-foulds reticulation networks. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 77–86, 2019.

32 Sarah M. McDonald, Martha I. Nelson, Paul E. Turner, and John T. Patton. Reassortment in segmented RNA viruses: mechanisms and outcomes. *Nature Reviews Microbiology*, 14(7):448–460, May 2016.

33 Siavash Mirarab and Tandy Warnow. ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. *Bioinformatics*, 31(12):i44–i52, 2015.

34 Erin K Molloy and Tandy Warnow. Fastmulrfs: fast and accurate species tree estimation under generic gene duplication and loss models. *Bioinformatics*, 36(Supplement_1):i57–i65, 2020.

35 Yukihiro Murakami, Leo van Iersel, Remie Janssen, Mark Jones, and Vincent Moulton. Reconstructing tree-child networks from reticulate-edge-deleted subnetworks. *Bulletin of Mathematical Biology*, 81(10):3823–3863, 2019.

36 Matthew D. Rasmussen and Manolis Kellis. Unified modeling of gene duplication, loss, and coalescence using a locus tree. *Genome Research*, 22(4):755–765, 2012.

37 Celine Scornavacca, Joan Carles Pons Mayol, and Gabriel Cardona. Fast algorithm for the reconciliation of gene trees and lgt networks. *Journal of Theoretical Biology*, 418:129–137, 2017.

38 Yuelong Shu and John McCauley. Gisaid: Global initiative on sharing all influenza data–from vision to reality. *Eurosurveillance*, 22(13):30494, 2017.

39 Claudia Solís-Lemus and Cécile Ané. Inferring phylogenetic networks with maximum pseudo-likelihood under incomplete lineage sorting. *PLOS Genetics*, 12(3):1–21, 2016.

40 Alexandros Stamatakis. Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

41 Cuong Than and Luay Nakhleh. Species tree inference by minimizing deep coalescences. *PLoS Computational Biologie*, 5(9):e1000501, 2009.

42 Cuong Than, Derek Ruths, and Luay Nakhleh. PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC Bioinformatics*, 9(1), July 2008.

43 Thu-Hien To and Celine Scornavacca. Efficient algorithms for reconciling gene trees and species networks via duplication and loss events. *BMC Genomics*, 16(S10), 2015.

44 Yi-Chieh Wu, Matthew D Rasmussen, Mukul S Bansal, and Manolis Kellis. Most parsimonious reconciliation in the presence of gene duplication, loss, and deep coalescence using labeled coalescent trees. *Genome Research*, 24(3):475–486, 2014.

45 Yun Yu, R. Matthew Barnett, and Luay Nakhleh. Parsimonious inference of hybridization in the presence of incomplete lineage sorting. *Systematic Biology*, 62(5):738–751, July 2013.

46 Yun Yu, James H. Degnan, and Luay Nakhleh. The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. *PLoS Genetics*, 8(4):e1002660, April 2012.

47 Yun Yu, Tandy Warnow, and Luay Nakhleh. Algorithms for MDC-based multi-locus phylogeny inference: Beyond rooted binary gene trees on single alleles. *Journal of Computational Biology*, 18(11):1543–1559, November 2011.

48 Louxin Zhang. From gene trees to species trees ii: Species tree inference by minimizing deep coalescence events. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(6):1685–1691, 2011.

49 Jiafan Zhu, Yun Yu, and Luay Nakhleh. In the light of deep coalescence: revisiting trees within networks. *BMC Bioinformatics*, 17(S14), November 2016.

A Experimental evaluation

A.1 Random datasets results

Figure 7 presents complete set of diagrams for random datasets.

A.2 Simulated data preparation

A.2.1 Species trees parameters

To simulate species trees, we ran sim.bd.taxa.age function implemented in R package TreeSim version 2.4 [21] with the following parameters: tree height = 1800000337.5 years, speciation rate = $1.8 * 10^{-9}$ events/year and extinction rate = 0 events/year. The number of taxa was set to either 12 or 20.

A.2.2 Inferring networks from species trees

After simulating a species tree using the general sampling approach, we assigned a time value to each of its nodes, corresponding to a length of a path connecting the root with the node. Note that the general sampling approach produces ultrametric species trees, therefore time values assigned to the leaves were equal.

Then we inferred a tree-child phylogenetic network with r reticulations from each of the species trees. We added r reticulations one by one by repeating the following procedure. To add a reticulation edge to a species tree/network, we start by randomly choosing a pair of branches and subdivide them, making two new vertices. We then sample a time value for each of the vertices from uniform(vertex.parent.time, vertex.child.time). Finally, we add a

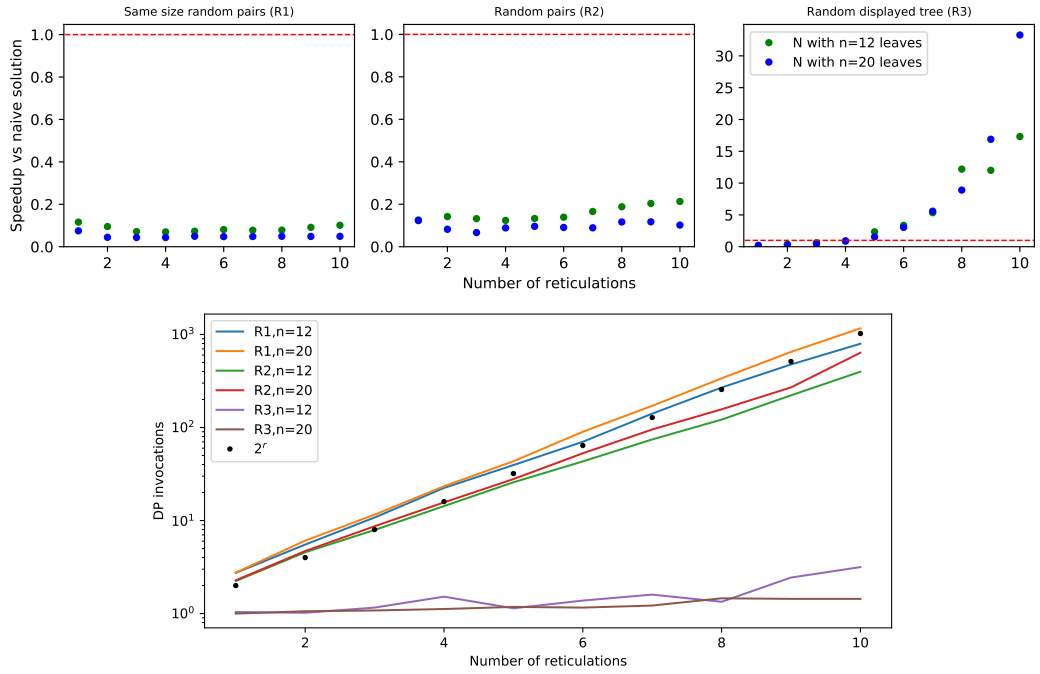


Figure 7 *Top:* Performance of Alg. 1 vs naïve approach for random datasets R1, R2, R3. *Bottom:* Average number of DP invocations necessary to calculate an answer for random datasets R1, R2 and R3.

reticulation edge from the vertex with the lower time value t_l to the vertex with the higher time value t_h. If the addition disturbs the tree-child property, we delete the reticulation edge and contract the vertices. Otherwise, we set the length of the reticulation edge to t_h - t_l.

Note that this way of introducing reticulation edges does not change time values of the leaves, hence all displayed trees of the resulting network are ultrametric and have equal heights.

A.2.3 Gene trees parameters

To simulate a gene tree from a randomly chosen displayed tree we ran SimPhy [30] with the following command:

```
simphy_lnx64 -sr <displayed tree> -rg 1 -rl 1 -si F:1 \
-sp F:$ps su F:0.0000000004 -lb F:$dl -ld F:lb -ll 3\
-hg LN:1.5,1 -oc 1 -o <output folder> -v 0 -cs 22
```

where <displayed tree>is a nexus file containing the randomly chosen displayed tree, $ps is a population size and $dl is a duplication/loss rate.

A.2.4 Sequences parameters

The sequences were simulated using INDELible v1.03 [13] by running a perl script INDELIble_wrapper.pl included in SimPhy [30]. We used GTR model with substitution rates (AC, AG, AT, CG, CT and GT respectively) sampled for each gene tree from Dirichlet(12.776722, 20.869581, 5.647810, 9.863668, 30.679899, 3.199725). The nucleotide frequencies (T, C, A and G respectively) were sampled from Dirichlet (113.48869, 69.02545, 78.66144, 99.83793), whilst α parameter was sampled from Lognormal(-0.470703916, 0.348667224). The alignment length was set to 1000 bp.

A.3 Simulated datasets results

Figures 8 and 9 present the complete set of diagrams.

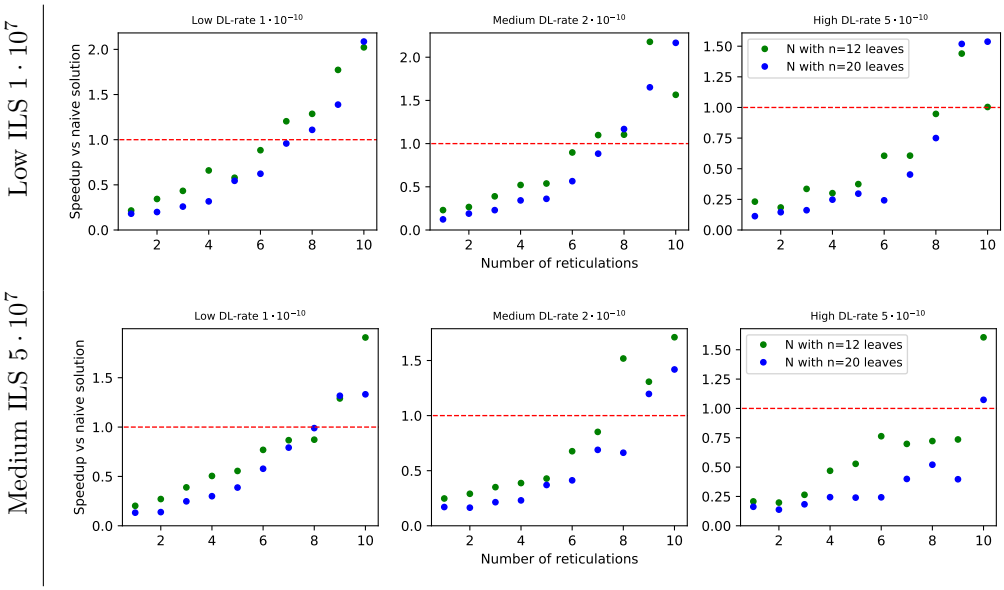


Figure 8 Performance of Alg. 1 vs naïve approach for six simulated datasets.

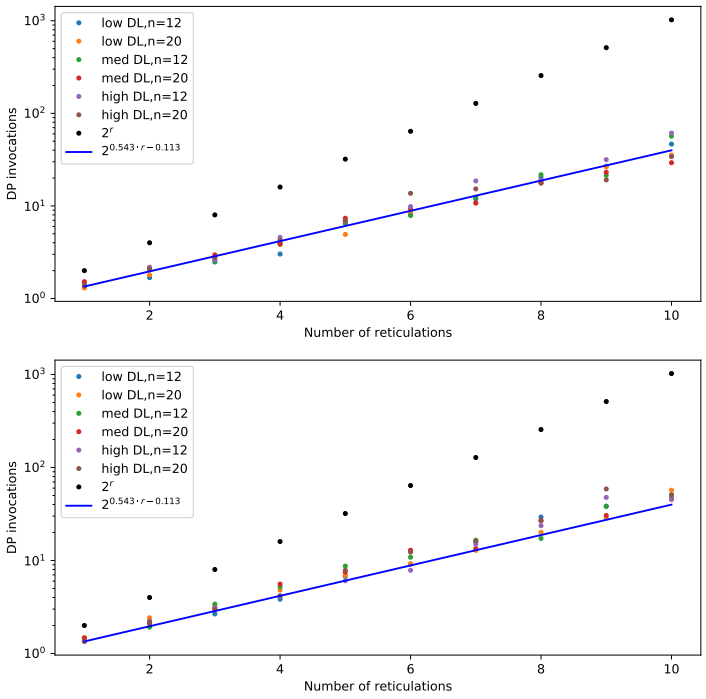


Figure 9 *Top:* Average number of DP invocations necessary to calculate an answer for datasets with low ILS. *Bottom:* Average number of DP invocations necessary to calculate an answer for datasets with medium ILS. Recall that blue line represents coefficients calculated by linear regression for data with low and medium ILS combined.

A.4 Empirical experiment dataset

Organism names and database accession numbers of all species used in our research are listed in Table 1.

Table 1 List of the full names and database accession numbers of coronavirus species used in our research. Species were chosen from the dataset studied in [29].

Abbreviated name	Organism name	Accession Number (GenBank/GISAID)	Host organism
Hu-Wuhan	BetaCoV Wuhan-Hu-1	NC_045512.2	Human
Hu-Italy	hCoV-19/Italy/ ABR-IZSGC-TE4836/2020	EPI_ISL_418260	Human
RaTG13	Bat CoV RaTG13	MN996532.1	Bat
Guangdong-Png	hCoV-19/pangolin/ Guangdong/1/2019	EPI_ISL_410721	Pangolin
Guanxi-Png-P2V	Pangolin CoV isolate PCoV_GX-P2V	MT072864	Pangolin
Bat-CoVZC45	Bat SARS-like CoV isolate bat-SL-CoVZC45	MG772933.1	Bat
Bat-CoVZXC21	Bat SARS-like CoV isolate bat-SL-CoVZXC21	MG772934.1	Bat
Bat-CoV273	Bat CoV BtCoV/273/2005	DQ648856.1	Bat
Bat-CoV 279	Bat CoV BtCoV/279/2005	DQ648857.1	Bat
HKU3-12	Bat SARS CoV HKU3-12	GQ153547.1	Bat
Rf1	Bat SARS CoV Rf1	DQ412042.1	Bat
SARS	SARS CoV BJ01	AY278488.2	Human
SARS-BJ182-4	SARS CoV BJ182-4	EU371562	Human
Rs3367	Bat SARS-like CoV Rs3367	KC881006.1	Bat
BM48-31-BGR	Bat CoV BM48-31/ BGR/2008	GU190215.1	Bat

Genome Halving and Aliquoting Under the Copy Number Distance

Ron Zeira[1]
Department of Computer Science, Princeton University, NJ, USA

Geoffrey Mon[1]
Department of Computer Science, Princeton University, NJ, USA

Benjamin J. Raphael[2]
Department of Computer Science, Princeton University, NJ, USA

Abstract

Large-scale genome rearrangements occur frequently in species evolution and cancer evolution. While the computation of evolutionary distances is tractable for balanced rearrangements, such as inversions and translocations, computing distances involving duplications and deletions is much more difficult. In the recently proposed Copy Number Distance (CND) model, a genome is represented as a Copy Number Profile (CNP), a sequence of integers, and the CND between two CNPs is the length of a shortest sequence of deletions and amplifications of contiguous segments that transforms one CNP into the other. In addition to these segmental events, genomes also undergo global events such as Whole Genome Duplication (WGD) or polyploidization that multiply the entire genome content. These global events are common and important in both species and cancer evolution. In this paper, we formulate the *genome halving problem* of finding a closest preduplication CNP that has undergone a WGD and evolved into a given CNP under the CND model. We also formulate the analogous *genome aliquoting problem* of finding the closest prepolyploidzation CNP under the CND distance. We give a linear time algorithm for the halving distance and a quadratic time dynamic programming algorithm for the aliquoting distance. We implement these algorithms and show that they produce reasonable solutions on simulated CNPs.

2012 ACM Subject Classification Applied computing → Molecular evolution

Keywords and phrases Genome rearrangements, Copy number distance, Whole genome duplication, polyploidization, genome halving distance, genome aliquoting distance

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.18

Supplementary Material *Software (Source Code)*: `https://github.com/raphael-group/CND-aliquoting`

Funding *Benjamin J. Raphael2]Corresponding author*: This work is supported by a US National Institutes of Health (NIH) grants U24CA211000 and U24CA248453.

1 Introduction

Genomes evolve over time through many types of mutations ranging from single-nucleotide mutations through large-scale alterations that affect both the order and the amount of genetic material. Such large-scale changes, termed *genome rearrangements*, are observed both in the evolution of species and of cancer cells [38, 28, 37, 10, 6, 23, 16]. Genome rearrangements can be categorized into two classes: structural rearrangements such as reversals, translocations and transpositions that change the order of DNA segments but not their quantity; and numerical rearrangements such as duplications and deletions that either create new copies of existing DNA segments or remove segments of the genome. The result of one genome evolving into another by a series of genome rearrangements is a new genome having a different structure and amount of DNA.

[1] First author
[2] Corresponding author

Computational models of genome rearrangements aim to calculate the minimum number of rearrangement events that transform one genome into the other, also called the *genome rearrangement distance*. Several types of rearrangement models have been proposed for structural events including the breakpoint (BP) distance [32, 31, 39], single-cut-or-join (SCJ) [13], the Hannenhalli-Pevzner model (HP) [18, 19] and the more general double-cut-and-join (DCJ) model [43, 3]. Representing the genome in these models requires the identification of homologous segments between the two genomes analyzed and determining the adjacencies between these segments in each genome. However, while these models admit polynomial time algorithms for the rearrangement distance when each genomic segment has a single copy in each genome, allowing for multiple copies often results in NP-hard problems [14].

Besides local rearrangement events, *whole genome duplication* (*WGD*) or polyploidization (>2 multiplication) events are viewed as a fundamental step in genome evolution as the multiplication of the genomic contents allows greater diversification of gene functions. In species evolution there has been strong evidence for WGD events reported for vertebrates [21], yeast [42] and for many plant genomes [5]. In fact, all angiosperms have undergone at least one WGD event in their evolutionary history and polyploidization is recognized as a major driving force for plant speciation [29]. In addition to species evolution, WGD is also a frequent event in cancer evolution with an estimated frequency of more than 30% in recent cancer studies [7, 45, 4, 9]. Furthermore, WGD is associated with poor prognosis across cancer types [4].

A basic question in the computational analysis of WGD is to reconstruct a closest ancestral preduplicated genome for a given extant genome. Namely, given a genome G and a rearrangement distance $d(\cdot,\cdot)$, the *genome halving problem* seeks to find an ancestral genome A such that the rearrangement distance $d(2A, G)$ between the duplicated genome $2A$ and the extant genome G is minimized [12]. The genome halving problem can be solved in linear time for the BP distance [39, 22], the SCJ distance [13], the HP distance [1] and the DCJ distance [25, 40, 2]. A generalization of the halving problem for polyploidization, i.e. finding a closest premultiplication ($p > 2$) genome is called the *genome aliquoting problem* [40]. The genome aliquoting problem can be solved in polynomial time for the BP distance [41] and the SCJ distance [13], while the complexity of the problem is not resolved for the DCJ distance, though a 2-approximation algorithm exists [41]. Apart from finding a preduplication genome, the genome halving and aliquoting problems also measure how close the extant genome is to a duplicated genome. By comparing different values of p. this allows us, for example, to distinguish if a genome has undergone duplication or triplication.

Motivated by applications in cancer evolution, alternative genome rearrangement models that focus on numerical rearrangement have recently been introduced [34, 8]. These models represent a genome as a *copy number profile* (*CNP*), a vector of integers indicating the number of copies of each segment from a reference genome. Thus, unlike structural rearrangement models, CNPs do not model the sequence of rearranged segments, but only the number of copies of segments of the reference genome. Therefore, genomes with different order of the segments may have the same CNP. However, the CNP representation is useful because it can be readily derived from DNA sequencing data or microarrays [7, 26, 17, 27, 35, 15]. The Copy Number Transformation (*CNT*) model models the evolution of CNPs by amplifications and deletions [34]. In this model an amplification (resp. deletion) increases (resp. decreases) the entries in a contiguous interval of the CNP. The Copy Number Distance (*CND*) between two profiles is defined as the length of a shortest sequence of amplifications and deletions that transform one profile into the other. The CND can be computed in linear time [46], and has

been used to analyze evolution of the genomes of multiple cancer types [34, 33, 36, 24, 44]. However, the CND does not adequately model WGD and polyploidzation events, which are frequent in cancer.

In this paper, we formulate the genome halving and genome aliquoting problems for CNPs under the CND model and give polynomial time algorithms for both problems. Similar to other rearrangement models, we define the halving and aliquoting distances under the CND model as the minimum CND from some duplicated CNP to a given extant CNP. For the copy number halving distance problem, we give a very simple linear time algorithm for finding a closest preduplicated CNP of an extant CNP. Moreover, we show that we can find a closest preduplicated CNP such that a the CND to the extant profile contains only deletions or a maximum number of amplifications. WGD followed by massive loss of genes is commonly known in evolution and thus finding a preduplication profile having such that after WGD there are only deletions is biologically reasonable [20]. For the copy number aliquoting problem we give a quadratic time dynamic programming algorithm. To this end we show that there exists a preduplicated CNP where each position in the profile is either amplified or deleted, and the number of new operations starting at each position is bounded. Furthermore, we show that each row in the dynamic programming table is a non decreasing function, thus enabling the calculation of each entry in constant time. We implement our algorithms and show on simulated data they are able to reconstruct a preduplication profile.

2 Preliminaries

In this section we present the CND model (Section 2.1) and formulate the halving and aliquoting problem under this model (Section 2.2).

2.1 Copy number profiles and distance

A *copy number profile* (*CNP*) $V = \langle v_1, \ldots, v_n \rangle$ is a vector of non-negative integers. We refer to each coordinate in a copy number profile as a *gene* although more generally these entries correspond to genomic segments or synteny blocks. Each entry of a copy number profile gives the number of copies of the gene in the genome. For example, $V = \langle 0, 10, 15, 30 \rangle$ is a CNP with four genes, where the second gene of V has 10 copies.

For integers i, j with $i \leq j$ let $[i, j] = [i, i+1, i+2, \ldots, j]$ denote the interval of integers from i to j. A *copy number operation* (CNO) is a triple (ℓ, h, w) where $\ell, h \in [1, n]$ denote two genes, and $w \in \{-1, 1\}$ denotes whether the CNO is a *deletion* or an *amplification*, respectively. We call ℓ and h the start and end genes (inclusive) of the contiguous segment $[\ell, h]$ of the CNP onto which the CNO is applied. When a CNO $c = (\ell, h, w)$ is *applied* to a CNP $V = \langle v_i \rangle_{i=1}^n$, the result is a new CNP $c(V) = U = \langle u_i \rangle_{i=1}^n$ defined as follows:

$$u_i = \begin{cases} v_i & i \notin [\ell, h] \\ 0 & v_i = 0 \text{ and } i \in [\ell, h] \\ v_i + w & v_i > 0 \text{ and } i \in [\ell, h] \end{cases}$$

We say that a gene i is *targeted* by a CNO $c = (\ell, h, w)$ if $\ell \leq i \leq h$.

A *copy number transformation* (CNT) is a vector $C = \langle c_1, c_2, \ldots, c_m \rangle$ of CNOs. We *apply* a CNT C on a CNP U in order, i.e., $C(U) = c_m(c_{m-1}(\ldots c_2(c_1(U)) \ldots))$. The cardinality or size of a CNT is the number of CNOs, and is denoted $|C|$. We say that a CNT C has direction $U \to V$ if $C(U) = V$.

Given two CNPs U and V of n genes, the *copy number distance (CND)* $d(U, V)$ is the smallest integer t such that there exists some CNT C where $|C| = t$ and $C(U) = V$, i.e. t is the minimum number of CNOs required to transform U into V. While we use the term distance for CND, note that the CND is neither symmetric nor satisfies the triangle inequality. We call a CNT C *optimal* for the direction $U \to V$ if $|C| = d(U, V)$ and $C(U) = V$. For example, $d(\langle 2, 1, 0, 1, 2, 2\rangle, \langle 4, 0, 0, 0, 3, 4\rangle) = 3$ and an optimal transformation includes one deletion and two amplifications (Figure 1a). If no CNT of any size exists between U and V, then the distance $d(U, V) = \infty$. For instance, there is no transformation for the reverse direction $\langle 4, 0, 0, 0, 3, 4\rangle \to \langle 2, 1, 0, 1, 2, 2\rangle$ in the previous example (Figure 1a).

In addition to CNOs that increase or decrease the CNs of a CNP by 1, we introduce here a new operation for multiplying a CNP by a scalar. For a CNP S and an integer $p > 1$, we denote by $pS = \langle ps_1, ps_2, \ldots, ps_n\rangle$ a *duplicated CNP* where each gene is multiplied by p. We call S the *preduplicated CNP* of duplicated CNP pS.

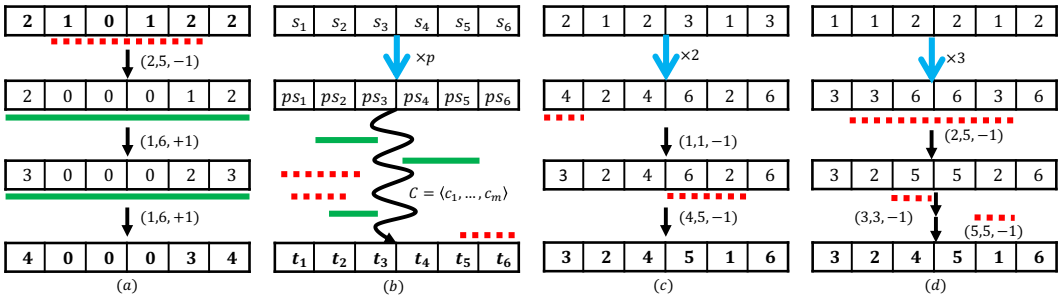


Figure 1 (a) A copy number transformation from $\langle 2, 1, 0, 1, 2, 2\rangle$ to $\langle 4, 0, 0, 0, 3, 4\rangle$ includes one deletion (red dotted line) and two amplifications (green solid lines). (b) Schematic overview of the aliquoting problem. Given a CNP $T = \langle t_1, \ldots, t_n\rangle$ (bold) and integer $p \geq 2$, find a preduplication profile $S = \langle s_1 \ldots s_n\rangle$ that minimizes $d(pS, T)$. (c-d) The halving and aliquoting ($p = 3$) solutions for CNP $\langle 3, 2, 4, 5, 1, 6\rangle$.

2.2 Copy number halving and aliquoting problems

Given a CNP T, we define the *CNP halving distance* $\eta_2(T)$ as the minimum CND between a doubled profile $2S$ and T:

$$\eta_2(T) = \min_S d(2S, T)$$

Similarly, for a CNP T and an integer $p \geq 2$, we define the *CNP aliquoting distance* $\eta_p(T)$ as the minimum CND between a duplicated profile pS and T:

$$\eta_p(T) = \min_S d(pS, T)$$

We say that $\hat{S}$ is an *optimal preduplicated CNP* of an extant CNP T for the halving (aliquoting resp.) problem if $\eta_2(T) = d(2\hat{S}, T)$ ($\eta_p(T) = d(p\hat{S}, T)$ resp.). We formulate the problems of finding an optimal preduplicated CNP as follows (Figure 1b).

▶ **Copy Number Profile Halving Problem.** *Given a CNP T, compute the halving distance $\eta_2(T)$ and find an optimal preduplication profile $\hat{S}$.*

▶ **Copy Number Aliquoting Halving Problem.** *Given a CNP T and an integer p, compute the aliquoting distance $\eta_p(T)$ and find an optimal preduplication profile $\hat{S}$.*

For example, for the CNP $T = \langle 3, 2, 4, 5, 1, 6\rangle$ the halving distance is $\eta_2(T) = 2$ using a preduplication profile $\hat{S} = \langle 2, 1, 2, 3, 1, 1\rangle$ (Figure 1c) whereas the aliquoting distance is $\eta_3(T) = 3$ using a preduplication profile $\hat{S} = \langle 1, 1, 2, 2, 1, 2\rangle$ (Figure 1d). Therefore under parsimony assumption, T is more likely to have originated from whole genome duplication than triplication.

3 Algorithms

In this section we give algorithms for the CNP halving and aliquoting problems. We begin by showing several properties that enable us to reduce the problem size and limit the search space of possible preduplication profiles (Section 3.1). Then we give a simple linear time algorithm for the halving problem in Section 3.2 and a quadratic dynamic programming algorithm for the aliquoting problem in Section 3.3.

3.1 Properties of aliquoting solutions

Here, we show a few properties that will simplify the halving and aliquoting problems. We first show that we may assume without loss of generality that the input CNP T has no zeroes and that T has no two adjacent genes that are congruent mod p. Therefore we can preprocess an T to remove such positions and reduce the profile size.

We start by showing we can remove genes with zero copy number without changing the halving/aliquoting distance. For brevity, we refer the reader to Appendix Section S1.1.1 for the full proof.

▶ **Proposition 1.** *Let $T = \langle t_1, t_2, \ldots, t_{i-1}, 0, t_{i+1}, \ldots, t_n\rangle$ be a profile with $t_i = 0$ and let $T' = \langle t_1, t_2, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n\rangle$ be the profile with gene i removed. Then, $\eta_p(T) = \eta_p(T')$.*

Next, we show that we can assume that no two consecutive genes in T have the same value modulo p. We rely on the following observation proved in [46]. The full proof of Proposition 2 is omitted and given in Appendix Section S1.1.2.

▶ **Observation 1.** *Let S and T be profiles whose entries are strictly positive and let C be a CNT from $S \to T$. Let a_i be the number of amplification events in C that target gene i, and let d_i be the number of deletion events that target gene i. Then, $t_i = s_i + a_i - d_i$.*

▶ **Proposition 2.** *Let $T = \langle t_1, t_2, \ldots, t_n\rangle$ be a CNP with $t_i \equiv t_{i+1} \bmod p$ and $t_i, t_{i+1} \neq 0$. Without loss of generality, assume that $t_i \leq t_{i+1}$. Let $T' = \langle t_1, t_2, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n\rangle$ be the CNP obtained by removing gene i from T. Then, $\eta_p(T) = \eta_p(T')$.*

By applying Propositions 1 and 2 repeatedly on T we obtain a shorter CNP T' such that $\eta_p(T) = \eta_p(T')$. Moreover, the proofs of Propositions 1 and 2 are also constructive, enabling us to obtain a preduplication profile for T given a preduplication profile for T'. Therefore, we can now assume without loss of generality that for the input CNT T, $t_i > 0$ and $t_i \not\equiv t_{i+1} \bmod p$ for all i.

We now turn to showing properties of optimal preduplication profiles and transformations that will help us analyze the problems and reduce the search space. We say the CNT C for $U \to V$ is *disjoint* if no gene is both amplified and deleted. We first show that there exists an optimal disjoint transformation for $\eta_p(T)$. This reduces the number of solutions we need to consider.

▶ **Lemma 1.** *For any optimal preduplication profile $\hat{S}$ there exists an optimal transformation C for $p\hat{S} \to T$ such that C is disjoint.*

Proof. As noted above, we assume that $\hat{S}, T$ have strictly positive values. We prove the claim by induction on the number of pairs of amplifications and deletions that overlap. In the base case, we show that one such pair can be replaced by two operations that do not overlap. Let $(i, j, 1)$ be an amplification event, and let $(k, \ell, -1)$ be a deletion event, such that $a = [i, j]$ and $d = [k, \ell]$ intersect.

(a) If $a \subseteq d$, then we can replace the pair with two deletions, where each targets one of the two (contiguous) segments of $d \setminus a$ (Figure 2a). If $d \subseteq a$, vice versa. One or both of the subsequent segments may be empty; we can delete those operations because they do not affect any gene.

(b) If $a \not\subseteq d$, then replace the pair with one amplification targeting $a \setminus d$ and one deletion targeting $d \setminus a$ (Figure 2b).

Since we assume $\hat{S}, T$ are strictly positive, by Observation 1 we only require that the number of amplifications minus the number of deletions stays the same at each gene. Hence, validity is preserved for all genes, and we have eliminated a pair of overlapping amplification and deletion events.

For the inductive case, we can apply the same modification to eliminate a pair of overlapping amplification and deletion events. To complete this case, it only remains to show that we decreased the number of such overlapping pairs. This is easy to see because each new operation targets a segment which is a subset of the segment of some operation of the same type that it replaced, so any overlapping pairs after the modification would have been overlapping pairs before the modification. ◀

(a) One segment is a subset of the other.

(b) Neither segment is a subset of the other.

Figure 2 Modifying pairs of overlapping operations to obtain a disjoint transformation.

In conjunction with Observation 1, Lemma 1 implies that given t_i, the value of $\hat{s}_i$ determines how many events (either all amplifications or all deletions) target gene i in some optimal transformation. Conversely, the number of amplifications or deletions that affect each i uniquely determines the preduplication profile s_i. So, it suffices to find an optimal preduplication profile for $\eta_p(T)$, which induces an optimal disjoint transformation.

Finally, we bound the number of events in affecting each gene in any optimal disjoint transformation for $\eta_p(T)$, which also bounds how many preduplication profiles we need to consider to find an optimal one.

▶ **Lemma 2.** *For all CNPs T and $p \geq 2$, $\eta_p(T) \leq np$.*

Proof. Pick $S = \lceil T/p \rceil$, which may not be optimal in general. Then, for each of the n genes, we will need $p\lceil t_i/p \rceil - t_i \leq p$ deletion events. Assuming for an upper bound that each event targets exactly gene i, we can build a transformation for $pS \to T$ with $\leq np$ events. ◀

▶ **Corollary 1.** *For any CNP T and integer $p \geq 2$, there is an optimal preduplication profile with an optimal disjoint transformation in which every gene is affected by at most np deletions or at most np amplifications.*

3.2 CNP halving

In this section, we derive a simple algorithm for the CNP halving problem. We note that some cases of CNP halving are easy. For instance, if every value of T is even, then the CNP halving distance is zero because $\hat{S} = T/2$ is an optimal preduplication profile, and we need no CNOs at all since $2\hat{S} = T$. On the other hand if every value of T is odd, then the CNP halving distance is always 1 by setting $\hat{S} = \lceil T/2 \rceil = (T+1)/2$ as a preduplication profile and applying one CNO $(1, n, -1)$, which decrements by one. We will show here how to generalize this result to CNPs that contain both even and odd numbers.

To derive an algorithm for CNP halving, we make a few observations. We define an *odd run* in a CNP as a maximal-length contiguous segment of genes such that all of the gene values are odd. Similarly, an *even run* of a CNP is a maximal-length contiguous segment of genes such that all of the gene values are even. For example, in the CNP $\langle 1, 2, 4, 3, 5 \rangle$ there are two odd runs, $\langle 1 \rangle$ and $\langle 3, 5 \rangle$, and one even run $\langle 2, 4 \rangle$. We denote by $\text{odd}(V)$ ($\text{even}(V)$) the number of odd (even resp.) runs in a CNP V. We first show in the following proposition how each CNO affects the number of odd runs in a profile.

▶ **Proposition 3.** *Let V be a CNP and let $c = (\ell, h, w)$ be a CNO such that $\forall i, c(V)_i \geq 1$. Then,* $\text{odd}(c(V)) - \text{odd}(V) \leq 1$.

Proof. Denote by $\Delta_o = \text{odd}(c(V)) - \text{odd}(V)$ and $\Delta_e = \text{even}(c(V)) - \text{even}(V)$, the difference in odd and even runs respectively between $c(V)$ and V. Notice that amplifications and deletions affect the parity of each value in the CNP in the same way and therefore our proof is invariant to the operation type. We first observe that for any run $[i, j]$ fully contained within the target segment $[\ell, h]$, the parity of the run in $c(V)$ is toggled. Thus a fully contained odd run becomes and even run and vice versa. This enables us to separate our analysis into two cases: (a) operations that start and end in runs with the same parity, and (b) operations that start and end in runs with opposite parities (Figure 3ab). We divide each such case into five sub-cases: (I) the start and end of the operation are strictly within a run, (II) one side of the operation is bordering the next run and the other not, (III) one side of the operation is bordering the next run and the other is bordering the end of the profile, (IV) both sides of the operation are bordering the ends of the profile, and (V) both sides of the operation are bordering adjacent runs (Figure 3I-V).

We see that in all cases analyzed, both $\Delta_o \leq 1$ and $\Delta_e \leq 1$ (Figure 3). Notice that although all operations in case (a) affect only even runs, the complement operations that affect only odd runs are symmetrical by replacing each odd run with an even run. We conclude that each operation can increase the number of odd runs in a profile by at most 1. ◀

We now use this property to solve the CNP halving problem in linear time.

▶ **Theorem 1.** *$\eta_2(T) = \text{odd}(T)$ and $\hat{S} = \lceil T/2 \rceil$ is an optimal preduplicated CNP for a profile T.*

Proof. First, we show that $\eta_2(T) \leq d(2\hat{S}, T) \leq \text{odd}(T)$. Note that $2\hat{s}_i = t_i$ if and only if t_i is even, so we need CNOs to correct the odd genes. We can do this with $\text{odd}(T)$ deletions, one for each odd run. Each deletion decrements the genes in one odd run, and the deletions target pairwise disjoint segments because each odd run is a maximal segment. Hence, $\eta_2(T) \leq d(2\hat{S}, T) \leq \text{odd}(T)$.

Next, we show that $\text{odd}(T) \leq \eta_2(T)$. Any duplicated profile $2S$ has no odd runs, while T has $\text{odd}(T)$ odd runs. On the hand, by Proposition 3, each CNO can increase the number of odd runs in a profile by at most 1. Therefore, it takes at least $\text{odd}(T)$ CNOs to transform a doubled profile $2S$ into T showing that $d(2S, T) \geq \text{odd}(T)$ for any CNP S. Specifically we have $\eta_2(T) = \min_S d(2S, T) \geq \text{odd}(T)$. ◀

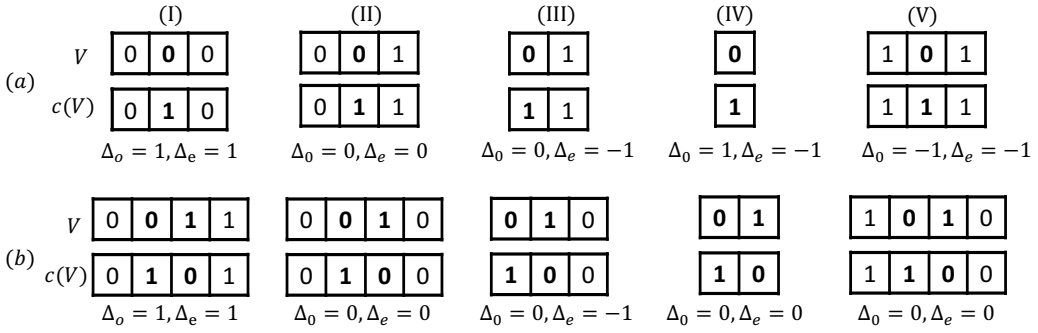


Figure 3 The affect of a CNO c on a CNP V in terms of the number of odd ($\Delta_o = \text{odd}(c(V)) - \text{odd}(V)$) and even ($\Delta_e = \text{even}(c(V)) - \text{even}(V)$) runs. Values in each profile represent the parity of the CN and the affected segment is marked in bold. Horizontal partition – operations that start and end in runs with (a) the same parity, (b) opposite parities. Vertical partition – the start and end of the operation are (I) strictly within a run, (II) one side bordering the next run and the other not, (III) one side bordering the next run and the other bordering the end of the profile, (IV) both sides bordering the ends of the profile, (V) both sides bordering adjacent runs (Figure 3(I-V)).

Notice that an optimal preduplication profile for the halving problem is not unique. For example, $\hat{S} = \langle 1 \rangle$ and $\hat{S} = \langle 2 \rangle$ are both optimal preduplication profiles for $T = \langle 3 \rangle$. One way to distinguish between optimal preduplication profiles is to look at the transformation they induce to the extant profile. For instance, Theorem 1 gives us the following corollary:

▶ **Corollary 2.** *$\hat{S} = \lceil T/2 \rceil$ is an optimal preduplicated CNP for a profile T and the transformation $\hat{S} \to T$ uses only deletions.*

On the other hand, in the following proposition we show how to select an optimal preduplication profile such that the transformation will use a maximum number of amplifications.

▶ **Proposition 4.** *The maximum number of amplifications in a transformation from an optimal duplicated genome $2\hat{S}$ to T is $\text{odd}(T)$ if there is no $t_i = 1$ and $\text{odd}(T) - 1$ if there is some $t_i = 1$.*

Proof. First, suppose there is no i such that $t_i = 1$. In this case we use $\hat{S} = \lfloor T/2 \rfloor$. We define a transformation $\hat{S} \to T$ that uses $\text{odd}(T)$ amplifications by applying one amplification on every odd run. For every even t_i value we have that $2\hat{s}_i = t_i$ and therefore this genes do not need to be modified. On the other hand, for odd t_i values we have that $2\hat{s}_i = t_i - 1$ and therefore one amplification on every odd runs transform $2\hat{S}$ into T.

Conversely, suppose there is a gene i having $t_i = 1$. For any CNP S, a duplicated CNP $2S$ has only values greater or equal to 2. Hence any transformation from a duplicated profile $2S$ to an extant profile T containing a value one must use at least one deletion. We define $\hat{S} = \lceil T/2 \rceil$ and show how to construct a transformation $\hat{S} \to T$ that uses one deletion and $\text{odd}(T) - 1$ amplifications. Let $\hat{i}$ be the leftmost gene of the leftmost odd run and let $\hat{j}$ be the rightmost gene of the rightmost odd run. We apply one deletion $(\hat{i}, \hat{j}, -1)$ which adjusts every odd value to its value in T. However, now we need to adjust the even runs in $[\hat{i}, \hat{j}]$. We do so by applying one amplification on each even run of T in $[\hat{i}, \hat{j}]$. Since $[\hat{i}, \hat{j}]$ covers all odd runs, there are $\text{odd}(T) - 1$ even runs in that segment. ◀

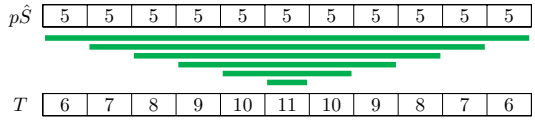


Figure 4 A non-trivial example of the aliquoting distance $\eta_5(T)$ with an optimal preduplication profile $\hat{S} = \langle 1, 1, \ldots, 1\rangle$. For genes 5, 6 and 7 (with copy numbers 10, 11, and 10 in T, respectively), $\hat{s}_i \notin \{\lceil t_i/p \rceil, \lfloor t_i/p \rfloor\}$.

3.3 CNP aliquoting

In this section we derive an algorithm for the CNP aliquoting problem. While CNP halving is equivalent to CNP aliquoting with $p = 2$, generalizing the solution to CNP halving for $p > 2$ by rounding T/p up or down does not work with CNP aliquoting. Namely, there are instances T, p for the aliquoting problem where the genes of an optimal preduplication profile are not necessarily $\lceil t_i/p \rceil$ nor $\lfloor t_i/p \rfloor$. Moreover, even for genes where $t_i \equiv 0 \mod p$, the optimal preduplication may not contain t_i/p in the i'th gene. For example, let $p = 5$ and consider the following "triangle" CNP $T = \langle 6, 7, 8, 9, 10, 11, 10, 9, 8, 7, 6\rangle$ (Figure 4). Using a preduplication profile $\lceil T/p \rceil = \langle 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2\rangle$, the CN distance $d(p\lceil T/p \rceil, T)$ is 12. Similarly, using a preduplication profile $\lfloor T/p \rfloor = \langle 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1\rangle$ the CN distance $d(p\lfloor T/p \rfloor, T)$ is 9. On the other hand, if we use the preduplication profile $\hat{S} = \langle 1, 1, \ldots, 1\rangle$, we have that $d(p\hat{S}, T) = 6$ using a "triangle" of amplification CNOs (Figure 4):

$$(1, 11, +1),\ (2, 10, +1),\ (3, 9, +1),\ \ldots,\ (6, 6, +1)$$

Each successive amplification targets two fewer genes than the previous amplification. For this example $\eta_p(T) \leq 6$, which is also the best possible distance (proof omitted) and therefore $\hat{S}$ is an optimal preduplication profile.

We begin by proposing a simple dynamic programming algorithm (Section 3.3.1). Then, we refine it with additional observations that enables us to improve the run time (Section 3.3.2).

3.3.1 An $O(n^3)$ dynamic programming algorithm

As we showed in Corollary 1, we can limit our search to finding an optimal preduplication profile $\hat{S}$ where we have $\leq 2n$ choices for the value of each s_i. We explicitly enumerate these choices:

- We use $\leq np$ deletions at gene i to reach t_i from $\hat{s}_i$. In this case, $p\hat{s}_i \geq t_i$ and $p\hat{s}_i - t_i \leq np$. Let b_i^- be the "base" number of deletions, i.e., the minimum number of deletions needed to reach t_i from any choice of $p\hat{s}_i$.

  $$b_i^- = p - t_i \bmod p = -t_i \bmod p$$

  This equation holds because $p\hat{s}_i$ must be a multiple of p, and so we calculate the minimum number of deletions we need to reach t_i from any multiple of p. Now, gene i could be subject to either b_i^- deletions, $b_i^- + p$ deletions, etc. until we reach our bound of $\leq np$ deletions. Each of these choices for number of deletions corresponds to a unique value for $\hat{s}_i$. We denote the set of possible number of deletion of gene i as:

  $$\mathcal{D}_i = \{b_i^- + kp \mid k \geq 0 \wedge b_i^- + kp \leq np\} = \{b_i^-,\ b_i^- + p,\ \ldots,\ b_i^- + (n-1)p\}$$

  Notice that $|\mathcal{D}_i| = n$.

- Alternatively, we use $\leq np$ amplifications at gene i to reach t_i. However, unlike the deletion case where we could always increase $\hat{s}_i$ in order to accommodate more deletions at a gene, for amplifications, we must decrease $\hat{s}_i$ to add more amplifications. At the same time, we must have $\hat{s}_i \geq 1$ since $t_i \neq 0$. This bounds the maximum number of amplifications we can have at each gene. For example, if $t_i < p$ then we cannot reach t_i using amplifications.
  Similar to deletions, we can also define a base number of amplification, corresponding to the minimum number of amplifications required to reach t_i for some choice of $\hat{s}_i$:

  $$b_i^+ = t_i \bmod p$$

  In addition, we define the set of choices for the number of amplifications of gene i as follows:

  $$\mathcal{A}_i = \{b_i^+ + kp \mid k \geq 0 \wedge b_i^+ + kp \leq np \wedge t_i - b_i^+ - kp \geq p\}$$

  Note that $p\hat{s}_i \geq p$ since $\hat{s}_i \geq 1$, which is where we derive the additional condition in the set. Therefore, $|\mathcal{A}_i| = \min\{n, \lfloor t_i/p \rfloor\}$ and in the case $t_i < p$ we have $\mathcal{A}_i = \varnothing$.

We define the following dynamic programming tables. For every $1 \leq i \leq n$ and every $x \in \mathcal{D}_i$, $D[i, x]$ will hold $\min_S d(pS, \langle t_1, \ldots, t_i \rangle)$ such that $ps_i - x = t_i$, i.e there is a disjoint transformation that applies exactly x deletions on the i'th gene. Similarly, for every $1 \leq i \leq n$ and every $x \in \mathcal{A}_i$, $A[i, x]$ will hold $\min_S d(pS, \langle t_1, \ldots, t_i \rangle)$ such that $ps_i + x = t_i$, i.e. there is a disjoint transformation that applies exactly x amplifications on the i'th gene. There are at most $2n^2$ values in the arrays A and D put together, because there are $\leq n$ choices for x in each table and n for each coordinate i, each corresponds to a value for s_i. We now show how to calculate each value in the tables in $O(n)$ time, using the values for the previous gene. For completeness, we initialize the tables using $D[0, 0] = A[0, 0] = 0$.

▶ **Theorem 2.** *The following hold:*

$$D[i, x] = \min \left\{ \min_{y \in \mathcal{D}_{i-1}} \{D[i-1, y] + \max\{0, x - y\}\}, \min_{y \in \mathcal{A}_{i-1}} \{A[i-1, y] + x\} \right\} \tag{1}$$

$$A[i, x] = \min \left\{ \min_{y \in \mathcal{A}_{i-1}} \{A[i-1, y] + \max\{0, x - y\}\}, \min_{y \in \mathcal{D}_{i-1}} \{D[i-1, y] + x\} \right\} \tag{2}$$

Proof. We prove the result for $D[i, x]$; an analogous proof works for $A[i, x]$. We show our result by induction on i. For an empty CNP the property holds by our initialization $D[0, 0] = A[0, 0] = 0$. Assume now that the theorem holds up to $i - 1$.

First, we show that $D[i, x] \leq \text{RHS}(1)$, the right hand side of Equation 1:

$$\min_{y \in \mathcal{D}_{i-1}} \{D[i-1, y] + \max\{0, x - y\}\}, \min_{y \in \mathcal{A}_{i-1}} \{A[i-1, y] + x\} \Big\}.$$

This is because we can assemble a solution for $D[i, x]$ using the following three cases (Figure 5):

(a) We select $y \in \mathcal{D}_{i-1}$ such that $y \geq x$ and look at the transformation corresponding to $D[i-1, y]$. In this case, we do not need to add any new operations, because we can pick x arbitrary deletions that target coordinate $i - 1$ and extend them to also target coordinate i (Figure 5a). Hence,

$$D[i, x] \leq \min_{\substack{y \in \mathcal{D}_{i-1} \\ y \geq x}} \{D[i-1, y]\}$$

(b) We select $y \in \mathcal{D}_{i-1}$ such that $y \leq x$ and look at the transformation corresponding to $D[i-1, y]$. Then, we extend all of the deletions at $i-1$ to also affect coordinate i, and add $x - y$ new deletions at coordinate i (Figure 5b). We have,

$$D[i, x] \leq \min_{\substack{y \in \mathcal{D}_{i-1} \\ y \leq x}} \{D[i-1, y] + (x - y)\}$$

(c) Finally, we select $y \in \mathcal{A}_{i-1}$ and look at the transformation corresponding to $A[i-1, y]$. We always need to introduce x new deletions because we cannot make use of any of the existing operations at $i-1$ since they are all amplifications and the transformation we are looking for is disjoint (Figure 5c). Therefore,

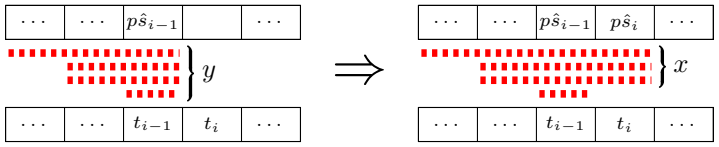
$$D[i, x] \leq \min_{y \in \mathcal{A}_{i-1}} \{A[i-1, y] + x\}$$

Taking the min of all three cases gives us RHS(1).

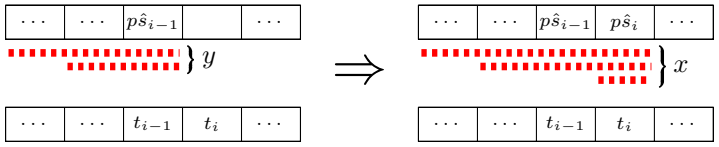
To complete the proof, suppose for contradiction that $D[i, x] < \text{RHS}(1)$. Then, there exists some optimal transformation for the i'th prefix where coordinate i is affected by x deletions and coordinate $i-1$ is affected by y^* operations (either amplifications or deletions).

- If $i-1$ is affected by y^* amplifications, then removing the x deletions at i gives a copy number transformation for the $(i-1)$th prefix with y^* amplifications at coordinate $i-1$. But, $D[i, x] - x < \min_y\{A[i-1, y] + x\} - x = \min_y\{A[i-1, y]\}$ which is a contradiction.
- Similarly, if $i-1$ is affected by y^* deletions, we can remove either zero or $x - y^*$ deletions to get a copy number transformation for the $(i-1)$th prefix with y^* deletions at coordinate $i-1$ that is better than $D[i-1, y^*]$, which is a contradiction.

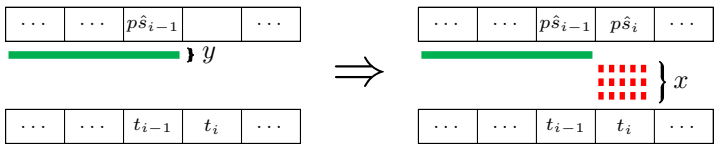
In all cases we get a contradiction, which completes the proof. ◀



(a) $\forall y \in \mathcal{D}_{i-1}$ such that $y \geq x$, extend x of the existing deletions to gene i.



(b) $\forall y \in \mathcal{D}_{i-1}$ such that $y \leq x$, extend all y deletions to i and add $x - y$ new deletions starting at i.



(c) $\forall y \in \mathcal{A}_{i-1}$, start x new deletions at i.

Figure 5 Illustration of different cases in the dynamic programming algorithm to compute $D[i, x]$ (Theorem 2).

Tables D and A can be populated in time $O(n^3)$, by trying each of the $O(n)$ values for $D[i-1, y]$ and $A[i-1, y]$ in order to calculate $D[i, x]$ or $A[i, x]$. Once all the values are calculated, we can find the aliquoting distance $\eta_p(T) = \min\left\{\min_x\{D[n, x]\}, \min_x\{A[n, x]\}\right\}$,

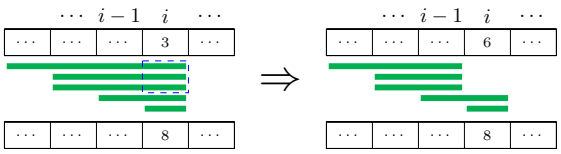


Figure 6 Trimming $p = 3$ events to show that $A[i, x]$ is non-decreasing in x.

which takes time $O(n)$. We also record the argmin that we use to populate each entry of D and A, which allows us to backtrack in order to compute the optimal pre-duplication profile. Hence, the entire algorithm works in time $O(n^3)$ and $O(n^2)$ space . Notice however that due to Propositions 1 and 2, the length of the profile n that we solve the problem for can be smaller than the original input profile.

3.3.2 An improved $O(n^2)$ dynamic programming algorithm

Here, we show that we only need to check $O(1)$ possibilities to calculate each entry of D and A, which reduces the run time of the algorithm to $O(n^2)$, the number of entries in both tables. First, we note that $D[i, x]$ and $A[i, x]$ are non-decreasing as we increase the number of operations x.

▶ **Lemma 3.** *If $\{x, x + p\} \subseteq \mathcal{D}_i$, then $D[i, x + p] \geq D[i, x]$ (similarly, if $\{x, x + p\} \subseteq \mathcal{A}_i$, $A[i, x + p] \geq A[i, x]$).*

Proof. We prove the statement for A (a similar proof works for D). Suppose that $\{x, x+p\} \subseteq \mathcal{A}_i$. Then, we show we can take a disjoint transformation for the i'th prefix that has $A[i, x+p]$ events, including $x+p$ amplifications at gene i, and modify it to get a transformation that has $\leq A[i, x + p]$ events but has x amplifications at gene i. This implies that $A[i, x] \leq A[i, x + p]$.

To do so, we can pick p arbitrary amplifications that target gene i, and shrink each of them by decrementing the end index of their segments, so that they no longer target gene i, but that they still target all of the genes other than i that it targeted before (Figure 6). Note that this implies that we increment the preduplication gene $\hat{s}_i$, because we have shown that the number of events at a gene implies the value of the preduplication gene there, and vice versa. In addition, since we are only considering the i'th prefix, this modification preserves the contiguity of every amplification. ◀

Moreover, we now show we can bound the increase in aliquoting distance when we increase the number of deletions/amplifications on a gene.

▶ **Lemma 4.** *If $\{x, x + p\} \subseteq \mathcal{A}_i$, then $A[i, x + p] - A[i, x] \leq p$ (and similarly, $D[i, x + p] - D[i, x] \leq p$).*

Proof. We prove the statement for A (a similar proof works for D). We can always add p new operations to $A[i, x]$ to get a solution for $A[i, x+p]$ if $x+p \in \mathcal{A}_i$, so $A[i, x+p] \leq A[i, x]+p$. ◀

Using these results, we improve the performance of the dynamic programming algorithm. First, at the end of the algorithm we return either $D[n, b_n^-]$ or $A[n, b_n^+]$ (base number of deletions or amplifications) instead of checking each entry in $D[n, \dots]$ and $A[n, \dots]$, since larger number of operations at coordinate n will have at least as large aliquoting distances. However, this does not improve the overall asymptotic time complexity of the algorithm. To achieve our improved time complexity, we show that we only need to try $O(1)$ possibilities

to calculate each entry in D and A. To accomplish this, we prove that the minimum of $O(n)$ y values in RHS(1) and RHS(2) in Theorem 2 can be expressed as the minimum of $O(1)$ values. To that end we define the following functions for every i and x:

$$y_1(x) \coloneqq \min\{y \in \mathcal{D}_{i-1} \mid y \geq x\}; \qquad y_2(x) \coloneqq \max\{y \in \mathcal{D}_{i-1} \mid y \leq x\};$$
$$y'_1(x) \coloneqq \min\{y \in \mathcal{A}_{i-1} \mid y \geq x\}; \qquad y'_2(x) \coloneqq \max\{y \in \mathcal{A}_{i-1} \mid y \leq x\};$$

▶ **Theorem 3.** *The following hold:*

$$D[i,x] = \min\{D[i-1, y_1(x)], D[i-1, y_2(x)] + (x - y_2(x)), A[i-1, b^+_{i-1}] + x\}$$
$$A[i,x] = \min\{A[i-1, y'_1(x)], A[i-1, y'_2(x)] + (x - y'_2(x)), D[i-1, b^-_{i-1}] + x\}$$

Proof. We prove the result for $D[i,x]$ with $y_1(x)$ and $y_2(x)$; an analogous proof works for $A[i,x]$ together with $y'_1(x)$ and $y'_2(x)$. We rearrange equation (1) in Theorem 2 as follows:

$$\begin{aligned} D[i,x] &= \min\left\{ \min_{y \in \mathcal{D}_{i-1}} \{D[i-1,y] + \max\{0, x-y\}\}, \min_{y \in \mathcal{A}_{i-1}} \{A[i-1,y] + x\} \right\} \\ &= \min\left\{ \min_{\substack{y \in \mathcal{D}_{i-1} \\ y \geq x}} \{D[i-1,y]\}, \min_{\substack{y \in \mathcal{D}_{i-1} \\ y \leq x}} \{D[i-1,y] + (x-y)\}, \min_{y \in \mathcal{A}_{i-1}} \{A[i-1,y] + x\} \right\} \end{aligned} \tag{3}$$

Now, we show that each of the three inner min expressions in equation (3) can be replaced with a single term (Figure 7).

(a) If there exists at least one $y \in \mathcal{D}_{i-1}$ such that $y \geq x$ (Figure 7a), then

$$\min_{\substack{y \in \mathcal{D}_{i-1} \\ y \geq x}} \{D[i-1,y]\} = D[i-1, y_1(x)]$$

This is because the non-decreasing property from Lemma 3 implies that we can check the entry for the minimum y to get the smallest value. So, we can replace the first min terms in equation (3) with $D[i-1, y_1(x)]$.

(b) If there exists $y \in \mathcal{D}_{i-1}$ such that $y \leq x$ (Figure 7b), then

$$\min_{\substack{y \in \mathcal{D}_{i-1} \\ y \leq x}} \{D[i-1,y] + (x-y)\} = D[i-1, y_2] + (x - y_2) \tag{4}$$

Let $y_2 = y_2(x)$ for conciseness. Suppose by contradiction there is some other value in $\mathcal{D}_{i-1}$ (which we can express as $y_2 - kp$ for $k \geq 1$) minimizes equation (4): $D[i-1, y_2 - kp] + (x - (y_2 - kp)) < D[i-1, y_2] + (x - y_2)$. Rearranging, we get

$$D[i-1, y_2] > D[i-1, y_2 - kp] + kp$$

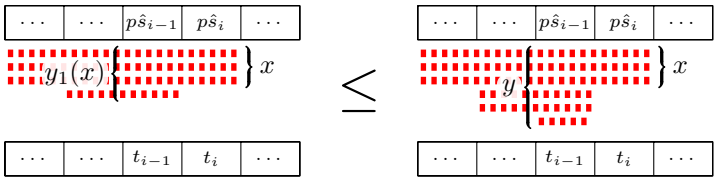
which contradicts Lemma 4. Hence, we can replace the second min terms in equation (3) with $D[i-1, y_2] + (x - y_2)$.

(c) Finally, the following equation hold (Figure 7c):

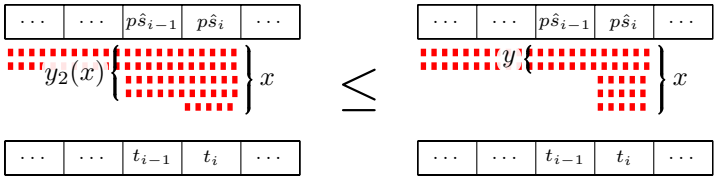
$$\min_{y \in \mathcal{A}_{i-1}} \{A[i-1,y] + x\} = \min_{y \in \mathcal{A}_{i-1}} \{A[i-1,y]\} + x = A[i-1, b^+_{i-1}] + x$$

This follows from Lemma 3; we pick the smallest y to get some min of $A[i-1, \cdot]$. So, we can replace the third min terms in equation (3) with $A[i-1, b^+_{i-1}] + x$. ◀

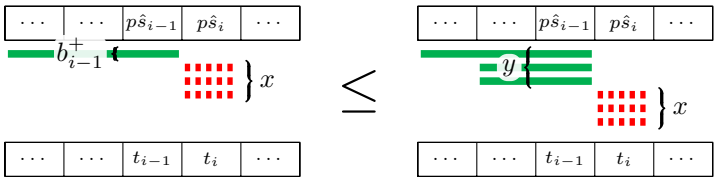
Since y_1, y_1', y_2, y_2' are computable in $O(1)$ time (Supplemental Proposition S5), we can leverage Theorem 3 to populate each table entry in $O(1)$ time, because we can figure out which three values we need to check using y_j, y_j' and take the minimum of these values. In conclusion, we can populate D and A in $O(n^2)$ time $O(n^2)$ space. However, if we are just interested in calculating the aliquoting distance without finding an optimal preduplication profile, we can use only $O(n)$ space since we simply need $D[i-1,\cdot]$ and $A[i-1,\cdot]$ to compute $D[i,\cdot]$ and $A[i,\cdot]$.



(a) $\forall y \in \mathcal{D}_{i-1}$ such that $y \geq x$, since we extend x of the existing deletions, Lemma 3 shows us we should keep y to minimum (i.e. $y_1(x)$).



(b) $\forall y \in \mathcal{D}_{i-1}$ such that $y \leq x$, since we add $x - y$ new deletions, y should be as large as possible (i.e. $y_2(x)$) to minimize the number of additional deletions due to Lemma 4.



(c) $\forall y \in \mathcal{A}_{i-1}$, since we start x new deletions, we should pick the smallest y possible (i.e. b^+_{i-1}) due to Lemma 3.

Figure 7 The cases for computing $D[i,x]$ for the improved dynamic programming algorithm (Theorem 3).

4 Experiments

We evaluated our halving and aliquoting algorithms on simulated CNPs in order to assess their ability to recover preduplication profiles. In each simulation, we generate a random preduplication profile $S \in \{1,\ldots,5\}^n$, multiply each entry by p and then apply k amplifications and deletions to create an extant profile T. We then use the aliquoting algorithm on T to estimate the aliquoting distance $\eta_p(T)$ and find a preduplication profile $\hat{S}$. We run simulations for profile lengths $n \in \{100, 200, 300\}$, polyploidity $p \in \{2,3,4\}$ and $k \in \{5,10,15\}$ events after polyploidization. To simulate events, we apply k random CNOs with uniform length and position, and with a ratio of deletions/amplifications of 3 to 1. We also make sure that the profiles pS and T generated have no zeros. We implemented the halving and aliquoting algorithms in Python 3, and we ran the simulations on a Thinkpad T470 computer with an Intel i7-7600U processor running Linux 5.11.10. For each configuration n, p, k, 100 instances were simulated.

We use several metrics to measure the algorithm performance. First, we evaluate $d(S, \hat{S})$ to measure how close the aliquoting preduplication profile is to the actual preduplication profile. However, there may exist multiple duplicated profiles with the same copy number distance from T. Therefore, we also compare $\Delta\eta_p(T) \coloneqq d(pS, T) - \eta_p(T) = d(pS, T) - d(p\hat{S}, T)$, to measure how close the estimated aliquoting distance was from the actual copy number distance between the true duplicated profile pS and T. This is a measure of how accurately we can recover the number of events that have occurred after polyploidization. Finally, we assess the effective run time of our algorithms as we increase the profile sizes.

We find that CNP aliquoting is more accurate than halving (Figures 8, S2 and S2). When aliquoting $p \geq 3$, we have $\Delta\eta_p(T) \leq 2$ and $d(S, \hat{S}) \leq 4$ for every simulated CNP, which suggests that not only are we are able to estimate the number of post-polyploidization events, but we can also recover a very close profile to the original preduplication profile S by solving the aliquoting problem. On the other hand, for halving ($p = 2$), $\Delta\eta_2(T)$ and $d(S, \hat{S})$ are larger, which is likely because there are many different preduplication profiles for halving. For instance, in Corollary 2 we show that there exists a preduplication profile that maximizes the number of deletions in the optimal transformation, but in Proposition 4 we prove that there exists a different preduplication profile that maximizes the number of amplifications. Notice however, that as the length of the profile increases, we do estimate the number of post-duplication events more accurately. This is partially because with larger CNPs, there is a smaller chance for events to cancel one another.

Finally, we measured the running time of our halving and aliquoting algorithms from Theorem 1 and Theorem 3, respectively (Figures 8c, S2c, and S3c). Notice that our implementation of the aliquoting algorithm contracts runs that have the same value modulo p, using Proposition 2. This preprocessing is done in linear time and does not affect the overall worst-case asymptotic time complexity. However, we find that in practice, it improves the run time significantly. Although the aliquoting dynamic programming algorithm is quadratic in the worst case, we see that on simulated profiles, the increase in running time is much lower. This is because the effective profile length for which we solve the problem depends on the number of runs modulo p and not the original length of the profile.

5 Discussion

In this paper, we formulate and solve the genome halving and genome aliquoting problems for CNPs under the CNT model. For the halving distance we derive a simple linear time algorithm and show how to obtain preduplicated genomes having a transformation with maximum number of deletions or amplifications. For the CNP aliquoting distance we derive a quadratic time dynamic programming algorithm by showing several properties of an optimal preduplication profile and carefully analyzing aliquoting sub problems. We further note that with $O(n)$ time preprocessing and postprocessing, the latter algorithm is quadratic in the number of distinct runs modulo p which can be effectively quite lower than the length of the profile, as we show on simulated CNPs. Finally, our simulations show that we are able accurately estimate the number of events post-duplication.

There are several directions for further research. First, for some CNPs there be many optimal preduplication profiles and our algorithm will not distinguish between these solutions, selecting one arbitrarily based on the implementation. It is therefore interesting to further explore the space of optimal preduplication profiles. Similar ambiguity in selecting a preduplication genome arises in other rearrangement distances, and one solution is to use an out-group in order to further constrain the preduplication genome. This modification, called

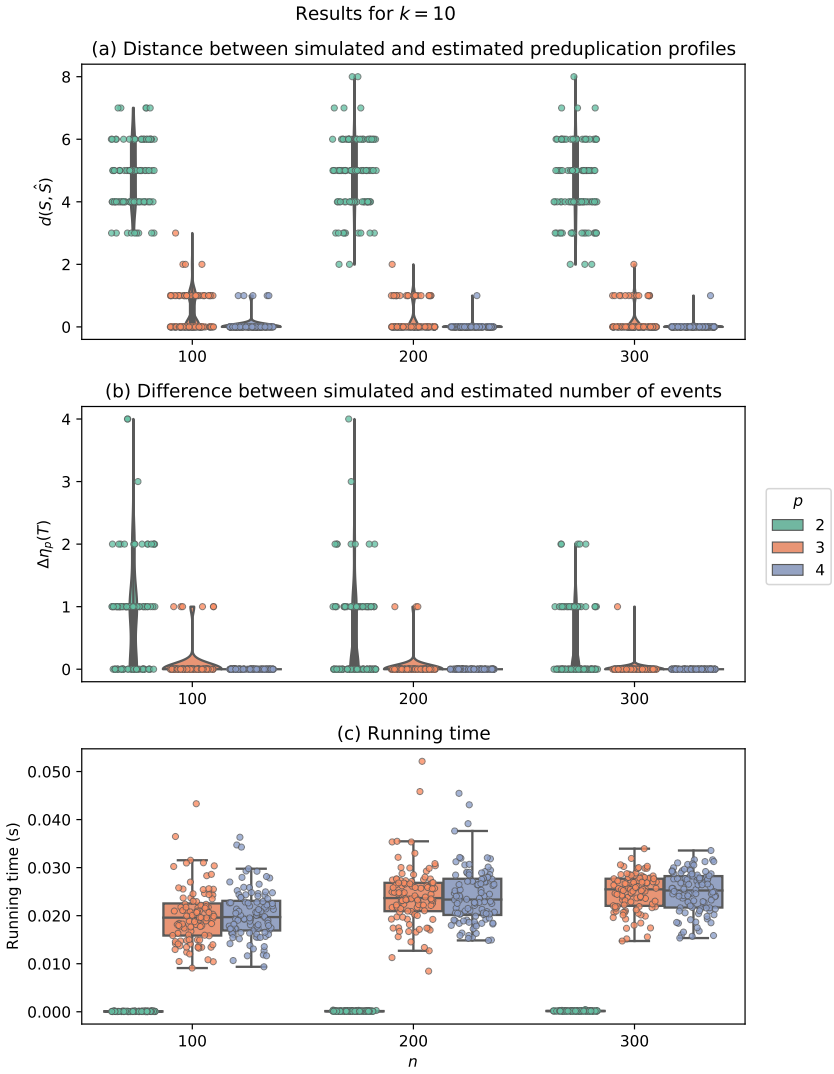


Figure 8 Simulation results for using $k = 10$ events after duplication. (a) $d(S, \hat{S})$ - the distance between simulated and estimated preduplication profiles. (b) $\Delta\eta_p(T)$ - the difference between the simulated and estimated number of events after duplication. (c) running time in seconds.

the *guided genome halving problem*, seeks to find a preduplication genome that minimizes the distance to a given out-group plus the distance from the duplicated genome to the extant genome [47]. An alternative solution which might be more biologically relevant in some cases such as cancer samples is to extend the copy number distance to also include a WGD event [30]. In this case, the copy number distance between a pair of profiles would be the minimum number of amplifications, deletions and WGDs that transform profile into the other. Third, these algorithms could be extended to address the issues of normal cell admixture and subclonality that arise in analyzing cancer sequencing data as has been previously done for CNPs [11, 44]. Finally, applying our algorithms to real cancer genomes with high tumor ploidy could help identify genomes with strong evidence for polyploidization during cancer evolution and provide new insights into highly aneuploid cancer genomes.

References

1 Max A. Alekseyev and Pavel A. Pevzner. Genome halving problem revisited. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, pages 1–15, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

2 Max A Alekseyev and Pavel A Pevzner. Colored de Bruijn graphs and the genome halving problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):98–107, 2007. doi:10.1109/TCBB.2007.1002.

3 Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In Philipp Bücher and Bernard M.E. Moret, editors, *Proc. Workshop on Algorithms in Bioinformatics*, volume 4175 of *LNCS*, pages 163–173. Springer, 2006. doi:10.1007/11851561_16.

4 Craig M Bielski, Ahmet Zehir, Alexander V Penson, Mark TA Donoghue, Walid Chatila, Joshua Armenia, Matthew T Chang, Alison M Schram, Philip Jonsson, Chaitanya Bandlamudi, et al. Genome doubling shapes the evolution and prognosis of advanced cancers. *Nature genetics*, 50(8):1189–1195, 2018.

5 John E Bowers, Brad A Chapman, Junkang Rong, and Andrew H Paterson. Unravelling angiosperm genome evolution by phylogenetic analysis of chromosomal duplication events. *Nature*, 422(6930):433, 2003.

6 Peter J. Campbell, Gad Getz, Jan O. Korbel, Joshua M. Stuart, Jennifer L. Jennings, Lincoln D. Stein, Marc D. Perry, Hardeep K. Nahal-Bose, B. F. Francis Ouellette, Constance H. Li, Esther Rheinbay, G. Petur Nielsen, Dennis C. Sgroi, Chin-Lee Wu, William C. Faquin, Vikram Deshpande, Paul C. Boutros, Alexander J. Lazar, Katherine A. Hoadley, David N. Louis, L. Jonathan Dursi, Christina K. Yung, Matthew H. Bailey, Gordon Saksena, Keiran M. Raine, Ivo Buchhalter, Kortine Kleinheinz, Matthias Schlesner, Junjun Zhang, Wenyi Wang, David A. Wheeler, Li Ding, Jared T. Simpson, Brian D. O'Connor, Sergei Yakneen, Kyle Ellrott, Naoki Miyoshi, Adam P. Butler, Romina Royo, Solomon I. Shorser, Miguel Vazquez, Tobias Rausch, Grace Tiao, Sebastian M. Waszak, Bernardo Rodriguez-Martin, Suyash Shringarpure, Dai-Ying Wu, German M. Demidov, Olivier Delaneau, Shuto Hayashi, Seiya Imoto, Nina Habermann, Ayellet V. Segre, Erik Garrison, Andy Cafferkey, Eva G. Alvarez, JoséMaría Heredia-Genestar, Francesc Muyas, Oliver Drechsel, Alicia L. Bruzos, Javier Temes, Jorge Zamora, Adrian Baez-Ortega, Hyung-Lae Kim, R. Jay Mashl, Kai Ye, Anthony DiBiase, Kuan-lin Huang, Ivica Letunic, Michael D. McLellan, Steven J. Newhouse, Tal Shmaya, Sushant Kumar, David C. Wedge, Mark H. Wright, Venkata D. Yellapantula, Mark Gerstein, Ekta Khurana, Tomas Marques-Bonet, Arcadi Navarro, Carlos D. Bustamante, Reiner Siebert, Hidewaki Nakagawa, Douglas F. Easton, Stephan Ossowski, Jose M. C. Tubio, Francisco M. De La Vega, Xavier Estivill, Denis Yuen, George L. Mihaiescu, Larsson Omberg, Vincent Ferretti, Radhakrishnan Sabarinathan, Oriol Pich, Abel Gonzalez-Perez, Amaro Taylor-Weiner, Matthew W. Fittall, Jonas Demeulemeester, Maxime Tarabichi, Nicola D. Roberts, Peter Van Loo, Isidro Cortés-Ciriano, Lara Urban, Peter Park, Bin Zhu, Esa Pitkänen, Yilong Li, Natalie Saini, Leszek J. Klimczak, Joachim Weischenfeldt, Nikos Sidiropoulos, Ludmil B. Alexandrov, Raquel Rabionet, Georgia Escaramis, Mattia Bosio, Aliaksei Z. Holik, Hana Susak, Aparna Prasad, Serap Erkek, Claudia Calabrese, Benjamin Raeder, Eoghan Harrington, Simon Mayes, Daniel Turner, Sissel Juul, Steven A. Roberts, Lei Song, Roelof Koster, Lisa Mirabello, Xing Hua, Tomas J. Tanskanen, Marta Tojo, Jieming Chen, Lauri A. Aaltonen, Gunnar Rätsch, Roland F. Schwarz, Atul J. Butte, Alvis Brazma, Stephen J. Chanock, Nilanjan Chatterjee, Oliver Stegle, Olivier Harismendy, G. Steven Bova, Dmitry A. Gordenin, David Haan, Lina Sieverling, Lars Feuerbach, Don Chalmers, Yann Joly, Bartha Knoppers, Fruzsina Molnár-Gábor, Mark Phillips, Adrian Thorogood, David Townend, Mary Goldman, Nuno A. Fonseca, Qian Xiang, Brian Craft, Elena Piñeiro-Yáñez, Alfonso Muñoz, Robert Petryszak, Anja Füllgrabe, Fatima Al-Shahrour, Maria Keays, David Haussler, John Weinstein, Wolfgang Huber, Alfonso Valencia, Irene Papatheodorou, Jingchun Zhu, Yu Fan,

David Torrents, Matthias Bieg, Ken Chen, Zechen Chong, Kristian Cibulskis, Roland Eils, Robert S. Fulton, Josep L. Gelpi, Santiago Gonzalez, Ivo G. Gut, Faraz Hach, Michael Heinold, Taobo Hu, Vincent Huang, Barbara Hutter, Natalie Jäger, Jongsun Jung, Yogesh Kumar, Christopher Lalansingh, Ignaty Leshchiner, Dimitri Livitz, Eric Z. Ma, Yosef E. Maruvka, Ana Milovanovic, Morten Muhlig Nielsen, Nagarajan Paramasivam, Jakob Skou Pedersen, Montserrat Puiggròs, S. Cenk Sahinalp, Iman Sarrafi, Chip Stewart, Miranda D. Stobbe, Jeremiah A. Wala, Jiayin Wang, Michael Wendl, Johannes Werner, Zhenggang Wu, Hong Xue, Takafumi N. Yamaguchi, Venkata Yellapantula, Brandi N. Davis-Dusenbery, Robert L. Grossman, Youngwook Kim, Michael C. Heinold, Jonathan Hinton, David R. Jones, Andrew Menzies, Lucy Stebbings, Julian M. Hess, Mara Rosenberg, Andrew J. Dunford, Manaswi Gupta, Marcin Imielinski, Matthew Meyerson, Rameen Beroukhim, Jüri Reimand, Priyanka Dhingra, Francesco Favero, Stefan Dentro, Jeff Wintersinger, Vasilisa Rudneva, Ji Wan Park, Eun Pyo Hong, Seong Gu Heo, André Kahles, Kjong-Van Lehmann, Cameron M. Soulette, Yuichi Shiraishi, Fenglin Liu, Yao He, Deniz Demircioğlu, Natalie R. Davidson, Liliana Greger, Siliang Li, Dongbing Liu, Stefan G. Stark, Fan Zhang, Samirkumar B. Amin, Peter Bailey, Aurélien Chateigner, Milana Frenkel-Morgenstern, Yong Hou, Matthew R. Huska, Helena Kilpinen, Fabien C. Lamaze, Chang Li, Xiaobo Li, Xinyue Li, Xingmin Liu, Maximillian G. Marin, Julia Markowski, Tannistha Nandi, Akinyemi I. Ojesina, Qiang Pan-Hammarström, Peter J. Park, Chandra Sekhar Pedamallu, Hong Su, Patrick Tan, Bin Tean Teh, Jian Wang, Heng Xiong, Chen Ye, Christina Yung, Xiuqing Zhang, Liangtao Zheng, Shida Zhu, Philip Awadalla, Chad J. Creighton, Kui Wu, Huanming Yang, Jonathan Göke, Zemin Zhang, Angela N. Brooks, Matthew W. Fittall, Iñigo Martincorena, Carlota Rubio-Perez, Malene Juul, Steven Schumacher, Ofer Shapira, David Tamborero, Loris Mularoni, Henrik Hornshøj, Jordi Deu-Pons, Ferran Muiños, Johanna Bertl, Qianyun Guo, and The ICGC/TCGA Pan-Cancer Analysis of Whole Genomes Consortium. Pan-cancer analysis of whole genomes. *Nature*, 578(7793):82–93, 2020. doi:10.1038/s41586-020-1969-6.

7 Scott L Carter, Kristian Cibulskis, Elena Helman, Aaron McKenna, Hui Shen, Travis Zack, Peter W Laird, Robert C Onofrio, Wendy Winckler, Barbara A Weir, et al. Absolute quantification of somatic dna alterations in human cancer. *Nature biotechnology*, 30(5):413, 2012.

8 Salim Akhter Chowdhury, Stanley E Shackney, Kerstin Heselmeyer-Haddad, Thomas Ried, Alejandro A Schäffer, and Russell Schwartz. Algorithms to model single gene, single chromosome, and whole genome copy number changes jointly in tumor phylogenetics. *PLoS computational biology*, 10(7):e1003740, 2014.

9 Stefan C. Dentro, Ignaty Leshchiner, Kerstin Haase, Maxime Tarabichi, Jeff Wintersinger, Amit G. Deshwar, Kaixian Yu, Yulia Rubanova, Geoff Macintyre, Jonas Demeulemeester, Ignacio Vázquez-García, Kortine Kleinheinz, Dimitri G. Livitz, Salem Malikic, Nilgun Donmez, Subhajit Sengupta, Pavana Anur, Clemency Jolly, Marek Cmero, Daniel Rosebrock, Steven E. Schumacher, Yu Fan, Matthew Fittall, Ruben M. Drews, Xiaotong Yao, Thomas B. K. Watkins, Juhee Lee, Matthias Schlesner, Hongtu Zhu, David J. Adams, Nicholas McGranahan, Charles Swanton, Gad Getz, Paul C. Boutros, Marcin Imielinski, Rameen Beroukhim, S. Cenk Sahinalp, Yuan Ji, Martin Peifer, Inigo Martincorena, Florian Markowetz, Ville Mustonen, Ke Yuan, Moritz Gerstung, Paul T. Spellman, Wenyi Wang, Quaid D. Morris, David C. Wedge, Peter Van Loo, Stefan C. Dentro, Amit G. Deshwar, Santiago Gonzalez, David J. Adams, Paul C. Boutros, David D. Bowtell, Peter J. Campbell, Shaolong Cao, Elizabeth L. Christie, Yupeng Cun, Kevin J. Dawson, Ruben M. Drews, Roland Eils, Dale W. Garsed, Gavin Ha, Lara Jerman, Henry Lee-Six, Dimitri G. Livitz, Thomas J. Mitchell, Layla Oesper, Myron Peto, Benjamin J. Raphael, S. Cenk Sahinalp, Adriana Salcedo, Steven E. Schumacher, Ruian Shi, Seung Jun Shin, Lincoln D. Stein, Oliver Spiro, Shankar Vembu, David A. Wheeler, Tsun-Po Yang, Quaid D. Morris, Paul T. Spellman, and David C. Wedge. Characterizing genetic intra-tumor heterogeneity across 2,658 human cancer genomes. *Cell*, 2021. doi:10.1016/j.cell.2021.03.009.

10 Li Ding, Timothy J Ley, David E Larson, Christopher A Miller, Daniel C Koboldt, John S Welch, Julie K Ritchey, Margaret A Young, Tamara Lamprecht, Michael D McLellan, Joshua F McMichael, John W Wallis, Charles Lu, Dong Shen, Christopher C Harris, David J Dooling, Robert S Fulton, Lucinda L Fulton, Ken Chen, Heather Schmidt, Joelle Kalicki-Veizer, Vincent J Magrini, Lisa Cook, Sean D McGrath, Tammi L Vickery, Michael C Wendl, Sharon Heath, Mark A Watson, Daniel C Link, Michael H Tomasson, William D Shannon, Jacqueline E Payton, Shashikant Kulkarni, Peter Westervelt, Matthew J Walter, Timothy A Graubert, Elaine R Mardis, Richard K Wilson, and John F DiPersio. Clonal evolution in relapsed acute myeloid leukaemia revealed by whole-genome sequencing. *Nature*, 481(7382):506–10, 2012. doi:10.1038/nature10738.

11 Mohammed El-Kebir, Benjamin J Raphael, Ron Shamir, Roded Sharan, Simone Zaccaria, Meirav Zehavi, and Ron Zeira. Complexity and algorithms for copy-number evolution problems. *Algorithms for Molecular Biology*, 12(1):13, 2017.

12 Nadia El-Mabrouk, Joseph H. Nadeau, and David Sankoff. Genome halving. In Martin Farach-Colton, editor, *Proc. Combinatorial Pattern Matching*, pages 235–250, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

13 Pedro Feijão and Joao Meidanis. SCJ: a breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1318–29, 2011. doi:10.1109/TCBB.2011.34.

14 Guillaume Fertin, Anthony Labarre, Irena Rusu, Stéphane Vialette, and Eric Tannier. *Combinatorics of genome rearrangements*. MIT press, 2009.

15 Andrej Fischer, Ignacio Vázquez-García, Christopher JR Illingworth, and Ville Mustonen. High-definition reconstruction of clonal composition in cancer. *Cell reports*, 7(5):1740–1752, 2014.

16 Moritz Gerstung, Clemency Jolly, Ignaty Leshchiner, Stefan C. Dentro, Santiago Gonzalez, Daniel Rosebrock, Thomas J. Mitchell, Yulia Rubanova, Pavana Anur, Kaixian Yu, Maxime Tarabichi, Amit Deshwar, Jeff Wintersinger, Kortine Kleinheinz, Ignacio Vázquez-García, Kerstin Haase, Lara Jerman, Subhajit Sengupta, Geoff Macintyre, Salem Malikic, Nilgun Donmez, Dimitri G. Livitz, Marek Cmero, Jonas Demeulemeester, Steven Schumacher, Yu Fan, Xiaotong Yao, Juhee Lee, Matthias Schlesner, Paul C. Boutros, David D. Bowtell, Hongtu Zhu, Gad Getz, Marcin Imielinski, Rameen Beroukhim, S. Cenk Sahinalp, Yuan Ji, Martin Peifer, Florian Markowetz, Ville Mustonen, Ke Yuan, Wenyi Wang, Quaid D. Morris, Stefan C. Dentro, Amit G. Deshwar, David J. Adams, Paul C. Boutros, David D. Bowtell, Peter J. Campbell, Shaolong Cao, Elizabeth L. Christie, Yupeng Cun, Kevin J. Dawson, Ruben M. Drews, Roland Eils, Matthew Fittall, Dale W. Garsed, Gavin Ha, Henry Lee-Six, Dimitri G. Livitz, Inigo Martincorena, Thomas J. Mitchell, Layla Oesper, Myron Peto, Benjamin J. Raphael, S. Cenk Sahinalp, Adriana Salcedo, Ruian Shi, Seung Jun Shin, Oliver Spiro, Lincoln D. Stein, Shankar Vembu, David A. Wheeler, Tsun-Po Yang, Quaid D. Morris, Paul T. Spellman, David C. Wedge, Peter Van Loo, Paul T. Spellman, David C. Wedge, PCAWG Evolution & Heterogeneity Working Group, and PCAWG Consortium. The evolutionary history of 2,658 cancers. *Nature*, 578(7793):122–128, 2020. doi:10.1038/s41586-019-1907-7.

17 Gavin Ha, Andrew Roth, Jaswinder Khattra, Julie Ho, Damian Yap, Leah M Prentice, Nataliya Melnyk, Andrew McPherson, Ali Bashashati, Emma Laks, et al. Titan: inference of copy number architectures in clonal cell populations from tumor whole-genome sequence data. *Genome research*, 24(11):1881–1893, 2014.

18 Sridhar Hannenhalli and Pavel A Pevzner. Transforming cabbage into turnip. In *Proc. Annual ACM Symposium on the Theory of Computing*, volume 46, pages 178–189, New York, New York, USA, 1995. doi:10.1145/225058.225112.

19 Sridhar Hannenhalli and Pavel A Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proc. IEEE Symposium on Foundations of Computer Science*, volume 36, pages 581–592, 1995. doi:10.1109/SFCS.1995.492588.

20 Jun Inoue, Yukuto Sato, Robert Sinclair, Katsumi Tsukamoto, and Mutsumi Nishida. Rapid genome reshaping by multiple-gene loss after whole-genome duplication in teleost fish suggested by mathematical modeling. *Proceedings of the National Academy of Sciences*, 112(48):14918–14923, 2015. doi:10.1073/pnas.1507669112.

21 Olivier Jaillon, Jean-Marc Aury, Frédéric Brunet, Jean-Louis Petit, Nicole Stange-Thomann, Evan Mauceli, Laurence Bouneau, Cécile Fischer, Catherine Ozouf-Costaz, Alain Bernot, et al. Genome duplication in the teleost fish tetraodon nigroviridis reveals the early vertebrate proto-karyotype. *Nature*, 431(7011):946–957, 2004.

22 Jakub Kováč. On the complexity of rearrangement problems under the breakpoint distance. *Journal of Computational Biology*, 21(1):1–15, 2014. doi:10.1089/cmb.2013.0004.

23 Yilong Li, Nicola D. Roberts, Jeremiah A. Wala, Ofer Shapira, Steven E. Schumacher, Kiran Kumar, Ekta Khurana, Sebastian Waszak, Jan O. Korbel, James E. Haber, Marcin Imielinski, Kadir C. Akdemir, Eva G. Alvarez, Adrian Baez-Ortega, Rameen Beroukhim, Paul C. Boutros, David D. L. Bowtell, Benedikt Brors, Kathleen H. Burns, Peter J. Campbell, Kin Chan, Ken Chen, Isidro Cortés-Ciriano, Ana Dueso-Barroso, Andrew J. Dunford, Paul A. Edwards, Xavier Estivill, Dariush Etemadmoghadam, Lars Feuerbach, J. Lynn Fink, Milana Frenkel-Morgenstern, Dale W. Garsed, Mark Gerstein, Dmitry A. Gordenin, David Haan, James E. Haber, Julian M. Hess, Barbara Hutter, David T. W. Jones, Young Seok Ju, Marat D. Kazanov, Leszek J. Klimczak, Youngil Koh, Jan O. Korbel, Eunjung Alice Lee, Jake June-Koo Lee, Andy G. Lynch, Geoff Macintyre, Florian Markowetz, Iñigo Martincorena, Alexander Martinez-Fundichely, Matthew Meyerson, Satoru Miyano, Hidewaki Nakagawa, Fabio C. P. Navarro, Stephan Ossowski, Peter J. Park, John V. Pearson, Montserrat Puiggròs, Karsten Rippe, Nicola D. Roberts, Steven A. Roberts, Bernardo Rodriguez-Martin, Steven E. Schumacher, Ralph Scully, Mark Shackleton, Nikos Sidiropoulos, Lina Sieverling, Chip Stewart, David Torrents, Jose M. C. Tubio, Izar Villasante, Nicola Waddell, Jeremiah A. Wala, Joachim Weischenfeldt, Lixing Yang, Xiaotong Yao, Sung-Soo Yoon, Jorge Zamora, Cheng-Zhong Zhang, Peter J. Campbell, PCAWG Structural Variation Working Group, and PCAWG Consortium. Patterns of somatic structural variation in human cancer genomes. *Nature*, 578(7793):112–121, 2020. doi:10.1038/s41586-019-1913-9.

24 Stefano Mangiola, Matthew KH Hong, Marek Cmero, Natalie Kurganovs, Andrew Ryan, Anthony J Costello, Niall M Corcoran, Geoff Macintyre, and Christopher M Hovens. Comparing nodal versus bony metastatic spread using tumour phylogenies. *Scientific reports*, 6:33918, 2016.

25 Julia Mixtacki. Genome halving under DCJ revisited. In Xiaodong Hu and Jie Wang, editors, *Proc. Computing and Combinatorics*, volume 5092 of *Lecture Notes in Computer Science*, pages 276–286. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-69733-6.

26 Serena Nik-Zainal, Peter Van Loo, David C Wedge, Ludmil B Alexandrov, Christopher D Greenman, King Wai Lau, Keiran Raine, David Jones, John Marshall, Manasa Ramakrishna, et al. The life history of 21 breast cancers. *Cell*, 149(5):994–1007, 2012.

27 Layla Oesper, Ahmad Mahmoody, and Benjamin J Raphael. Theta: inferring intra-tumor heterogeneity from high-throughput dna sequencing data. *Genome biology*, 14(7):R80, 2013.

28 J D Palmer and L A Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 28(1-2):87–97, 1988. URL: http://www.ncbi.nlm.nih.gov/pubmed/3148746.

29 Alexandre Pelé, Mathieu Rousseau-Gueutin, and Anne-Marie Chèvre. Speciation success of polyploid plants closely relates to the regulation of meiotic recombination. *Frontiers in Plant Science*, 9:907, 2018. doi:10.3389/fpls.2018.00907.

30 Marina Petkovic, Thomas BK Watkins, Emma C Colliver, Sofya Laskina, Charles Swanton, Kerstin Haase, and Roland F Schwarz. Whole-genome doubling-aware copy number phylogenies for cancer evolution with medicc2. *bioRxiv*, 2021. doi:10.1101/2021.02.28.433227.

31 Pavel Pevzner and Glenn Tesler. Transforming men into mice. In *Proc. Seventh annual international conference on Research in Computational Molecular Biology*, pages 247–256, New York, New York, USA, 2003. ACM Press. doi:10.1145/640075.640108.

32 David Sankoff and Mathieu Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of computational biology*, 5(3):555–570, 1998.

33 Roland F Schwarz, Charlotte KY Ng, Susanna L Cooke, Scott Newman, Jillian Temple, Anna M Piskorz, Davina Gale, Karen Sayal, Muhammed Murtaza, Peter J Baldwin, et al. Spatial and temporal heterogeneity in high-grade serous ovarian cancer: a phylogenetic analysis. *PLoS medicine*, 12(2):e1001789, 2015.

34 Roland F Schwarz, Anne Trinh, Botond Sipos, James D Brenton, Nick Goldman, and Florian Markowetz. Phylogenetic quantification of intra-tumour heterogeneity. *PLoS computational biology*, 10(4):e1003535, 2014.

35 Ronglai Shen and Venkatraman E Seshan. Facets: allele-specific copy number and clonal heterogeneity analysis tool for high-throughput dna sequencing. *Nucleic acids research*, 44(16):e131–e131, 2016.

36 Andrea Sottoriva, Haeyoun Kang, Zhicheng Ma, Trevor A Graham, Matthew P Salomon, Junsong Zhao, Paul Marjoram, Kimberly Siegmund, Michael F Press, Darryl Shibata, et al. A big bang model of human colorectal tumor growth. *Nature genetics*, 47(3):209, 2015.

37 Steven H Strauss, Jeffrey D Palmer, Glen T Howe, and Allan H Doerksen. Chloroplast genomes of two conifers lack a large inverted repeat and are extensively rearranged. *Proceedings of the National Academy of Sciences*, 85(11):3898–3902, 1988.

38 Alfred H Sturtevant and Th Dobzhansky. Inversions in the third chromosome of wild races of drosophila pseudoobscura, and their use in the study of the history of the species. *Proceedings of the National Academy of Sciences*, 22(7):448–450, 1936.

39 Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10(1):120, 2009. doi:10.1186/1471-2105-10-120.

40 Robert Warren and David Sankoff. Genome halving with double cut and join. *Journal of Computational Biology*, 7(2):357–371, 2009.

41 Robert Warren and David Sankoff. Genome aliquoting revisited. *Journal of Computational Biology*, 18(9):1065–1075, 2011. URL: http://online.liebertpub.com/doi/abs/10.1089/cmb.2011.0087.

42 Kenneth H. Wolfe and Denis C. Shields. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature*, 387:708 EP–, 1997. doi:10.1038/42711.

43 Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005. doi:10.1093/bioinformatics/bti535.

44 Simone Zaccaria, Mohammed El-Kebir, Gunnar W. Klau, and Benjamin J. Raphael. Phylogenetic copy-number factorization of multiple tumor samples. *Journal of Computational Biology*, 25(7):689–708, 2018. PMID: 29658782. doi:10.1089/cmb.2017.0253.

45 Travis I Zack, Steven E Schumacher, Scott L Carter, Andrew D Cherniack, Gordon Saksena, Barbara Tabak, Michael S Lawrence, Cheng-Zhong Zhang, Jeremiah Wala, Craig H Mermel, et al. Pan-cancer patterns of somatic copy number alteration. *Nature genetics*, 45(10):1134, 2013.

46 Ron Zeira, Meirav Zehavi, and Ron Shamir. A linear-time algorithm for the copy number transformation problem. *Journal of Computational Biology*, 24(12):1179–1194, 2017.

47 Chunfang Zheng, Qian Zhu, Zaky Adam, and David Sankoff. Guided genome halving: hardness, heuristics and the history of the hemiascomycetes. *Bioinformatics (Oxford, England)*, 24(13):i96–i104, July 2008. doi:10.1093/bioinformatics/btn146.

S1 Appendix

S1.1 Reducing input profile size

In order to show these Proposition 1 and Proposition 2, we define two ways of modifying profiles that "preserve" the validity of a transformation between them. Let $S = \langle s_i \rangle$ and $T = \langle t_i \rangle$ be CNPs with n genes, and let C be a transformation for $S \to T$. We *remove* a gene i from S and T to obtain new CNPs S' and T', where

$$S' = \langle s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n \rangle$$
$$T' = \langle t_1, t_2, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n \rangle$$

Then, we can modify C to obtain a new transformation C' for $S' \to T'$, where $|C'| \leq |C|$, because we can modify each event in C to "skip" gene i (Figure S1a).

Similarly, we can *insert* a gene i to S and T to get S^* and T^*, such that

$$S^* = \langle s_1, s_2, \ldots, s_i, \mathbf{s_i}, s_{i+1}, \ldots, s_n \rangle$$
$$T^* = \langle t_1, t_2, \ldots, t_i, \mathbf{t_i}, t_{i+1}, \ldots, t_n \rangle$$

That is, we create a new gene next to gene i, with the same value as the gene at i. Then, we can modify a transformation C for $S \to T$ to create a transformation C^* for $S^* \to T^*$, where $|C^*| \leq |C|$, because we can modify each event in C to "stretch over" a new gene at i (Figure S1b).

(a) Removing gene 2. **(b)** Inserting gene 1.

Figure S1 Modifying profiles and transformations by removing and inserting genes.

S1.1.1 Proof of Proposition 1

Proof. Let $\hat{S}$ be an optimal preduplication profile for $\eta_p(T)$ and let C be a transformation for $p\hat{S} \to T$ of size $\eta_p(T)$. We can delete gene i from $\hat{S}$ to get $\hat{S}'$ and since T' can be formed from T by also removing gene i, we can obtain a transformation C' for $p\hat{S}' \to T'$ such that $|C'| \leq |C|$. This implies $\eta_p(T') \leq \eta_p(T)$.

To show the other direction, we insert a gene at i into $\hat{S}'$ and T' to get $\hat{S}^*$ and T^*, respectively. Then, we have some transformation C^* for $p\hat{S}^* \to T^*$ such that $|C^*| \leq |C'|$. Finally, we can set $\hat{s}^*_i = 0$ and $t^*_i = 0$, such that $\hat{S}^*$ is now equal to $\hat{S}$ and T^* is now equal to T. But, C^* is still valid for $p\hat{S} \to T$, because modifying $\hat{S}^*$ in such a manner does not affect any other gene, and preserves the validity of C^* at gene i, which has value zero in both $p\hat{S}$ and T. So, $\eta_p(T) \leq \eta_p(T')$, which completes the proof. ◀

S1.1.2 Proof of Proposition 2

Proof. First, we show $\eta_p(T') \leq \eta_p(T)$. Let $\hat{S}$ be an optimal preduplication profile for $\eta_p(T)$, and let C be a transformation $p\hat{S} \to T$. We can remove gene $i+1$ from $\hat{S}$ to get $\hat{S}'$, and so there is a valid transformation C' for $p\hat{S}' \to T'$ that is no larger than C. This implies that $\eta_p(T') \leq \eta_p(T)$, and that we can get an optimal preduplication profile for T' by deleting a gene of T.

Next, we show $\eta_p(T') \geq \eta_p(T)$. Let $\hat{S}'$ be an optimal preduplication profile (with $n-1$ genes) for $\eta_p(T')$, and let C' be a transformation $p\hat{S}' \to T'$ of size $\eta_p(T')$. Then, we can insert a gene at i into $\hat{S}'$ to get $\hat{S}^*$. Similarly, we can insert a gene at i into T' to get T^*. Then, we obtain a transformation C^* for $p\hat{S}^* \to T^*$ such that $|C^*| \leq |C'|$. Finally, because $t_{i+1} \geq t_i$, we need to modify $\hat{S}^*$ and T^* such that $T^* = T$, while preserving the validity of C^* for $p\hat{S}^* \to T^*$. Note that because $t_i \equiv t_{i+1} \bmod p$ and $t_i \leq t_{i+1}$, it follows that $(t_{i+1} - t_i) = kp$ for some $k \geq 0$. Since S^* and T^* are a result of gene insertion at i, we have $\hat{s}^*_{i+1} = \hat{s}'_i \neq 0$ and $t^*_{i+1} = t_i \neq 0$. Thus, we can increase $\hat{s}^*_{i+1}$ by k and increase t^*_{i+1} by kp to get two new profiles, $\tilde{S}$ and $\tilde{T}$. After doing so, C^* is still a valid transformation for $p\tilde{S} \to \tilde{T}$ because:

- We did not change the values of genes other than $i+1$ in either profile and we did not modify C^*, so C^* is still valid at these genes.
- Consider gene $i+1$. Before modifying $\hat{S}^*$ and T^*, we had that $\hat{s}^*_{i+1} = \hat{s}'_i \neq 0$ and $t^*_{i+1} = t_i \neq 0$. Hence, by Observation 1, if a is the number of amplifications in C^* that target gene $i+1$ and if d is the number of deletions that target gene $i+1$, then

  $$t^*_{i+1} = p\hat{s}^*_{i+1} + a - d$$

  Now, when we modify $\hat{S}^*$ and T to get $\tilde{S}$ and $\tilde{T}$, we set $\tilde{s}_{i+1} = \hat{s}^*_{i+1} + k$ and $\tilde{t}_{i+1} = t^*_{i+1} + kp$. Note that $\tilde{t}_{i+1} = p\tilde{s}_{i+1} + a - d$, because we can add kp to both sides of the previous equation. In addition, both $\tilde{t}_{i+1}$ and $\tilde{s}_{i+1}$ are nonzero, so C^* is valid at gene $i+1$.

Together, we have $T^* = T$, a preduplication profile S^* and transformation $p\hat{S}^* \to T$ of size $\eta_p(T')$, This implies $\eta_p(T) \leq \eta_p(T')$. ◀

S1.2 Computing $y_1(x), y'_1(x), y_2(x), y'_2(x)$

▶ **Proposition S5.** *$y_1(x), y'_1(x), y_2(x), y'_2(x)$ are all computable in $O(1)$ time.*

Proof. We show each computation separately.

- To compute $y_1(x)$, note that

  $$\begin{aligned} y_1(x) &= \min\{y \mid y \in \mathcal{D}_{i-1},\ y \geq x\} \\ &= \min\{b^-_{i-1} + kp \mid k \geq 0 \wedge b^-_{i-1} + kp \leq np \wedge b^-_{i-1} + kp \geq x\} \\ &= b^-_{i-1} + p \cdot \min\{k \geq 0 \mid np \geq b^-_{i-1} + kp \geq x\} \end{aligned}$$

  So, we can find the min k that satisfies these conditions in order to compute y_1.

  $$\begin{aligned} b^-_{i-1} + kp &\geq x \\ kp &\geq x - b^-_{i-1} \\ k &\geq (x - b^-_{i-1})/p \end{aligned}$$

  We can compute the right hand side, take the ceiling; the result is always non-negative because $|x - b^- i - 1|$ cannot exceed p. Afterwards, we double check that this satisfies the bound $b^-_{i-1} + kp \leq np$ that we get from bounding the number of total events. If it does, we have all we need to compute $y_1(x)$. If it does not, then $y_1(x)$ does not exist, and we ignore its term in the $\min\{\dots\}$ when computing $D[i, x]$.

- To compute $y_2(x)$, we use a similar approach.

$$\begin{aligned} y_2(x) &= \max\{y \mid y \in \mathcal{D}_{i-1},\ y \le x\} \\ &= \max\{b^-_{i-1} + kp \mid k \ge 0 \wedge b^-_{i-1} + kp \le np \wedge b^-_{i-1} + kp \le x\} \\ &= b^-_{i-1} + p \cdot \max\{k \ge 0 \mid b^-_{i-1} + kp \le np \wedge b^-_{i-1} + kp \le x\} \end{aligned}$$

  So, we can find the max k that satisfies the conditions to compute y_2.

$$\begin{aligned} b^-_{i-1} + kp &\le x \\ kp &\le x - b^-_{i-1} \\ k &\le (x - b^-_{i-1})/p \end{aligned}$$

  We can pick the max k that works by computing the right hand side and taking the floor to get k. We do not need to check that $b^-_{i-1} + kp \le np$, because we get this from $x \le np$, but we do need to be careful to check if $k \ge 0$; if not, then x is small enough that no choice of k works. If such a k exists, then $y_2(x) = b^-_{i-1} + kp$. Note that $y_2(x) = y_1(x) - p$, because y_2 and y_1 are computed from the closest multiples of p that "sandwich" x.
- Computing $y'_1(x)$ is similar to computing $y_1(x)$, with the additional condition that $t_{i-1} - b^+_{i-1} - kp \ge p$ to ensure that $\hat{s}_{i-1} \ge 1$.

$$\begin{aligned} y'_1(x) &= \min\{y \mid y \in \mathcal{A}_{i-1},\ y \ge x\} \\ &= \min\{b^+_{i-1} + kp \mid k \ge 0 \wedge b^-_{i-1} + kp \le np \wedge b^-_{i-1} + kp \ge x \wedge t_{i-1} - b^+_{i-1} - kp \ge p\} \\ &= b^+_{i-1} + p \cdot \min\{k \ge 0 \mid np \ge b^+_{i-1} + kp \ge x \wedge t_{i-1} - b^+_{i-1} \ge (k+1)p\} \end{aligned}$$

  So, we can find the min k that satisfies the first condition in the same way that we computed $y_1(x)$, and check that this k satisfies the additional condition which enforces $\hat{s}_i \ge 1$.
- Computing $y'_2(x)$ is similar to computing $y_2(x)$, with the additional condition as mentioned in the case for $y'_1(x)$.

$$\begin{aligned} y'_2(x) &= \max\{y \mid y \in \mathcal{A}_{i-1},\ y \le x\} \\ &= \max\{b^+_{i-1} + kp \mid k \ge 0 \wedge b^-_{i-1} + kp \le np \wedge b^-_{i-1} + kp \le x \wedge t_{i-1} - b^+_{i-1} - kp \ge p\} \\ &= b^+_{i-1} + p \cdot \max\{k \ge 0 \mid b^+_{i-1} + kp \le np \wedge b^+_{i-1} + kp \le x \wedge t_{i-1} - b^+_{i-1} \ge (k+1)p\} \end{aligned}$$

  We compute the max k that satisfies all but the last condition in the same way that we computed $y_2(x)$. Then, compute the max k that satisfies the second condition, which is an upper bound on k. We can take the min of these two values to get the max k that satisfies both. ◀

S1.3 Supplemental results

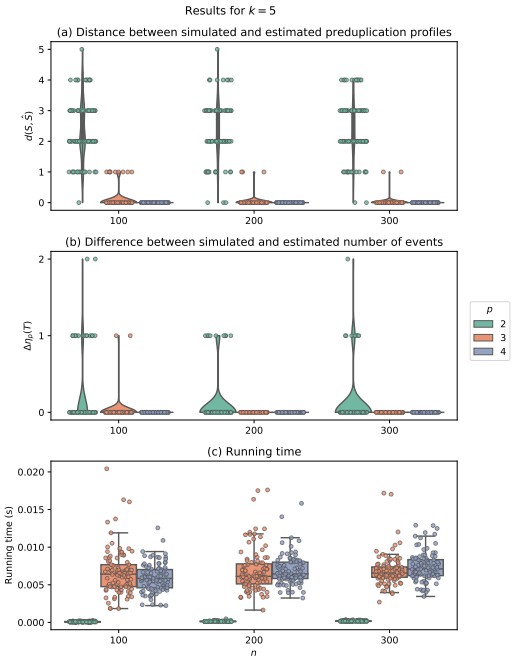


Figure S2 Simulation results for using $k = 5$ events after duplication. (a) $d(S, \hat{S})$ - the distance between simulated and estimated preduplication profiles. (b) $\Delta\eta_p(T)$ - the difference between the simulated and estimated number of events after duplication. (c) running time in seconds.

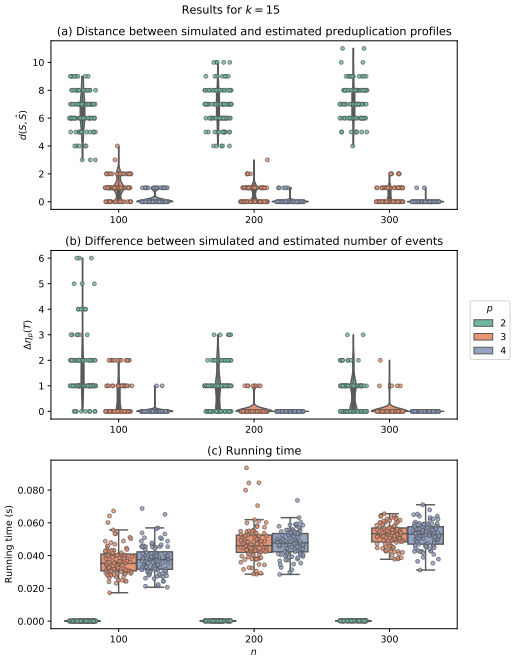


Figure S3 Simulation results for using $k = 15$ events after duplication. (a) $d(S, \hat{S})$ - the distance between simulated and estimated preduplication profiles. (b) $\Delta\eta_p(T)$ - the difference between the simulated and estimated number of events after duplication. (c) running time in seconds.

Efficient Haplotype Block Matching in Bi-Directional PBWT

Ardalan Naseri[1]
School of Biomedical Informatics, University of Texas Health Science Center at Houston, TX, USA

William Yue[1]
School of Biomedical Informatics, University of Texas Health Science Center at Houston, TX, USA

Shaojie Zhang[2]
Department of Computer Science, University of Central Florida, Orlando, FL, USA

Degui Zhi[2]
School of Biomedical Informatics, University of Texas Health Science Center at Houston, TX, USA

Abstract

Efficient haplotype matching search is of great interest when large genotyped cohorts are becoming available. Positional Burrows-Wheeler Transform (PBWT) enables efficient searching for blocks of haplotype matches. However, existing efficient PBWT algorithms sweep across the haplotype panel from left to right, capturing all exact matches. As a result, PBWT does not account for mismatches. It is also not easy to investigate the patterns of changes between the matching blocks. Here, we present an extension to PBWT, called bi-directional PBWT that allows the information about the blocks of matches to be present at both sides of each site. We also present a set of algorithms to efficiently merge the matching blocks or examine the patterns of changes on both sides of each site. The time complexity of the algorithms to find and merge matching blocks using bi-directional PBWT is linear to the input size.

Using real data from the UK Biobank, we demonstrate the run time and memory efficiency of our algorithms. More importantly, our algorithms can identify more blocks by enabling tolerance of mismatches. Moreover, by using mutual information (MI) between the forward and the reverse PBWT matching block sets as a measure of haplotype consistency, we found the MI derived from European samples in the 1000 Genomes Project is highly correlated (Spearman correlation r=0.87) with the deCODE recombination map.

2012 ACM Subject Classification Applied computing → Genetics; Applied computing → Computational genomics

Keywords and phrases PBWT, Bi-directional, Haplotype Matching

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.19

Supplementary Material *Software (Source Code)*: `https://github.com/ZhiGroup/bi-PBWT` archived at `swh:1:dir:ac5c1bfc4d8e31d338b887dd9d5131bbeefbe09a`

Funding This work was supported by the National Institutes of Health R01HG010086.

Acknowledgements This research has been conducted using the UK Biobank Resource under Application Number 24247.

1 Introduction

Diploid organisms such as humans inherit two copies of chromosomes, one from each parent. Each haplotype sequence of the two copies of a chromosome can be represented as a long string. Haplotype matches are usually considered to be binary and the matches between any pair of haplotypes may be due to a common ancestor or natural selection. While short

[1] These authors contributed equally to this work.
[2] Corresponding authors.

matches between two individuals may have been caused by chance, a match shared between several haplotypes or individuals are more informative since it is less likely that the matches occurred by random. The lower the probability of haplotype matches by chance, the stronger the evidence that the segments have been inherited from a common ancestor [14].

The availability of large-scale genetic data has promoted the need for efficient algorithms and methods. Positional Burrows-Wheeler Transform (PBWT) [5] proposed by Richard Durbin, provides an efficient data structure for compression and searching for matches in large haplotype sequences. The underlying idea of PBWT is to sort the haplotype sequences at each variant site k by their reversed prefix order. An additional array, called the divergence array, keeps track of the reverse prefix match between adjacent haplotypes in sorted order. The presented algorithms in PBWT provide an efficient approach to find all pairwise matches between haplotype sequences ending at each variant site.

PBWT data structure has already been utilized to find matching blocks in large haplotype panels [4, 1, 11]. A matching block contains several haplotype sequences that share a common reverse prefix of length L at a site. The concept of a matching block is a direct extension of pairwise matches to multi-way matches due to transitivity. Instead of a sequence with a minimum length of L shared by only two haplotypes, a block of matches is defined by a minimum number of haplotypes or individuals (W), where $W \geq 2$.

For the sake of memory efficiency, the PBWT algorithms are typically a scan across a haplotype panel from the first variant site in a forward direction. At each variant site during the pass, the panel is sorted by the reversed prefix order while updating the divergence values for each haplotype. The computed values are only kept transiently and only the active array at the current variant site k is kept in memory. As a result, the memory usage for PBWT algorithms will be only $O(M)$, where M denotes the number of haplotypes. This, however, comes with the cost of only being able to access data at a single site and only from one direction. While the forward-only PBWT sweep is sufficient for identifying pairwise matches between any two haplotypes or matching blocks, it is not efficient in determining the boundaries of the matching blocks nor does it facilitate the tracking of changes in matching blocks. As a result, no mismatch will be allowed in otherwise long matching blocks and recombination events cannot be observed between blocks of haplotypes around a site.

Here, we present an extension to Durbin's PBWT by allowing the information at both directions from site k to be accessible. This is enabled by a memory-efficient two-pass sweep: first, a reverse pass storing all PBWT data structures in an intermediate file and then a forward pass creating a transient forward PBWT. In the forward PBWT, we have random access to the reverse PBWT at each variant site and observe the potential changes beyond the variant site using the pre-computed reverse PBWT.

The bi-directional PBWT can be used to efficiently track haplotype sequences across the matching blocks at both the forward and the reverse PBWTs. Since the matching blocks at either side of a variant site define the partition of the sequence indices, the correspondence between the forward matching blocks and the reverse matching blocks can be expressed as the intersection of their partitions. In the context of PBWT, we formulate the problem as the *all partition matching problem*, where all matching partitions are enumerated, and the *large partition matching problem*, where only sufficiently large partitions are reported. Both problems can be solved by efficient $O(M)$ algorithms. In doing so, bi-directional PBWT allows integration of the changes within a matching block.

We demonstrate two potential applications of the haplotype block matching algorithm using bi-directional PBWT. The first application involves the identification of matching blocks allowing mismatches. Additionally, the blocks of matches that undergo changes around a site may indicate phasing errors or recombination events. Preliminary results of these applications are shown using UK Biobank and 1000 Genomes project data [3].

2 Methods

2.1 Background and notation

Following Durbin's notation [5], we define a panel of haplotype sequences as $X \in \{0,1\}^{M\times N}$, a two-dimensional matrix of binary values, where M denotes the number of haplotype sequences and N the number of variant sites. Each row is a haplotype sequence and each column $X_k, k = 0, ..., N-1$ is an array representing the values of haplotypes at the site k. If s and t are two haplotype sequences, and $s[j,k)$ and $t[j,k)$ denote their subsequences from site j to site $k-1$, then there is a match between s and t from j to k if $s[j,k) = t[j,k)$.

The concept of haplotype match can be defined over blocks [4, 1, 11]. Assume C is the set of sequence indices: $C = \{0, ..., M-1\}$. We can define a haplotype matching block or a block as a tuple $(c, j, k), j < k$, where $c \subset C$ is a subset of sequence indices and j and $k-1$ denote the starting and ending sites of the match, respectively. The length of a block is defined as $l = k - j$, and the width of a block is $w = |c|$. If $l \geq L$ and $w \geq W$, we call the block a (L, W)-block. In particular, a block is called width maximal at each site k if the number of sequences sharing a substring with a length $\geq L$ cannot be increased. A block is called length maximal if one or multiple sequences in the block have a different value at the site k resulting in less than W haplotypes in the block. Width and length maximal blocks can be found in $O(NM)$ time in Positional Burrows-Wheeler Transform (PBWT) [4, 1, 11].

The basic idea using PBWT to find matching blocks is that at any site k, all matching haplotype sequences with a length $\geq L$ are adjacent to each other in the PBWT panel [11]. In other words, if a set of haplotypes builds a block of matches with a length L, they will be contiguous in the PBWT panel.

2.1.1 PBWT data structures: PBWT matrix, positional prefix array, and divergence array

The underlying idea of the Positional Burrows-Wheeler Transform (PBWT) is to store haplotype sequences based on their reversed prefix order. It also enables efficient algorithms for identifying matches by introducing additional data structures augmenting the original haplotype panel X. The positional prefix array a_k contains the sequence indices sorted based on their reversed prefix order. The PBWT array y_k stores the values of haplotype sequences in the reversed prefix order at each site ($y_k[i] = X_k[a_k[i]]$). The divergence array d_k at the variant site k for each haplotype stores the starting position of the match between the haplotype with its preceding haplotype sequence in the reversed prefix order. In other words, the divergence value keeps track of the starting site index of the longest match for each haplotype. We refer to the value of the divergence array for each haplotype as its divergence value. d_k is utilized to both identify a long match with a length $\geq L$ and determine the starting position of the match.

Note that in a typical PBWT all-versus-all matching algorithm, the haplotype panel is swept from left to right and the data structures a_k, y_k, and d_k can be built transiently, thus enabling an efficient memory footprint of $O(M)$. Indeed this is the primary mode that PBWT can be used in large data sets. However, such sweeping algorithm precludes access to information from both sides of the site k, and thus limiting the investigation of changes of haplotype blocks.

2.2 Bi-directional PBWT

2.2.1 Overview

Here, we assume that PBWT data structures for both forward and reverse directions are available at each site. Figure 1 shows the bi-directional PBWT data structures using a simple example of a haplotype panel. Given a minimum length L, width maximal blocks ending at site k define a partition of the set of indices C for either side. Assume the *block set* $S(k, L)$ represents a set of blocks of matching haplotypes in the forward PBWT ending at a variant site k. In other words, $S(k, L)$ is a partition of the set of sequence indices C. Similarly, we define the block set $T(k, L)$ for the reverse PBWT at site k, another partition of C. For the sake of simplicity, we omit the k and L and denotes the partitions as S and T. It is of our interest to find the matching between these two block sets, where the matching is defined over all subsets shared by S and T, also known as the intersection of partitions $U(S, T) = \{s \cap t | s \in S \wedge t \in T\} \backslash \{\varnothing\}$. Additionally, the forward and the reverse PBWTs do not have to be back-to-back and a small gap may be allowed. I.e., we can consider the matching between $S(k, L)$ and $T(k + g, L)$, where g is the size of the gap.

Here, we define two versions of the matching problems, the *all-block matching problem* and the *W-width block matching problem*. For the *all-block matching problem*, the goal is to enumerate all possible blocks in $U(S, T)$. In the case of haplotype sequences from population genotyping data, we expect there will be a strong correlation between the blocks of S and T. This can be quantified by the mutual information (MI) between the sets of matching blocks in forward (S) and reverse (T) directions:

$$MI(S,T) = \sum_{s \in S} \sum_{t \in T} p_{(S \cdot T)}(s,t) log\left(\frac{p_{(S \cdot T)}(s,t)}{p_S(s) p_T(t)}\right),$$

where $p_{(S \cdot T)}(s, t)$ is the joint probability function of S and T, which can be calculated by enumerating the blocks in $U(S, T)$.

For the *W-width block matching problem*, the goal is the find all the matching blocks in $U(S, T)$ that contain $\geq W$ haplotypes. This may sound like yet another way of solving the $(2L + g, W)$-block problem. However, the benefit of defining the problem as such is that it would allow mismatches in the gap region of an otherwise exact matching block.

Our approach for finding blocks of matches using bi-directional PBWT can be split into three subroutine algorithms: 1) Finding blocks of matches exceeding the given length and width cut-off using bi-directional PBWT. 2) Matching blocks from both sides. 3) Extracting the length of matches, in the case of finding $(2L + g, W)$-blocks. Figure 1 shows a simple schematic of our method for detecting clusters around the site k. The matching blocks from both sides are considered around a given gap in terms of sites.

2.2.2 Algorithm 1: Find width maximal blocks

We start by independently finding width maximal blocks ((W, L)-blocks) on both sides at each site k that contain at least W sequences and include at least L sites. This will output the block sets $S(k, L)$ and $T(k, L)$ for each site k. Using the divergence arrays for each side, we can easily find blocks that extend for at least L matching sites. Blocks form contiguous segments in the positional prefix array and are capped off at both ends by large divergence values. Blocks of matches are separated by a sequence j where $d_j > k - L$. We can separate blocks of matches at each site and assign a unique ID to each block. For each sequence, we store its corresponding block ID in an array for lookup in Algorithm 2. Algorithm 1 has a

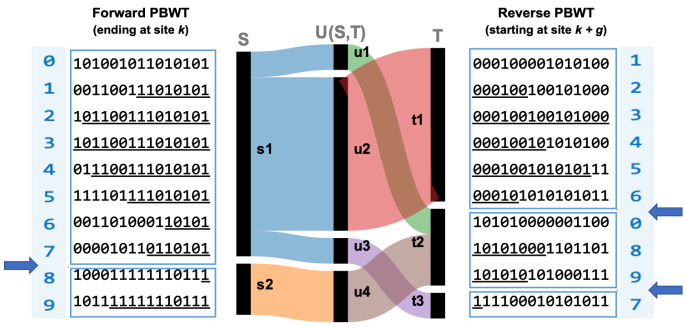


Figure 1 Bi-directional PBWT data structures. The standard PBWT data structures for the forward and the reverse PBWTs are shown at both sides. For $L = 5$, the L-block set at the forward PBWT, $S = \{s1 = \{0, 1, 2, 3, 4, 5, 6, 7\}, s2 = \{8, 9\}\}$ and the L-block set at the reverse PBWT, $T = \{t1 = \{1, 2, 3, 4, 5, 6\}, t2 = \{0, 8, 9\}, t3 = \{7\}\}$, are integrated to form $U = U(S, T) = \{u1 = \{0\}, u2 = \{1, 2, 3, 4, 5, 6\}, u3 = \{7\}, u4 = \{8, 9\}\}$. The arrows on the sides mark the boundaries of matching blocks.

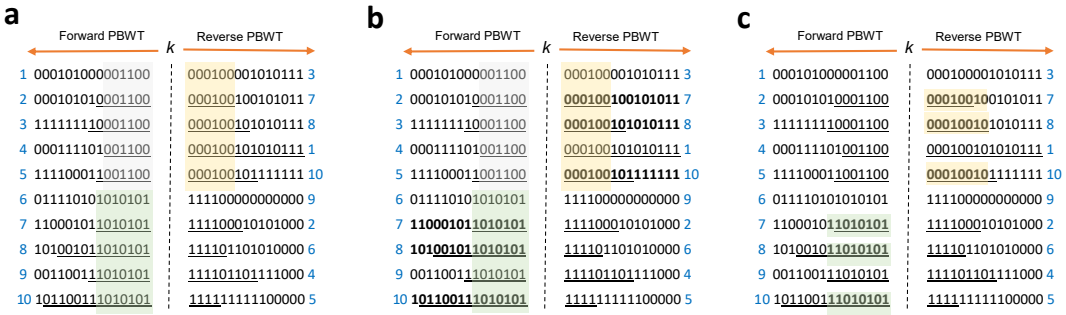


Figure 2 A simple schematic of bi-directional PBWT for finding blocks of matches before and after the site k: 1. Finding blocks of matches exceeding a given length in terms of sites L (≥ 6) and haplotypes W (≥ 3) at both sides separately (a). 2. Merging the blocks of matches from both sides (b). 3. Determination of the length in terms of sites (L) of continuous matching blocks (c). The length in terms of the number of sites in the matching block in green increases since the 6th sequence is discarded.

time complexity of $O(M)$ since we only iterate through the divergence array at each site. Figure 2 illustrates the process for finding width maximal blocks and Algorithm 1 outlines the pseudo-code for finding width maximal blocks on one side of the site.

2.2.3 Algorithm 2: Match blocks by enumerating intersection of partitions

Recall from Algorithm 1 that width maximal blocks represent a set of sequences that match for at least L sites. In Algorithm 2, we solve the *W-width block matching problem* by enumerating the intersection of block sets that exceed the minimum number of haplotypes (W) on both sides. Note that the *all-block matching problem* can be solved similarly by simply ignoring the minimum number of haplotypes W.

We define an array *link* where *link*[i] is the pair <forward block ID of haplotype i, reverse block ID of haplotype i>. After sorting *link*, haplotypes in the same forward and reverse blocks will be adjacent to each other and form contiguous sections in the array. Each section in the array that has the same two-block pair becomes a candidate for a matching block

Algorithm 1 Find width maximal blocks at site k.

```
blockID ← 1
start ← −1
for i ← 0, M − 1 do
    if d[i] > k − L then
        assign blockID to sequences in [start, i − 1]
        blockID = blockID + 1
        start ← i
    end if
end for
```

with the only criteria left to check being if the size of the block candidate is greater than or equal to W. Note that since the pairs in the *link* array only contain block ID values from $1 - M$, we can utilize a radix sort. Algorithm 2 has a time complexity of $O(M)$ time due to the radix sort and a memory complexity of $O(M)$. Figure 2 illustrates the process for merging blocks and Algorithm 2 outlines the pseudo-code for merging blocks.

Algorithm 2 Match blocks in forward and reverse PBWT at position k.

```
link ← []
for i ← 0, M − 1 do
    link[i] ← <forward block ID of haplotype i, reverse block ID of haplotype i>
end for
radixSort(link)
start ← 0
for i ← 1, M − 1 do
    if link[i] ≠ link[i − 1] then
        if i − start ≥ W then                ▷ Check if the block's width is at least W
            report width-maximal matching block from [start, i − 1]
        end if
        start ← i
    end if
end for
```

2.2.4 Algorithm 3: Report block length

Recall that Algorithm 1 only checks that a block matches for a minimum of L sites and doesn't compute how much further the block matches past those L sites. Once a block candidate from Algorithm 2 has met the size requirement of containing at least W sequences, we utilize Algorithm 3 to find the length of the block on both sides of site k. If on the left side, a block includes sequences at positions $x_0, x_1, ..., x_j$ in the positional prefix array, then we define *minPos* as the $\min_{0 \le i \le j} x_i$ and the *maxPos* as the $\max_{0 \le i \le j} x_i$. The length that the block extends to the left can then be represented by $k - \max_{minPos < i \le maxPos} d_i$. The same logic can be applied to find the length of the block on the right side. Note that we do not include *minPos* in the query range since the definition of the divergence array states that d_i is a comparison between string i and string $i - 1$ in the positional prefix array. To be able to compute the expression $\max_{minPos < i \le maxPos} d_i$ efficiently, we can utilize a range query data structure. Since we do not need to perform range updates, sparse table

provides a lightweight data structure that can be built in $O(M)$ time by dividing the data into blocks of size b ($b \in \Theta(log(M))$) and computing the sparse table over the maximums in each block. For each query, there will be a maximum of two b-sized blocks that will not be completely encapsulated by the range query. These two b-sized blocks will be found at the two boundaries of the range query and the maximum for these two blocks can be computed manually and efficiently using bitwise operations. Each range query can be performed in $O(b)$ where b is a small constant factor. Alternatively, Cartesian trees can be used which guarantees the time complexity of $O(M)$ for pre-processing and $O(1)$ for range queries. In practice, however, we found that using sparse tables with the block technique seemed to be faster than Cartesian trees in both precomputation and query runtimes. Since each sequence is only included in a single query and queries take $O(1)$ time, it takes $O(M)$ time to complete all queries. Thus, Algorithm 3's runtime is dominated by building the sparse table, which takes $O(M)$ time and $O(M)$ memory to construct. Figure 3 illustrates the process for reporting block length and Algorithm 3 outlines the pseudo-code for finding the length of a block reported in Algorithm 2.

Algorithm 3 Report block length.

1: **for all** blocks reported in Algorithm 2 **do**
2: $l_f = k - \max_{minPos<i<=maxPos} d_f[i]$ ▷ report forward length
3: $k' = k + g - 1$
4: $l_r = \max_{minPos<i<=maxPos} d_r[i] - k'$ ▷ report reverse length
5: **end for**

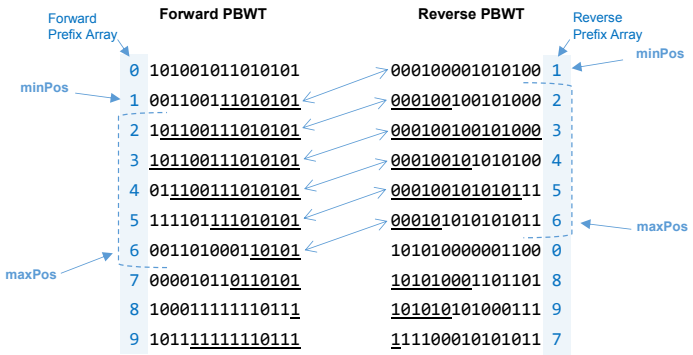


Figure 3 A simple example illustrating Algorithm 3 – Report Cluster Length. To find the length of the cluster on both sides, we identify $minPos$ and $maxPos$ on both sides and perform a maximum range query on the divergence arrays.

2.2.5 Efficient implementation of bi-directional PBWT

To compute a bi-directional PBWT efficiently, we must be able to efficiently read a VCF file in reverse order (from the last site to the first site). In our implementation, we utilized the C++ method $seekg()$ to manipulate the position of the input stream pointer. We begin by reading the first site in the VCF file and extracting the number of individuals, M, by counting the number of "|". We then jump to the end of the file and start the process of

reading the VCF file in reverse order line by line. To extract each line from the end of the file, we use $seekg()$ to move the input stream pointer back one position at a time until we reach a newline character. Since we already know M, we can optimize this process by initially moving the input stream pointer back $4 * M$ positions since we know each individual in the VCF file takes up 4 characters. Then we only need to move the input stream pointer back one character at a time through the fixed fields which usually comprise less than 500 characters. By doing this, we can compute the reverse PBWT with comparable efficiency to the forward PBWT.

Bi-directional PBWT builds off of Durbin's algorithms 1 and 2. In these 2 algorithms, Durbin iterates through all N sites while maintaining a positional prefix array (i.e. a sorted array of binary strings) and a divergence array (d_i stores the smallest value j such that the haplotype sequences in the prefix array positions i and $i-1$ match from $site_j$ to the current site). Before running the aforementioned algorithms, we create the PBWT data structure for reverse sequences first and then use the stored values while scanning the panel in the forward direction. Note that we only need to store values from the reverse PBWT since we can simultaneously compute blocks and the forward PBWT at the same time. Source code is available at `https://github.com/ZhiGroup/bi-PBWT`.

3 Results

3.1 Runtime and memory usage

We evaluated the run time and memory usage of bi-directional PBWT on large-scale haplotype panels. The scalability of bi-directional PBWT was evaluated with respect to the size of the input data and the length parameter. Table 1 shows the resource consumption when running bi-directional PBWT (both forward and reverse combined) on chromosome 21 of the UK Biobank panel comprising 974,818 haplotypes with the parameters of $L = 0.5$ Mbps and $W = 100$. Table 2 shows the runtime growth of bi-directional PBWT with respect to the input size M when all other variables are held constant. From Table 2, we can observe that our method has an asymptotic runtime of $O(M)$ with respect to M. Table 3 shows the runtime growth of bi-directional PBWT with respect to input size N when all other variables are held constant. From Table 3, we can see that the algorithm has an asymptotic runtime of $O(N)$ with respect to N. Table 4 shows the runtime growth of bi-directional PBWT with respect to the parameter L when all other variables are held constant. From Table 4, we can see that the asymptotic runtime of bi-directional PBWT is not affected by L. All experiments were carried out with a 2.10 GHz Intel Xeon E5-2620 v4 using a single core.

Table 1 Run time and memory usage of bi-directional PBWT on chromosome 21 of the UK Biobank.

Run time	Peak memory	Peak disk usage	#Blocks
3.2 hrs	261.44 MB	73.1 GB	245,563

$W = 100$, $L = 0.5$ Mbps.

Table 2 Runtime growth with respect to the number of haplotypes M.

Size of M	50,000	100,000	200,000	400,000	800,000
Time (s)	607.1	1,432.54	3,000.11	5,971.03	11,284.42

$N = 9793$, $W = 100$, $L = 0.5$ Mbps.

Table 3 Runtime growth with respect to the number of sites N.

Size of N	500	1000	2000	4000	8000
Time (s)	773.91	1,455.34	3,273.91	5,955.54	11,953.19

$M = 974818$, $W = 100$, $L = 0.5$ Mbps.

Table 4 Runtime growth with respect to the target length L.

Size of L (Mbps)	0.1	0.2	0.4	0.8	1.6
Time (s)	12,090.35	12,336.94	13,727.56	13,471.45	13,957.97

$M = 974818$, $N = 9793$

The benchmarks show that the asymptotic runtime is linear with respect to the size of the input and that the asymptotic memory usage is $O(M)$.

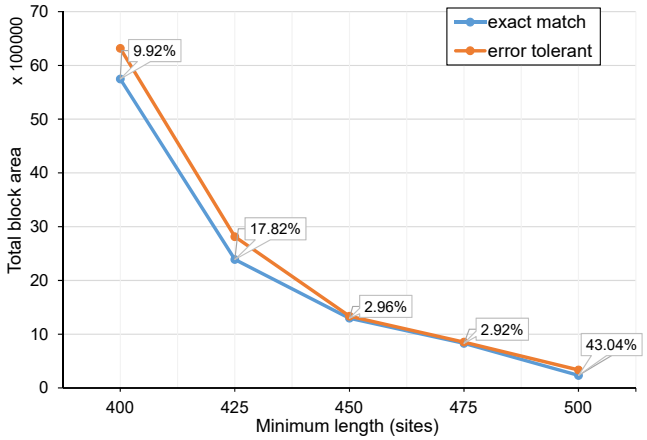


Figure 4 Comparison of total areas covered by matching blocks using exact matches vs. allowing a mismatching site. The percentage increase in the total area covered by matching blocks is shown for different minimum target lengths.

3.2 Blocks of haplotype matches in UK Biobank

To establish the benefit of allowing mismatches in the otherwise perfect matching blocks, we ran bi-directional PBWT either with a tolerance of mismatch or with a perfect match in the gap region. We used a gap size of $g = 1$ and haplotypes of chromosome 21 in the UK Biobank data [2] to count the total sizes of matching blocks as the total area that is covered by any matching block. We used the total area to investigate the differences between exact matches and allowing only one mismatch (error) using bi-directional PBWT. The total area and the percentages of differences for exact matching blocks and error-tolerant mode are shown in Figure 4. The minimum number of haplotypes W was set to 200. As shown in the figure, based on different cut-off values for the minimum length, the gain can vary from $\sim 3 - 43\%$. The percentage of additional area covered by matching blocks while allowing only 1 mismatch in the middle of blocks can result in $\sim 43\%$ more area compared to exact matches for $L = 500$.

We searched for haplotype matches using bi-directional PBWT in UK Biobank with a minimum length of 0.5 Mbps shared among at least 100 haplotypes using all individuals. Figure 5 shows the number of clusters at each site. Please note that there must be a mismatch

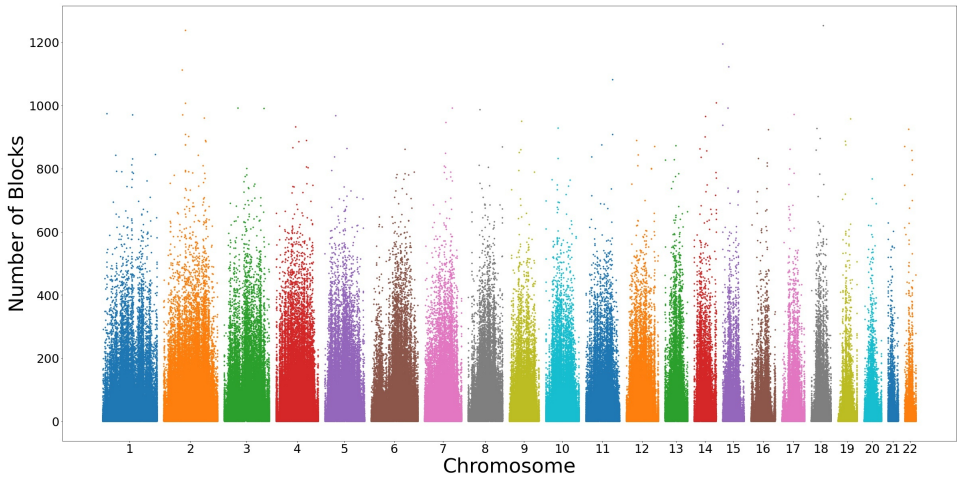


Figure 5 Manhattan plot for the number of blocks allowing a mismatching site in the block using all autosomal chromosomes in the UK Biobank data. The minimum length for both sides was set to 0.5 Mbps.

at each site in the gap for the cluster to be counted. The minimum length should also hold for both sides of matches around each site. The plot shows the availability of such haplotype blocks across the chromosomes. There's also a dip in chromosome 6 which covers the HLA region. Figure 6 shows the average number of haplotypes that have an alternating allele in each block at every site. As shown in the figure, there is a peak in chromosome 6 which again comprises the HLA region. The mismatches in each cluster are shared by several haplotypes and the sizes of the blocks in terms of the number of haplotypes are larger compared to other regions.

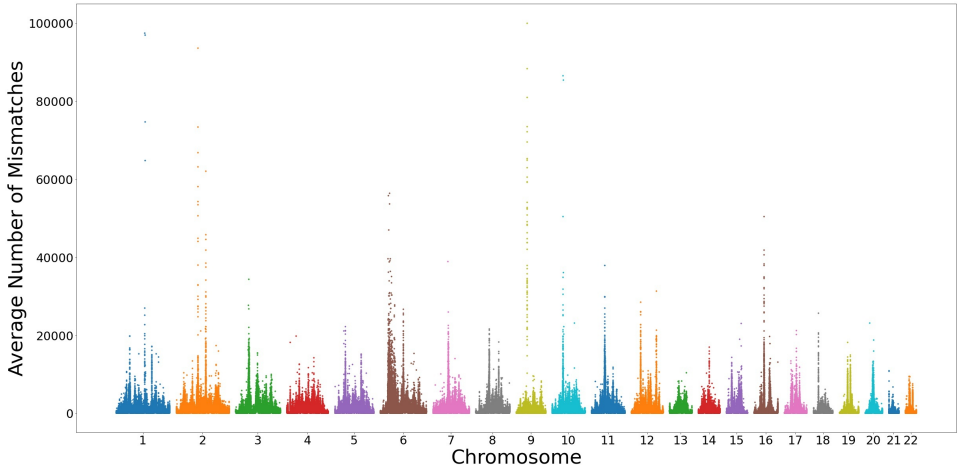


Figure 6 Manhattan Plot for the average number of mismatches in each block at every site using all autosomal chromosomes in the UK Biobank data.

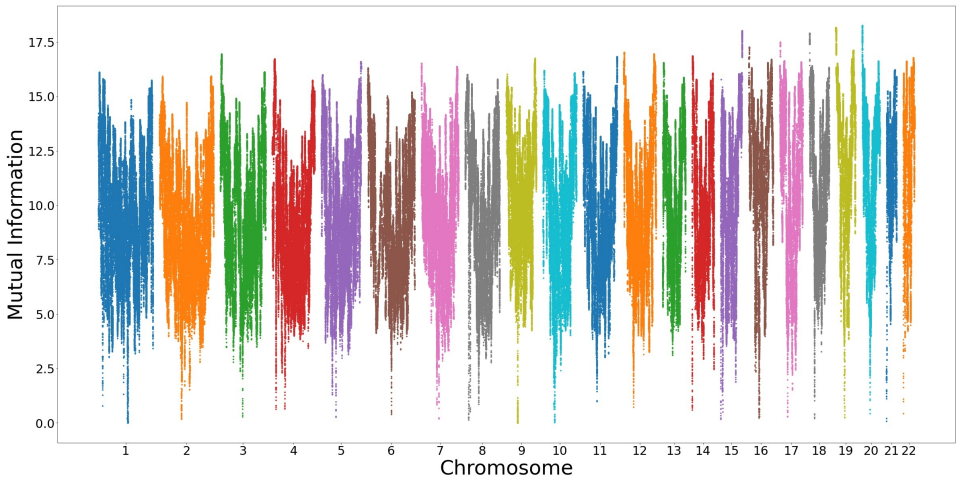


Figure 7 Manhattan plot for the mutual information at each site using all 22 autosomal chromosomes in the UK Biobank data.

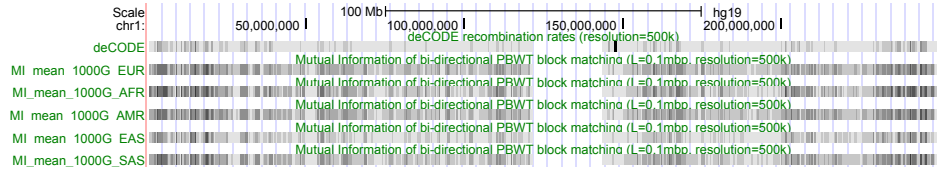


Figure 8 Mutual information (MI) calculated by bi-directional PBWT (L = 0.1 Mbps) in 1000 Genomes Project data and deCODE recombination rate at 500 kbps scale. The MI's were calculated for each population separately.

3.3 Correlation between Mutual Information and Genetic Map

The mutual information was calculated for all sites in the UK Biobank data. Figure 7 shows the Manhattan plot for the mutual information at each site using all 22 autosomal chromosomes. The minimum length of the block was set to 0.5 Mbps. We also calculated the mutual information in 1000 Genomes Project data using the minimum length cut-off of 0.1 Mbps. Figure 8 shows the mutual information for each population in 1000 Genomes Project using chromosome 1 and the deCODE [7] recombination rate (cM/Mbps) at 500k resolution. We also computed the correlation between the mutual information and deCODE recombination map using the UK Biobank data and 1000 Genomes Project data in chromosome 1 (L = 0.1 Mbps). The Spearman's rank correlation coefficient for chromosome 1 of all participants was 0.85 and the European population in 1000 Genomes Project was 0.87. The correlation coefficient for the European population is slightly higher as expected when compared to the deCODE genetic map. For UK Biobank, the Spearman's rank correlation coefficient was 0.82 using all participants. In general, the mutual information correlates with the recombination rate at low resolution.

4 Conclusions and Discussion

The PBWT data structure provides a concise representation of a haplotype panel which enables efficient exact haplotype matching. PBWT data structures have been used for genotype imputation [8, 12], building a representation of an ancestral recombination graph

(ARG) [13], query search [9], and detecting Identical by Descent (IBD) segments [10, 15, 6]. In this work, we presented an extension of the original PBWT data structure to bi-directional. To achieve the time and memory efficiency that is required for large biobank-scale data, we designed a set of efficient algorithms. Specifically, the algorithms aim to efficiently identify blocks of matches that are present in both directions of the PBWT instances around each variant site.

Bi-directional PBWT allows the investigation of matching blocks at each site beyond the currently active site in PBWT. Therefore, it provides flexibility in detecting matches in otherwise rigid exact haplotype matches in the original PBWT. In our results, we showed that allowing only one mismatching site will have some $\sim 3 - 43\%$ increase in the reported area involved in a matching block. The presented algorithms for allowing one mismatch can be extended to multiple mismatching sites occurring with a minimum distance. Assuming that blocks of matches are less likely to occur by random, the divergence values in the forward PBWT can be updated using the block information in the reverse PBWT. As a result, approximate haplotype matches can be enumerated in the forward PBWT. Therefore, it can provide an efficient haplotype matching solution that can tolerate genotyping errors.

We also showed that the mutual information (MI) obtained by bi-directional PBWT can be used to estimate recombination rates at low resolution. The Spearman correlation coefficient between the MI of the European samples in the 1000 Genomes Project and the deCODE genetic map was 0.87. The mutual information here certainly does not provide a fine resolution map, however, it does highlight another application of the bi-directional PBWT. Another approach to determine the recombination rates can be the calculation of *edit distance* of two block sets of matches in the forward and backward PBWT, where the edit distance is defined as the minimal number of sequence reassignments to achieve similar blocks in both directions.

References

1 Jarno Alanko, Hideo Bannai, Bastien Cazaux, Pierre Peterlongo, and Jens Stoye. Finding all maximal perfect haplotype blocks in linear time. *Algorithms for Molecular Biology*, 15(1):2, 2020.

2 Clare Bycroft, Colin Freeman, Desislava Petkova, Gavin Band, Lloyd T Elliott, Kevin Sharp, Allan Motyer, Damjan Vukcevic, Olivier Delaneau, Jared O'Connell, et al. The UK Biobank resource with deep phenotyping and genomic data. *Nature*, 562(7726):203–209, 2018.

3 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

4 Luís Cunha, Yoan Diekmann, Luis Kowada, and Jens Stoye. Identifying maximal perfect haplotype blocks. In *Brazilian Symposium on Bioinformatics*, pages 26–37. Springer, 2018.

5 Richard Durbin. Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.

6 William A Freyman, Kimberly F McManus, Suyash S Shringarpure, Ethan M Jewett, Katarzyna Bryc, The 23, Me Research Team, and Adam Auton. Fast and Robust Identity-by-Descent Inference with the Templated Positional Burrows–Wheeler Transform. *Molecular Biology and Evolution*, 38(5):2131–2151, 2021.

7 Bjarni V Halldorsson, Gunnar Palsson, Olafur A Stefansson, Hakon Jonsson, Marteinn T Hardarson, Hannes P Eggertsson, Bjarni Gunnarsson, Asmundur Oddsson, Gisli H Halldorsson, Florian Zink, et al. Characterizing mutagenic effects of recombination through a sequence-level genetic map. *Science*, 363(6425), 2019.

8 Po-Ru Loh, Petr Danecek, Pier Francesco Palamara, Christian Fuchsberger, Yakir A Reshef, Hilary K Finucane, Sebastian Schoenherr, Lukas Forer, Shane McCarthy, Goncalo R Abecasis, et al. Reference-based phasing using the haplotype reference consortium panel. *Nature Genetics*, 48(11):1443, 2016.

9 Ardalan Naseri, Erwin Holzhauser, Degui Zhi, and Shaojie Zhang. Efficient haplotype matching between a query and a panel for genealogical search. *Bioinformatics*, 35(14):i233–i241, 2019.

10 Ardalan Naseri, Xiaoming Liu, Kecong Tang, Shaojie Zhang, and Degui Zhi. RaPID: ultra-fast, powerful, and accurate detection of segments identical by descent (IBD) in biobank-scale cohorts. *Genome Biology*, 20(1):1–15, 2019.

11 Ardalan Naseri, Degui Zhi, and Shaojie Zhang. Discovery of runs-of-homozygosity diplotype clusters and their associations with diseases in UK Biobank. *medRxiv*, 2020. doi:10.1101/2020.10.26.20220004.

12 Simone Rubinacci, Olivier Delaneau, and Jonathan Marchini. Genotype imputation using the positional burrows wheeler transform. *PLoS Genetics*, 16(11):e1009049, 2020.

13 Vladimir Shchur, Liliia Ziganurova, and Richard Durbin. Fast and scalable genome-wide inference of local tree topologies from large number of haplotypes based on tree consistent pbwt data structure. *bioRxiv*, page 542035, 2019.

14 Elizabeth A Thompson. Identity by Descent: Variation in Meiosis, Across Genomes, and in Populations. *Genetics*, 194(2):301–326, 2013.

15 Ying Zhou, Sharon R Browning, and Brian L Browning. A fast and simple method for detecting identity-by-descent segments in large-scale data. *The American Journal of Human Genetics*, 106(4):426–437, 2020.

Fast Approximate Shortest Hyperpaths for Inferring Pathways in Cell Signaling Hypergraphs

Spencer Krieger[1]
Department of Computer Science, The University of Arizona, Tucson, AZ, USA

John Kececioglu
Department of Computer Science, The University of Arizona, Tucson, AZ, USA

Abstract

Cell signaling pathways, which are a series of reactions that start at receptors and end at transcription factors, are basic to systems biology. Properly modeling the reactions in such pathways requires *directed hypergraphs*, where an edge is now directed between two sets of vertices. Inferring a pathway by the most parsimonious series of reactions then corresponds to finding a *shortest hyperpath* in a directed hypergraph, which is NP-complete. The state of the art for shortest hyperpaths in cell-signaling hypergraphs solves a mixed-integer linear program to find an optimal hyperpath that is restricted to be acyclic, and offers no efficiency guarantees.

We present for the first time a heuristic for general shortest hyperpaths that properly handles *cycles*, and is guaranteed to be *efficient*. Its accuracy is demonstrated through exhaustive experiments on all instances from the standard NCI-PID and Reactome pathway databases, which show the heuristic finds a hyperpath that *matches* the state-of-the-art mixed-integer linear program on over 99% of all instances that are acyclic. On instances where only cyclic hyperpaths exist, the heuristic *surpasses* the state-of-the-art, which finds no solution; on every such cyclic instance, enumerating all possible hyperpaths shows that the solution found by the heuristic is in fact *optimal*.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Systems biology; Theory of computation → Shortest paths; Mathematics of computing → Hypergraphs

Keywords and phrases Systems biology, cell signaling networks, reaction pathways, directed hypergraphs, shortest hyperpaths, efficient heuristics, hyperpath enumeration

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.20

Supplementary Material Source code for the shortest hyperpath heuristic and hyperpath enumeration is available at `http://hyperpaths.cs.arizona.edu`.

Funding This research was supported by the US National Science Foundation through Grants CCF-1617192 and IIS-2041613 to John Kececioglu.

Acknowledgements We thank T. M. Murali for introducing us to the problem of shortest hyperpaths in cell-signaling hypergraphs, orienting us to the biology literature, and discussing the JUP/DSP example; Anna Ritz for discussing the NCI-PID and Reactome datasets, and providing the BioPax parser; and the anonymous referees for their helpful comments.

1 Introduction

Signaling pathways are cornerstones of molecular and cellular biology. They underly cellular communication, govern environmental response, and their perturbation has been implicated in the cause of many diseases. While signaling pathways have classically been modeled as ordinary graphs, using directed or undirected edges to link pairs of interacting molecules [31, 32], both Klamt, Haus and Theis [20] and Ritz, Tegge, Kim, Poirel and Murali [28] have shown that ordinary graphs cannot adequately represent cellular activity that involves the assembly and disassembly of protein complexes, and multiway reactions among such complexes.

[1] Corresponding author.

Directed *hypergraphs* are generalizations of ordinary graphs, where an edge (now called a hyperedge) is directed from one set of vertices (called its tail) to another set of vertices (called its head), and have been used to model many cellular processes [25, 16, 6, 14, 20, 24, 33, 27, 28]. In particular, a biochemical reaction that involves multiple *reactants* – all of which must be present for the reaction to proceed – and that results in multiple *products* – all of which are produced upon its completion – is correctly captured by a single *hyperedge*, directed from its set of reactants to its set of products. Despite hypergraphs affording more faithful models of reaction networks, the lack of practical hypergraph algorithms has hindered their potential for correctly representing and reasoning about molecular reactions.

Biologically, a typical cell-signaling pathway consists of membrane-bound receptors that bind to extracellular ligands, triggering intracellular cascades of reactions, culminating in the activation of transcriptional regulators and factors [2]. Computationally, treating receptors as sources, and transcription factors as targets, finding the most efficient way to synthesize a particular transcription factor from a set of receptors maps to the shortest hyperpath problem we consider here: Given a cell-signaling network whose reactants and reactions are modeled by the vertices and weighted hyperedges of a directed hypergraph, together with a set of sources and a target, find a hyperpath consisting of hyperedges from the sources to the target of minimum total weight. We briefly summarize prior work on related problems next.

Related work

Hypergraphs have been studied in the algorithms community [18, 12, 4] and applied within systems biology to metabolic networks [8, 1, 3, 5] and cell-signaling networks [27, 26, 11, 30].

In the field of algorithms, Italiano and Nanni [18] first proved that finding a shortest source-sink hyperpath is NP-complete, even when hyperedges have a single head vertex. In a seminal paper that is the source for much of the subsequent work on hypergraphs, Gallo, Longo, Pallottino and Nguyen [12] explore special cases of hypergraphs, and define several versions of hyperpaths, including what they call a B-path (though see the correction of Nielsen and Pretolani [22]), which is essentially equivalent to our definition of hyperpath in Section 2. They show the vertices reachable from a source vertex in a hypergraph can be found in time linear in the total size of the tail and head sets of all hyperedges, give an efficient algorithm for a variant of shortest hyperpaths with a so-called additive cost function, and prove that finding a minimum cut in a hypergraph is NP-complete. Ausiello and Laura [4] survey results on hypergraphs whose hyperedges have singleton head sets, and note that a consequence of the NP-completeness reduction [18] for shortest hyperpaths from the set cover problem is that, unless $\mathrm{P}=\mathrm{NP}$, no approximation algorithm can exist for shortest hyperpaths on hypergraphs of n vertices with approximation ratio $\big(1-o(1)\big)\ln n$.

In metabolic networks, Cottret, Milreu and Acuña et al. [8] examine the minimum precursor problem: given a hypergraph G, a set of sources S, and a set of targets T, find a source subset $P \subseteq S$ of minimum cardinality that has a hyperpath from P to T. They show this problem is NP-complete, and give an algorithm that enumerates all minimal precursor sets whose hyperpath is acyclic. Acuña, Milreu and Cottret et al. [1] subsequently enumerate all minimal precursor sets allowing cycles. Andrade, Wannagat and Klein et al. [3] extend these algorithms to accommodate stoichiometry and conserve intermediate metabolites within the hyperpath. Carbonell, Fichera, Pandit and Faulon [5] give an efficient algorithm to find a source-sink hyperpath if one exists – irrespective of its length – and prove that finding any hyperpath that must contain a specified set of hyperedges is NP-complete. They also offer an approach to hyperpath enumeration that relies on solutions to this NP-complete problem, for which they employ a heuristic.

In cell-signaling networks, Ritz, Avent and Murali [27, 26] were the first to solve the shortest *acyclic* hyperpath problem by formulating it as a mixed-integer linear program (MILP) – the current state of the art for shortest hyperpaths – and showed that in practice optimal acyclic hyperpaths can be found even for large cell-signaling hypergraphs. Their formulation does not extend to hyperpaths with cycles, and requires exponential time in the worst-case (which may be unavoidable, as the acyclic problem remains NP-complete). Recently, Franzese, Groce, Murali and Ritz [11] defined a parameterized notion of connectivity that interpolates between hyperpath- and ordinary-path-connectivity, while Schwob, Zhan and Dempsey [30] modified the acyclic MILP of Ritz et al. [26] to include time-dependence among reactions.

Our contributions

In contrast to prior work, we give a heuristic for shortest hyperpaths that handles cycles, is worst-case efficient, and finds hyperpaths that are demonstrably optimal or close to optimal in real cell-signaling hypergraphs. In particular, we make the following contributions.

- We present an *efficient heuristic* for shortest hyperpaths, that on a hypergraph of size ℓ, which measures the total cardinality of all hyperedge tail and head sets, with m hyperedges that are doubly-reachable from the source and sink vertices, and k defined analogously to ℓ over these doubly-reachable hyperedges, runs in $O(\ell + m^2\, k)$ time.
- We also give a practical algorithm for *hyperpath enumeration* that generates all possible hyperpaths, allowing us to tractably measure how close our heuristic is to the optimum.
- Our heuristic *matches the state-of-the-art* MILP for shortest acyclic hyperpaths on over 99% of all instances from two standard databases of cell-signaling pathways.
- Our heuristic *surpasses the state-of-the-art* on instances where every source-sink hyperpath is cyclic, and hence the MILP finds no solution. On all such cyclic instances, our hyperpath enumeration algorithm verified that the heuristic was in fact *optimal*.

To our knowledge, this heuristic is the *first in the literature* for shortest source-sink hyperpaths in general directed hypergraphs, where hyperedges have arbitrary tail and head sets, and the length of a hyperpath is the sum of the weights of its hyperedges.

Plan of the paper

The next section defines the shortest source-sink hyperpath problem. Section 3 then presents our heuristic for shortest hyperpaths, and analyzes its time complexity. Section 4 compares the heuristic, through experiments on all source-sink instances from standard databases, to the state-of-the-art MILP for acyclic instances, or to the optimum of all enumerated hyperpaths for cyclic instances, and discusses biological examples of cyclic shortest hyperpaths. Section 5 concludes, and Appendix A gives our algorithm for generating all hyperpaths.

2 Shortest hyperpaths in directed hypergraphs

A directed hypergraph is a generalization of an ordinary directed graph, where an edge, instead of touching two vertices, now connects two subsets of vertices. Formally, a directed *hypergraph* is a pair (V, E), where V is a set of vertices, and E is a set of directed *hyperedges*.[2] Each hyperedge $e \in E$ is an ordered pair (X, Y), where both $X, Y \subseteq V$ are vertex subsets. Edge e is directed from set X to set Y. We call set X the *tail* of e, and set Y the *head*

[2] The literature sometimes uses the term *hyperarc* for an edge in a directed hypergraph, but we prefer the simpler term *hyperedge* (just as the term *edge* is used for both directed and undirected ordinary graphs).

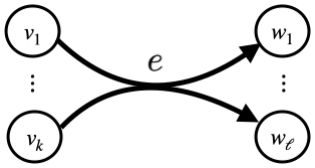


Figure 1 A hyperedge e with $\text{tail}(e) = \{v_1, \ldots, v_k\}$ and $\text{head}(e) = \{w_1, \ldots, w_\ell\}$. To use e in a hyperpath P, every vertex $v_i \in \text{tail}(e)$ must have a preceding hyperedge f on P with $v_i \in \text{head}(f)$.

of e, and refer to these sets by the functions $\text{tail}(e) = X$ and $\text{head}(e) = Y$. We also refer to the *in-edges* of vertex v by $\text{in}(v) := \{e \in E : v \in \text{head}(e)\}$, and the *out-edges* of v by $\text{out}(v) := \{e \in E : v \in \text{tail}(e)\}$. Figure 1 shows a directed hyperedge.

In ordinary directed graphs, a path from a vertex s to a vertex t is a sequence of edges starting from s and ending at t, where for consecutive edges e and f in the sequence, the preceding edge e must enter the vertex that the following edge f leaves. We say t is reachable from s when there is such a path from s to t.

In generalizing these notions to directed hypergraphs, the conditions both for when a hyperedge can follow another in a hyperpath, and when a vertex is reachable from another, become more involved. A hyperpath is again a sequence of hyperedges, but now for hyperedge f in a hyperpath, for every vertex $v \in \text{tail}(f)$, there must be some hyperedge e that precedes f in the hyperpath for which $v \in \text{head}(e)$. Reachability is captured by the following notion of superpath.

▶ **Definition 1** (Superpath). *In a directed hypergraph (V, E), an s, t-superpath, for vertices $s, t \in V$, is an edge subset $F \subseteq E$ such that the hyperedges of F can be ordered $e_1, e_2, \ldots, e_k$, where*

(i) $\text{tail}(e_1) = \{s\}$,
(ii) *for each* $1 < i \leq k$,

$$\text{tail}(e_i) \subseteq \{s\} \cup \bigcup_{1 \leq j < i} \text{head}(e_j),$$

(iii) *and* $t \in \text{head}(e_k)$.

For an s, t-superpath, we call s its source, t its sink, and we say t is reachable from s.

We can now define hyperpaths in terms of superpaths. Recall that a set S is *minimal* with respect to some property X if S satisfies X, but no proper subset of S satisfies X.

▶ **Definition 2** (Hyperpath). *An s, t-hyperpath is a minimal s, t-superpath.*

In other words, a hyperpath P is a superpath for which removing any edge $e \in P$ leaves a subset $P - \{e\}$ that is no longer a superpath. Essentially, hyperpaths eliminate unnecessary edges from superpaths. Figures 5 and 6 later show examples of hyperpaths.

We say a hyperpath P contains a *cycle* if, for every ordering $e_1, \ldots, e_k$ of its hyperedges satisfying properties (i)–(iii) in the definition of superpath, P contains some hyperedge f with a vertex in $\text{head}(f)$ that also occurs in $\text{tail}(e)$ for an earlier hyperedge e in the ordering. While in ordinary graphs a minimal s, t-path can never contain a cycle, in hypergraphs an s, t-hyperpath can in fact contain cycles, as shown in the examples discussed in Section 4.6.

We can now define the shortest hyperpath problem. For an edge weight function $\omega(e)$, we extend ω to edge subsets $F \subseteq E$ by $\omega(F) := \sum_{e \in F} \omega(e)$.

▶ **Definition 3** (Shortest Hyperpaths). *The Shortest Hyperpaths problem is the following. Given a directed hypergraph (V, E), a positive edge weight function $\omega : E \to \mathcal{R}^+$, source $s \in V$ and sink $t \in V$, find an s,t-hyperpath $P \subseteq E$ of minimum total weight $\omega(P)$.*

Note that for positive edge weights, Shortest Hyperpaths is equivalent to finding an s,t-*superpath* of minimum total weight.

Shortest Hyperpaths with a single source and sink vertex also captures more general versions of the problem with *multiple sources* and *multiple sinks*, as follows. To find a hyperpath that starts from a set of sources $S \subseteq V$, simply add a new source vertex s to the hypergraph together with a single hyperedge $(\{s\}, S)$ of zero weight, and equivalently find a hyperpath from the single source s. To find a hyperpath that reaches *all* vertices in a set of sinks $T \subseteq V$, add a new sink vertex t, a zero-weight hyperedge $(T, \{t\})$, and equivalently find a hyperpath to the single sink t. To find a hyperpath that reaches *some* vertex in a set of sinks $T \subseteq V$, add new sink vertex t, zero-weight hyperedges $(\{v\}, \{t\})$ from all $v \in T$, and again equivalently find a hyperpath to the single sink t. Thus versions of shortest hyperpaths with multiple sources and sinks can be reduced to the problem with a single source and sink.

Shortest Hyperpaths is NP-complete [18] (even for acyclic hypergraphs with singleton head sets), so we likely cannot efficiently compute shortest hyperpaths in the worst-case. The next section presents an efficient heuristic for shortest hyperpaths that is highly accurate at finding demonstrably optimal or near-optimal hyperpaths in real cell-signaling hypergraphs.

3 An efficient shortest hyperpath heuristic

We now give a fast heuristic for Shortest Hyperpaths that always finds an s,t-hyperpath if one exists. While the heuristic is not guaranteed to find a shortest s,t-hyperpath, our later experiments on real cell-signaling hypergraphs show it quickly finds a hyperpath that is optimal or remarkably close to optimal on the overwhelming majority of instances from exhaustive experiments over the two standard cell-signaling databases in the literature. We present detailed *pseudocode* for the heuristic at a level that can be directly implemented, as the heuristic is carefully designed and many of its component algorithms are surprisingly tricky to implement correctly. After describing the heuristic, we give a *time analysis* that shows it is always efficient.

The pseudocode accesses a hypergraph G through fields G.vertices and G.edges. We access the tail-set and head-set of a hyperedge edge e through fields e.head and e.tail. We access the set of in-edges and out-edges of a vertex v through fields v.in and v.out. For a list Q that is handled as a queue, operation Q.Put(x) appends item x to the rear of Q, while operation Q.Get() removes and returns the item at the front of Q. For a min-heap H, operation H.Insert(x, k) inserts item x with key k into H, and returns a reference p to the heap node containing this pair (x, k) in H; operation H.Extract() removes and returns the item in H with minimum key; and operation H.Decrease(p, k) takes a reference p to a heap node in H and decreases its key to k if k is smaller than its current key. All functions assume hypergraph G is passed by reference.

The heuristic builds on fast algorithms for computing reachability in a hypergraph, and we discuss these algorithms first. In general, vertex v is *forward reachable* from source s in hypergraph G if there is an s,v-hyperpath in G. A hyperedge e is forward reachable from s if all vertices $v \in \text{tail}(e)$ are forward reachable from s. When e is forward reachable from s, an s,e-*hyperpath* is a hyperpath from s to tail(e) followed by e. A vertex v is *backward traceable* from sink t if $v = t$, or recursively if $v \in \text{tail}(e)$ for an edge e where some $w \in \text{head}(e)$ is backward traceable from t. A hyperedge e is backward traceable from t if some $v \in \text{head}(e)$ is backward traceable from t.

Figure 2 gives pseudocode for the functions `ForwardReachable` and `BackwardTraceable`. Function `ForwardReachable` returns the set of all hyperedges that are forward reachable from source s, while function `BackwardTraceable` returns the set of hyperedges that are backward traceable from sink t. Function `ForwardReachable` uses Boolean vertex field v.reached, and integer edge field e.count, which it assumes have already been initialized to the values v.reached = false for all $v \in V$ and e.count $= \left|\text{tail}(e)\right|$ for all $e \in E$. Function `BackwardTraceable` also uses Boolean edge field e.marked, which it similarly assumes is initialized to false for all e. (This initialization will be done once for hypergraph G in the shortest hyperpath heuristic, which allows these functions when called repeatedly to run in time bounded by just the size of the forward-reachable or backward-traceable subgraphs.) Function `ForwardReachable` uses field e.count to detect when all vertices in tail(e) have been reached from s, and hence e is now reached from s. Function `BackwardTraceable` performs a similar but simpler computation in reverse from sink t. The worst-case time for both these functions is linear in the size of the subgraph they explore, as analyzed in Section 3.1.

Figure 3 gives pseudocode for function `ShortestHyperpathHeuristic`, our heuristic. Like Dijkstra's algorithm for shortest paths in an ordinary graph (see [7, pp. 658–663]), this function maintains a heap H, but in contrast to Dijkstra's algorithm this is now a heap of hyperedges e (rather than a heap of vertices), which are prioritized by keys that are the best known estimate of the length of a shortest s, e-hyperpath. We refer to this estimate as the current *path length* for e. The heuristic starts from the out-edges of source s, and in a reaching computation repeatedly extracts from heap H the hyperedge e with minimum key. When hyperedge e is removed from H, the estimated path length for e is evaluated, and stored in field e.length. To compute this length estimate, it must construct the best s, e-hyperpath it can find, and evaluate its total weight. Of course, computing an optimal s, e-hyperpath is NP-complete, so it uses a greedy heuristic to construct this path by function `RecoverShortPath`. This path-construction heuristic consists of two steps: (1) recovering an s, e-superpath by recursively backward-traversing hyperedges that enter tail(e), and (2) finding a minimal subset of this superpath that is an s, e-hyperpath while attempting to minimize its weight.

Figure 4 gives pseudocode for function `RecoverShortHyperpath` that implements this greedy path-construction heuristic. For the first step, recovering the s, e-superpath S is done by recursively backward-traversing what we call *in-edges*: those hyperedges whose head-sets intersect the tail-set of a given hyperedge. Function `ShortestHyperpathHeuristic` maintains for a hyperedge e the field e.inedges, which stores the subset of the in-edges to e whose length field has been determined.

For the second step, function `RecoverShortHyperpath` attempts to find the minimum weight subset of S that is still a superpath by greedily considering hyperedges $f \in S$ for removal in decreasing order of f.length. This repeatedly tests whether tail(e) is still reachable from s on removing f by calling Boolean function `IsReachable`. Pseudocode for `IsReachable` is not given, but it simply implements a version of function `ForwardReachable` that halts and returns true as soon as it adds e to the set of hyperedges reachable from s, or returns false after collecting the entire reachable set without encountering e.

To summarize, the shortest hyperpath heuristic proceeds greedily like Dijkstra's algorithm, but with some important differences: it maintains a heap of hyperedges prioritized by estimated shortest path lengths to tail-sets, records a set of potential in-edges to a given hyperedge used for recovering a hyperpath to the hyperedge, and recovering such a hyperpath now involves another greedy heuristic to find a minimal superpath of small total weight.

```
function ForwardReachable (s, G) begin    • Find all edges forward-reachable from source s in G
    Create queue Q                        • Initialize a queue of reached vertices
    Q.Put(s)
    s.reached := true
    (F, R) := (∅, ∅)

    while not Q.Empty() do begin          • Process the reached vertices v
        v := Q.Get()
        R ∪:= {v}

        for e ∈ v.out do begin            • Detect which out-edges e of v are now reached
            e.count −:= 1
            if e.count = 0 then begin     • All vertices in tail(e) have been reached, so e is reached
                F ∪:= {e}
                for w ∈ e.head do
                    if not w.reached then begin
                        Q.Put(w)
                        w.reached := true
                    end
            end
        end
    end

    for v ∈ R do begin                    • Restore fields, and return the reachable hyperedges
        v.reached := false
        for e ∈ v.out do e.count +:= 1
    end
    return F
end

function BackwardTraceable (t, G) begin   • Find all edges backward-traceable from sink t in G
    Create queue Q                        • Initialize a queue of reached vertices
    Q.Put(t)
    t.reached := true
    (F, B) := (∅, ∅)

    while not Q.Empty() do begin          • Process the reached vertices v
        v := Q.Get()
        B ∪:= {v}

        for e ∈ v.in do                   • Collect the traceable hyperedges e
            if not e.marked then begin
                F ∪:= {e}
                e.marked := true
                for w ∈ e.tail do
                    if not w.reached then begin
                        Q.Put(w)
                        w.reached := true
                    end
            end
    end

    for v ∈ B do v.reached := false       • Restore fields, and return the traceable hyperedges
    for e ∈ F do e.marked := false
    return F
end
```

Figure 2 Reachability computations. Function `ForwardReachable`, given source vertex s in hypergraph G, returns all hyperedges e for which tail(e) is reachable by a hyperpath from s. Function `BackwardTraceable`, given sink vertex t in G, returns all hyperedges e for which some vertex $v \in$ head(e) is backward-traceable from t. These functions assume fields v.reached, e.marked, and e.count have been initialized to false, false, and $|\text{tail}(e)|$, respectively, for all v and e in G.

```
function ShortestHyperpathHeuristic (s, t, G, ω) begin        • Find a short s,t-hyperpath in G

    for v ∈ G.vertices do                                      • Initialize fields
        (v.reached, v.removed) := (false, false)
    for e ∈ G.edges do
        (e.count, e.marked, e.node, e.inedges) := (|e.tail|, false, nil, ∅)

    D := ForwardReachable(s, G)                                • Restrict G to doubly-reachable edges D
         ∩ BackwardTraceable(t, G)
    Remove from G all edges not in D

    Create min-heap H                                          • Initialize edge heap H
    for e ∈ s.out with e.tail = {s} do
        e.node := H.Insert(e, ω(e))
    s.reached := true

    while not H.Empty() do begin                               • Process reached hyperedges by their path lengths
        e := H.Extract()
        e.removed := true

        P := RecoverShortHyperpath(s, e, G)                    • Recover a short hyperpath to e, and its path length
        e.length := ω(P)

        F := ∅                                                 • Collect for e its out-edges F, and detect which are now reached
        for v ∈ e.head do begin
            for f ∈ v.out do begin
                if not v.reached and not f.marked then
                    f.count −:= 1
                if not f.marked then begin
                    F ∪:= {f}
                    f.marked := true
                end
            end
            v.reached := true
        end
        for f ∈ F do
            f.marked := false

        for f ∈ F do begin                                     • Update path lengths, in-edges, and add reached edges to H
            f.inedges ∪:= {e}
            if f.node ≠ nil and not f.removed then             • Update path length to an edge on H
                H.Decrease(f.node, ω(RecoverShortHyperpath(s, f, G)))
            else if f.node = nil and f.count = 0 then          • Add reached edge to H
                f.node := H.Insert(f, ω(RecoverShortHyperpath(s, f, G)))
        end
    end

    (P*, L*) := (∅, ∞)                                         • Recover the best s,t-hyperpath P*
    for e ∈ t.in do
        if e.node ≠ nil then begin
            P := RecoverShortHyperpath(s, e, G)
            if ω(P) < L* then
                (P*, L*) := (P, ω(P))
        end

    Restore to G all edges not in D                            • Unrestrict G, and return the best hyperpath
    return P*
end
```

Figure 3 Efficient heuristic for shortest source-sink hyperpaths. Given source s, sink t, and edge weights ω, function `ShortestHyperpathHeuristic` finds an s,t-hyperpath in hypergraph G, attempting to minimize its length. If no s,t-hyperpath exists, the empty path is returned. For doubly-reachable hyperedges e, the heuristic maintains fields e.length (the total weight of the shortest hyperpath found to e), and e.inedges (the subset of edges f with head(f) touching tail(e) where f.length is known), which are used in `RecoverShortHyperpath` to recover a short hyperpath to e.

```
function RecoverShortHyperpath (s, e, G) begin      • Recover a short s,e-hyperpath in G

    Create queue Q                                  • Initialize a queue with the in-edges entering e
    for f ∈ e.inedges do begin
        Q.Put(f)
        f.marked := true
    end

    S := {e}                                        • Collect s,e-superpath S, tracing backward from e
    while not Q.Empty() do begin
        f := Q.Get()
        S ∪:= {f}
        for g ∈ f.inedges do
            if not g.marked then begin
                Q.Put(g)
                g.marked := true
            end
    end
    for f ∈ S do
        f.marked := false

    Remove from G all edges not in S                • Greedily construct an s,e-hyperpath P ⊆ S
    S −:= {e}
    P := {e}
    for f ∈ S in decreasing order of f.length do begin
        Remove f from G
        if not IsReachable(s, e, G) then begin
            Restore f back to G
            P ∪:= {f}
        end
    end

    Restore back to G all edges removed             • Restore G, and return hyperpath P
    return P
end
```

Figure 4 Recovering a short hyperpath from the source to a hyperedge. Given source vertex s and hyperedge e, function `RecoverShortHyperpath` returns an s, e-hyperpath P in hypergraph G, attempting to minimize its length. The edges of hyperpath P are greedily selected from an s, e-superpath S that is guaranteed to exist in G, where S is recovered by tracing backward from e.

We note for this heuristic that the inapproximability of the shortest hyperpath problem [4], together with the polynomial time analysis of the following section, imply that unless P = NP, the heuristic cannot be a constant-factor approximation algorithm for shortest hyperpaths.

3.1 Time complexity

We now bound the time complexity of our shortest hyperpath heuristic. To obtain the overall running time of function `ShortestHyperpathHeuristic`, we analyze in turn its component functions `ForwardReachable`, `BackwardTraceable`, and `RecoverShortHyperpath`.

Our analysis is in terms of the following parameters measured on a hypergraph (or an induced subgraph). For a hypergraph G with vertices V and hyperedges E, we denote its number of vertices and edges by $n := |V|$ and $m := |E|$. We also use the *size* parameter

$$\ell \;:=\; \sum_{e \in E} \Big(\big|\mathrm{tail}(e)\big| \;+\; \big|\mathrm{head}(e)\big| \Big) ,$$

and *degree* parameter

$$d \;:=\; \max_{v \in V} \Big\{ \big|\mathrm{in}(v)\big|, \big|\mathrm{out}(v)\big| \Big\} .$$

We assume all tail and head sets are nonempty, and every vertex is touched by a hyperedge, which implies $m + n = O(\ell)$. When we need to refer to these measures for a particular hypergraph G, such as on an induced subgraph, we explicitly subscript the parameters by the specific hypergraph, such as $n_G, \ldots, d_G$, where these parameters are then measured in terms of the vertices and edges of subscripted hypergraph G.

For the reachability computations `ForwardReachable` and `BackwardTraceable`, their running time can be expressed in an output-sensitive way, in terms of the size of the edge sets they return (or equivalently the size of the subgraph of G they explore). For `ForwardReachable`, let $R \subseteq V$ be the set of vertices reachable from source s, and $F \subseteq E$ be the set of hyperedges reachable from s that are returned. The total time for `ForwardReachable` is dominated by the time for its main while-loop, which takes time $\Theta\big(\sum_{v \in R} \big|\mathrm{out}(v)\big| \; + \; \sum_{e \in F} \big|\mathrm{head}(e)\big|\big)$, or equivalently, $\Theta\big(\sum_{e \in E} \big|\mathrm{tail}(e) \cap R\big| \; + \; \sum_{f \in F} \big|\mathrm{head}(f)\big|\big)$. In terms of input hypergraph G, this is $O(\ell_G)$. For `BackwardTraceable`, let $B \subseteq V$ be the set of vertices it reaches from sink t, and $F \subseteq E$ be the set of hyperedges traceable from t that are returned. A similar analysis shows the time for `BackwardTraceable` is $\Theta\big(\sum_{e \in E} \big|\mathrm{head}(e) \cap B\big| \; + \; \sum_{f \in F} \big|\mathrm{tail}(f)\big|\big)$, which is again $O(\ell_G)$. So the time for both `ForwardReachable` and `BackwardTraceable` is $O(\ell_G)$, but can be bounded more tightly in terms of the subgraph they actually explore.

For function `RecoverShortHyperpath`, all its computations are performed on G restricted to edge subset $D \subseteq E$, the doubly-reachable edges that are both forward-reachable from s and backward-traceable from t. Let us call the doubly-reachable subgraph of G induced by D, that `RecoverShortHyperpath` computes over, hypergraph H. The time to collect s, e-superpath S by tracing back from e is at most $O\big(\sum_{f \in S} \sum_{v \in \mathrm{tail}(f)} \big|\mathrm{in}(v)\big|\big)$, which is $O(d_H \, \ell_H)$. The time to greedily construct the s, e-hyperpath $P \subseteq S$, in terms of its cardinality $k = |S|$, is at most $O(m_H + k \log k + k \, \ell_H)$, which is $O(k \, \ell_H)$. Thus the total time for `RecoverShortHyperpath` is $O(d_H \, \ell_H) + O(k \, \ell_H)$, which is $O(\ell_H \, m_H)$.

For function `ShortestHyperpathHeuristic`, we break its time down as follows. The time for the initialization, collecting the doubly-reachable edges D by calling `ForwardReachable` and `BackwardTraceable`, and restricting G to its subgraph H induced by D, is $O(\ell_G)$. The main while-loop executes for m_H iterations, and spends $O(m_H \log m_H)$ time for all Extracts. The total time across all iterations to compute s, e-hyperpath P for all extracted edges e by calling `RecoverShortHyperpath` is $O(\ell_H \, m_H^2)$. The total time to collect the out-edges F for extracted e across all iterations is $O\big(\sum_{e \in D} \sum_{v \in \mathrm{head}(e)} \big|\mathrm{out}(v)\big|\big) = O(d_H \, \ell_H)$. The total time across all iterations for Decrease and Insert, which take $O(1)$ amortized time per edge in F using a Fibonacci heap (see [7, pp. 510–522]), is also $O(d_H \, \ell_H)$. The time to recover the best s, t-hyperpath P^* is $O(d_H \, \ell_H \, m_H)$. Adding up these bounds, the total time for `ShortestHyperpathHeuristic` is $O(\ell_G) + O(m_H \log m_H) + O(\ell_H \, m_H^2) + O(d_H \, \ell_H) + O(d_H \, \ell_H \, m_H)$, which is $O(\ell_G + \ell_H \, m_H^2)$.

Notice that the overall running time of the heuristic is dominated by the total time to recover short hyperpaths, which requires invoking `RecoverShortHyperpath` whenever the path length to a hyperedge is updated. This is necessary in hypergraphs, since in contrast to ordinary graphs the length of the hyperpath to a hyperedge can no longer be expressed as a simple function (like a sum or a minimum) of the lengths of the hyperpaths to its in-edges.

To summarize, the *time complexity* of the shortest s, t-hyperpath heuristic, in terms of the number of hyperedges m and size parameter ℓ, measured on both the input hypergraph G and its restriction to the doubly-reachable subgraph H of hyperedges that are simultaneously forward-reachable from source s and backward-traceable from sink t, is

$$O\Big(\ell_G \; + \; \ell_H \, m_H^2\Big) \, .$$

As demonstrated in Section 4, for real biological instances the size of the doubly-reachable subgraph H is significantly smaller than the full input hypergraph G, so designing the heuristic to compute mainly over H yields a large speedup in practice.

The next section shows this heuristic is remarkably *close to optimal* on real cell-signaling hypergraphs. Given that no practical exact algorithm exists for general shortest hyperpaths, we determine the optimum by enumerating all s,t-hyperpaths, and taking the minimum of their lengths. Appendix A gives our algorithm for generating all source-sink hyperpaths.

4 Experimental results

We now present results from computational experiments on real pathway databases comparing our heuristic to the optimum, study the prevalence of cyclic instances of shortest hyperpaths, report actual running times, and discuss biological examples of cyclic hyperpaths.

4.1 Datasets

We evaluate the quality of our heuristic on four datasets built by combining different annotated signaling pathways from two pathway databases, NCI-PID and Reactome. NCI-PID [29] is a curated human-pathway database containing biochemical reactions for complex assembly, cellular transport, and transcriptional regulation. Reactome [19] also contains curated human signaling pathways, and is actively maintained with new reactions being continuously added. We constructed hypergraphs from three subsets of NCI-PID pathways used in Ritz et al. [28], named the `Small`, `Medium`, and `Large` datasets. The `Small` dataset is a small Wnt signaling pathway consisting of the union of two pathways: "degradation of β-catenin" and "canonical Wnt signaling". The `Medium` dataset is a larger Wnt signaling pathway including four additional pathways: "noncanonical Wnt signaling", "Wnt signaling network", "regulation of nuclear β-catenin", and "presenilin action in Notch and Wnt signaling", which correspond to non-canonical branches of Wnt signaling. The `Large` dataset contains all NCI-PID pathways. Similarly, the `Reactome` dataset is the union of all Reactome pathways. The NCI-PID and Reactome pathways were downloaded in the BioPAX format [10] from Pathway Commons, and processed using a parser from Franzese et al. [11] built on PaxTools [9].

To construct the hypergraphs for each dataset, we mapped each entity (protein, small molecule, and so on) to a vertex in the hypergraph. We parsed each complex as a unique vertex, different from the entities in the complex. Multiple forms of the same protein map to different vertices denoting compartmentalization and post-translational modifications, such as phosphorylation and ubiquitination. We treated each variant as a distinct entity because many pathways show the transportation of a protein from one cellular compartment to another, or a protein being marked for degradation by ubiquitination, so the corresponding vertices need to be distinct to reflect these variants. Each reaction was mapped to a hyperedge, where the reactants and positive regulators comprise the hyperedge tail, and the head contains the products. All hyperedges were given unit weight, even though the heuristic handles weighted edges, as many NCI-PID reactions have no reaction rate.

Table 1 gives statistics on these four signaling hypergraphs. The hypergraphs are very sparse: there are fewer hyperedges than vertices in all four datasets. The hypergraphs from the `Large` and `Reactome` datasets contain respectively 40 and 433 self-loops, showing that many cyclic hyperpaths are likely to exist. However, a small number of these self-loops are unreachable, due to an otherwise unreachable vertex appearing in both the head and tail of the hyperedge. The sources and targets in the experiments are respectively vertices with no in-edges (or whose only in-edge is an unreachable self-loop), and vertices with no out-edges.

Table 1 Dataset Summaries.

	NCI-PID							
Dataset	Small		Medium		Large		Reactome	
Vertices	56		350		9,009		20,458	
Hyperedges	36		228		8,456		11,802	
Pathways	2		6		213		2,516	
Sources	19		138		3,200		8,296	
Targets	10		102		2,636		5,066	
Self-loops	1		8		40		433	
Unreachable self-loops	1		7		14		32	
	mean	max	mean	max	mean	max	mean	max
Tail size	1.8	3	1.9	5	1.9	10	2.4	26
Head size	1.3	3	1.3	4	1.1	5	1.6	28
Forward-reachable set	35	35	192	192	6,169	6,169	4,645	4,645
Backward-traceable set	28	28	49	70	1,198	2,863	4,027	7,021
Doubly-reachable set	27	27	42	60	756	1,836	929	1,725
In-degree	0.8	5	0.8	15	1.0	323	0.9	1,056
Out-degree	1.1	4	1.2	24	1.7	326	1.4	1,167

On average, hyperedges from all four hypergraphs have small head and tail sets, and the average in- and out-degree of vertices is low, reflecting the sparseness of the hypergraphs.

4.2 Experimental setup

To prepare hypergraphs from each dataset for our experiments, we parsed the union of the pathways in the dataset. We then connected a supersource to all source vertices (vertices without in-edges) by a single zero-weight hyperedge. We also connected vertices whose only in-edge was a self-loop to the supersource, as otherwise the hyperedge was not traversable. For each target (a vertex without out-edges), we connected the target to the sink by a zero-weight ordinary directed graph edge, giving us a target instance. Note that the supersource is the same for each instance. These vertices are reasonable choices for sources and targets, as they are the molecules at which biologists stopped annotating a given pathway.

For each target instance, we trimmed the hypergraph to the doubly-reachable set: the set of hyperedges that were both forward-reachable from the source, and backward-traceable from the sink. Table 1 gives the average and maximum size of the forward-reachable, backward-traceable, and doubly-reachable sets over all instances for a given dataset, which dramatically decrease the size of the hypergraph. For each instance, we found a hyperpath from the supersource to the sink using our shortest hyperpath heuristic, and compared its length to the solution of the MILP of Ritz et al. [26] if the heuristic hyperpath was acyclic. For each instance with a cyclic heuristic hyperpath, we exhaustively enumerated all supersource-to-sink hyperpaths, and compared the heuristic hyperpath to the shortest hyperpath. Enumerating all hyperpaths takes many hours in practice, and is not feasible to compute for all target instances.

4.3 Abundance of cyclic hyperpaths

Cyclic shortest hyperpaths appear in all four datasets. To take one example, in the `Small` and `Medium` datasets, the shortest – and only – hyperpath from ubiquitinated β-catenin to APC is cyclic, so for this target instance the acyclic shortest hyperpath MILP returns no

Table 2 Acyclic Instance Summaries.

	NCI-PID			
Dataset	Small	Medium	Large	Reactome
Target instances	10	102	2,636	5,066
Reachable instances	10	90	2,220	2,432
Acyclic instances	9	89	2,182	2,210
Heuristic optimal	100% (9)	100% (89)	99% (2,159)	100% (2,210)

hyperpath. This particular source-target pair is specially selected to create a cyclic shortest hyperpath, as ubiquitinated β-catenin has an in-edge and APC has an out-edge, so they would not normally be considered sources and targets under our definition. Nevertheless, this demonstrates there are cyclic hyperpaths in the NCI-PID database, even in the union of just two pathways, that are missed by the state of the art when computing only acyclic shortest hyperpaths.

In the Large dataset, 38 instances have cyclic heuristic hyperpaths. Of these, 22 were cyclic because of a self-loop, and 16 were cyclic due to a non-trivial cycle. For all these instances, no acyclic hyperpath exists between the supersource and the sink. It is likely that even more cycles exist within the hypergraph from the Large dataset, as there were 8 self-loops that were not on any hyperpath found by the heuristic.

In the Reactome dataset, we found 22 targets with cyclic shortest hyperpaths, where only one of these targets was cyclic due to a self-loop. Reactome is much sparser than NCI-PID, and 432 of the 433 self-loops in Reactome are never used in a heuristic hyperpath.

The abundance of cyclic hyperpaths in the NCI-PID and Reactome datasets demonstrates the importance of a shortest hyperpath algorithm that properly handles cycles. We discuss two examples of these cyclic shortest hyperpaths later in Section 4.6.

4.4 Quality of the hyperpath heuristic

To determine the quality of our hyperpath heuristic, we compared the length of the heuristic hyperpath to the optimal shortest hyperpath. On the target instances with cyclic heuristic hyperpaths, there is no practical exact algorithm for finding the shortest hyperpath. Therefore, to find the optimum we generated all source-sink hyperpaths and kept the shortest one, using the enumeration algorithm presented in Appendix A. On target instances where the heuristic returned an acyclic hyperpath, we compared its length to the hyperpath returned by the MILP for shortest acyclic hyperpaths. It is possible that a shorter cyclic hyperpath exists, but enumerating all hyperpaths is not practical for all instances.

Table 2 summarizes the quality of the heuristic on acyclic instances. On the Small, Medium, and Reactome datasets, the heuristic hyperpath is optimal on all target instances, meaning the heuristic hyperpath and the shortest acyclic hyperpath from the MILP have the same length. On the Large dataset, the heuristic is optimal on over 99% of the instances, demonstrating the quality of our heuristic on these biological datasets. On the small fraction of instances where our heuristic was suboptimal, the maximum length difference between the heuristic hyperpath and the MILP hyperpath was 6, while the median difference was 1.

Table 3 summarizes the quality of the heuristic on instances where it returned a cyclic hyperpath. On all these cyclic instances, the acyclic MILP returned no hyperpath, so we could not compare the heuristic to an optimal shortest hyperpath other than by exhaustively enumerating all hyperpaths and picking the shortest. Each cyclic instance from the Reactome dataset had many enumerated hyperpaths, with an average of 25.6 hyperpaths enumerated

Table 3 Cyclic Instance Summaries.

	NCI-PID							
Dataset	Small		Medium		Large		Reactome	
Target instances	10		102		2,636		5,066	
Reachable instances	10		90		2,220		2,432	
Cyclic instances	1		1		38		22	
Heuristic optimal	100%		100%		100%		100%	
Non-trivial cycles	1		1		22		21	
	mean	max	mean	max	mean	max	mean	max
Number of paths*	1	1	1	1	49.8	364	25.6	136
Path length range†	0	0	0	0	9.2	43	7.3	15

*Total number of hyperpaths for a cyclic target instance
†Difference between the length of the longest hyperpath and the heuristic hyperpath

per instance and a maximum of 136 hyperpaths. The hyperpaths also tended to have different lengths with a maximum difference between the heuristic hyperpath length and the longest enumerated hyperpath length of 15, with an average of 7.3. The cyclic instances from the `Large` dataset had even more alternate hyperpaths, with an average of 49.8 hyperpaths per instance, and a maximum of 364. These instances also had an even greater range of hyperpath lengths, with an average difference of 9.2, and one instance where the difference between the longest hyperpath and the shortest was 43 hyperedges. This demonstrates that the heuristic is discriminating between hyperpaths of different lengths and choosing the shortest hyperpath over worse hyperpaths, indicating the quality of the heuristic. For all cyclic target instances, the enumerated hyperpaths were cyclic, and many shared the same cycle, with most of their differences occurring in hyperedges outside this cycle.

4.5 Implementation and running time

The heuristic is implemented in Python 2.7.3, comprising around 500 lines of code. The parser used to convert the BioPAX format into hypergraphs is from [11]. For directed hypergraph representations and reachability calculations, we used Halp (`github.com/Murali-group/halp/`). All heuristic and enumeration source code is available at `hyperpaths.cs.arizona.edu`.

Experiments were run on a laptop with a 2.9 GHz Intel Core i5 CPU, and 16 GB of RAM. The running time of the hyperpath heuristic was 55 seconds on average for the instances from the `Large` and `Reactome` datasets, which have just under 1,000 hyperedges on average. Memory usage was low, taking less than 2 GB of memory for the heuristic.

4.6 Biological examples

We now discuss two instances with cyclic shortest hyperpaths from the `Large` and `Reactome` datasets. The hyperpath found by our heuristic for these two instances is optimal (as was the case for all instances where the heuristic found a cyclic path), and is drawn in Figures 5 and 6. We describe the hypergraph structure and constituent reactions for each instance.

Assembly of the JUP/DSP complex

The first example captures the assembly of the JUP/DSP complex from the `Large` dataset. Figure 5 shows the shortest hyperpath returned by our heuristic with the JUP/DSP complex as the target. All vertices at the top of the figure are connected to the supersource.

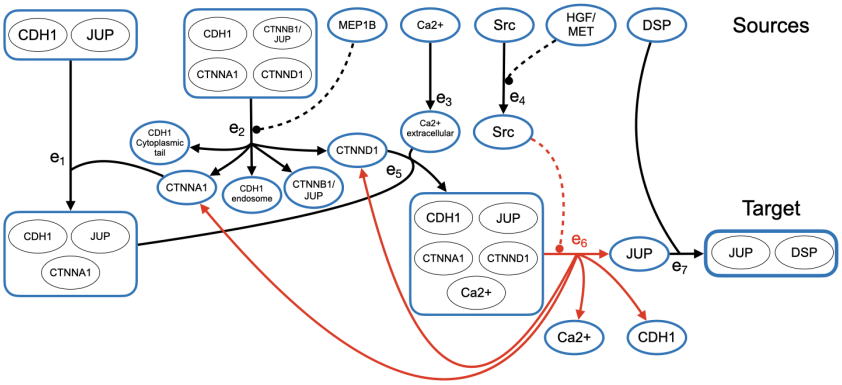


Figure 5 The cyclic shortest hyperpath from the supersource to the JUP/DSP complex in the `Large` dataset. All vertices in the hyperpath connected to the supersource are shown at the top of the figure. The hyperedges in this hyperpath come from four different pathways, and show the different complexes JUP participates in until finally being free to bind with desmoplakin (DSP). Hyperedge e_6, shown in red, creates two separate cycles back to α-catenin and δ-catenin.

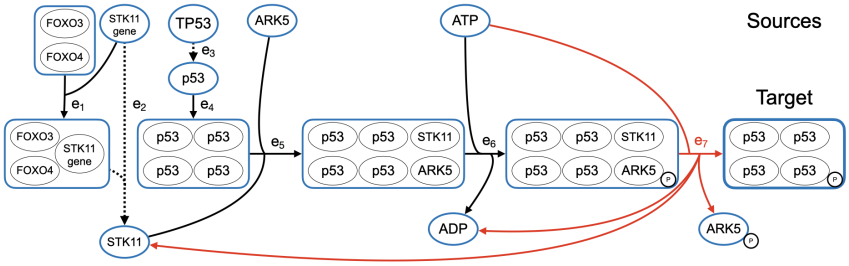


Figure 6 The cyclic shortest hyperpath from the supersource to phosphorylated p53 in the `Reactome` dataset. All vertices in the hyperpath connected to the supersource are shown at the top of the figure. The hyperedges in this hyperpath show the transcription of STK11 and p53 (TP53) before NUAK1 (ARK5) participates in the phosphorylation of the p53 tetramer. Hyperedge e_7, shown in red, creates a cycle when the phosphorylation of p53 breaks up a complex, returning STK11 to its solitary state. Hyperedges e_2 and e_3 show transcription, and are drawn dotted.

This hyperpath includes seven hyperedges from four different NCI-PID pathways: "E-cadherin signaling in the nascent adherens junction" (hyperedges e_1 and e_5), "Posttranslational regulation of adherens junction stability and dissassembly" (hyperedges e_2, e_6 and e_7), "Signaling events mediated by PRL" (hyperedge e_3), and "Signaling events mediated by hepatocyte growth factor receptor (c-Met)" (hyperedge e_4). We briefly describe the key events in this hyperpath. Protein γ-catenin (also known as junction plakoglobin or JUP) is initially complexed with Cadherin 1 (CDH1) in the tail of hyperedge e_1. In hyperedge e_2, the metalloprotease meprinβ cleaves E-cadherin (CDH1), releasing it from its complex with α-catenin (CTNNA1) and δ-catenin (CTNND1) [17]. The CDH1/JUP complex adds α-catenin (CTNNA1, in hyperedge e_1) and CTNND1 and Ca^{2+} (in hyperedge e_5) to form a five-member complex. Hepatocyte growth factor (HGF) activates the proto-oncogene tyrosine-protein kinase Src (hyperedge e_4) [23]. Src regulates the breakup of this complex into its individual components [21] (hyperedge e_6), freeing JUP to bind with DSP and creating the two cycles in this hyperpath via CTNNA1 and CTNNB1. The hyperpath culminates in the formation of a complex between desmoplasmin (DSP) and JUP.

The hypergraph for this instance is large, with 6,168 forward-reachable hyperedges, 2,642 backward-traceable hyperedges, and 1,665 doubly-reachable hyperedges. There is no acyclic hyperpath from the supersource to JUP/DSP. When enumerating all s, t-hyperpaths for this instance, there were 16 alternate hyperpaths, and the longest hyperpath had 3 more hyperedges than the heuristic path, which was verified to be optimal.

Phosphorylation of p53

The second example captures the phosphorylation of p53 by NUAK1 (ARK5) from the `Reactome` dataset. The heuristic hyperpath, which is optimal, is shown in Figure 6. All of the vertices at the top are connected to the supersource.

Hyperedge e_1 shows the complex formation of FOXO3 and FOXO4 with the STK11 gene, allowing for the transcription of the gene in hyperedge e_2. Hyperedges e_3 and e_4 deal with the transcription of protein p53 (TP53), and its formation into a homotetramer. The p53 tetramer then forms a complex with NUAK1 (ARK5) and STK11 in hyperedge e_5, allowing for the phosphorylation of NUAK1 via ATP in hyperedge e_6. Once NUAK1 is phosphorylated, it directly phosphorylates p53 [15], activating it and allowing it to assist in DNA damage repair. The final hyperedge e_7, shown in red, breaks apart the p53 tetramer/NUAK1/STK11 complex, resulting in a cycle of free STK11. This hyperpath features two transcriptional hyperedges e_2 and e_3, shown dotted.

This example from `Reactome` is slightly smaller than the example from the `Large` dataset, with only 4,645 forward-reachable edges, 7,021 backward-traceable edges, and 1,632 hyperedges in the doubly-reachable set. There was no acyclic hyperpath for this instance. In contrast to the first example, no alternate hyperpaths to the target exist in the hypergraph.

5 Conclusion

We have presented the first heuristic for Shortest Hyperpaths in general directed hypergraphs with positive edge weights, where the length of a hyperpath is the sum of the weights of its hyperedges. The heuristic handles cycles, is guaranteed to be efficient, and is highly accurate in practice. It matches the state-of-the-art mixed-integer linear program for shortest acyclic hyperpaths on over 99% of all instances from the NCI-PID and Reactome databases, and surpasses the state-of-the-art on all instances where no acyclic hyperpath exists. Moreover, exhaustive enumeration of all source-sink hyperpaths demonstrates that on every cyclic instance from these databases, the heuristic was provably optimal.

Further research

Given that we can quickly find hyperpaths that are close to optimal on real cell-signaling hypergraphs, several research directions beckon. While the inapproximability of Shortest Hyperpaths [4] rules out a constant-factor approximation unless P = NP, is there an approximation algorithm whose *approximation ratio* on hypergraphs with n vertices matches the theoretical lower bound of $\ln n$? More practically, given that in our experiments our heuristic was suboptimal only on acyclic instances, is there a fast method for *acyclic hyperpaths* that outperforms our heuristic? Since a user would like to know how close to optimal a computed hyperpath is for their particular input graph, is there an efficient heuristic that, as well as giving an upper bound on the optimum through its hyperpath, also outputs a *lower bound* on the length of the shortest hyperpath? Many intriguing research avenues are open.

References

1 Vicente Acuña, Paulo Vieira Milreu, Ludovic Cottret, Alberto Marchetti-Spaccamela, Leen Stougie, and Marie-France Sagot. Algorithms and complexity of enumerating minimal precursor sets in genome-wide metabolic networks. *Bioinformatics*, 28(19):2474–2483, July 2012.

2 Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*. Garland Science, New York, 2007.

3 Ricardo Andrade, Martin Wannagat, Cecilia C. Klein, Vicente Acuña, Alberto Marchetti-Spaccamela, Paulo V. Milreu, Leen Stougie, and Marie-France Sagot. Enumeration of minimal stoichiometric precursor sets in metabolic networks. *Algorithms for Molecular Biology*, 11(1):25, 2016.

4 Giorgio Ausiello and Luigi Laura. Directed hypergraphs: introduction and fundamental algorithms—a survey. *Theoretical Computer Science*, 658:293–306, 2017.

5 Pablo Carbonell, Davide Fichera, Shashi B. Pandit, and Jean-Loup Faulon. Enumerating metabolic pathways for the production of heterologous target chemicals in chassis organisms. *BMC Systems Biology*, 6(1):10, 2012.

6 Tobias S. Christensen, Ana P. Oliveira, and Jens Nielsen. Reconstruction and logical modeling of glucose repression signaling pathways in *Saccharomyces cerevisiae*. *BMC Systems Biology*, 3(1):7, 2009.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 3rd edition, 2009.

8 Ludovic Cottret, Paulo Vieira Milreu, Vicente Acuña, Alberto Marchetti-Spaccamela, Fábio Viduani Martinez, Marie-France Sagot, and Leen Stougie. Enumerating precursor sets of target metabolites in a metabolic network. In *Proceedings of the 8th Workshop on Algorithms in Bioinformatics*, pages 233–244, 2008.

9 Emek Demir, Özgün Babur, Igor Rodchenkov, Bülent Arman Aksoy, Ken I Fukuda, Benjamin Gross, Onur Selçuk Sümer, Gary D Bader, and Chris Sander. Using biological pathway data with Paxtools. *PLoS Computational Biology*, 9(9):e1003194, 2013.

10 Emek Demir, Michael P. Cary, and Suzanne Paley et al. The BioPAX community standard for pathway data sharing. *Nature Biotechnology*, 28(9):935–942, 2010.

11 Nicholas Franzese, Adam Groce, T.M. Murali, and Anna Ritz. Hypergraph-based connectivity measures for signaling pathway topologies. *PLoS Computational Biology*, 15(10):1–26, October 2019.

12 Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2–3):177–201, 1993.

13 H.W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985.

14 Lenwood S. Heath and Allan A. Sioson. Semantics of multimodal network models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):271–280, 2009.

15 X Hou, J-E Liu, W Liu, C-Y Liu, Z-Y Liu, and Z-Y Sun. A new role of NUAK1: directly phosphorylating p53 and regulating cell proliferation. *Oncogene*, 30(26):2933–2942, June 2011.

16 Zhenjun Hu, Joe Mellor, Jie Wu, Minoru Kanehisa, Joshua M. Stuart, and Charles DeLisi. Towards zoomable multidimensional maps of the cell. *Nature Biotechnology*, 25(5):547–554, 2007.

17 Maya Huguenin, Elaine J. Müller, Sandra Trachsel-Rösmann, Beatrice Oneda, Daniel Ambort, Erwin E. Sterchi, and Daniel Lottaz. The metalloprotease meprinbeta processes E-cadherin and weakens intercellular adhesion. *PLoS One*, 3(5):e2153, May 2008.

18 Giuseppe F. Italiano and Umberto Nanni. Online maintenance of minimal directed hypergraphs. Technical report, Department of Computer Science, Columbia University, 1989.

19 G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G.R. Gopinath, G.R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein. Reactome: a knowledgebase of biological pathways. *Nucleic Acids Research*, 33:D428–432, 2005.

20 Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5):e1000385, 2009.

21 Susana Miravet, José Piedra, Julio Castaño, Imma Raurell, Clara Francì, Mireia Duñach, and Antonio García de Herreros. Tyrosine phosphorylation of plakoglobin causes contrary effects on its association with desmosomes and adherens junction components and modulates β-catenin-mediated transcription. *Molecular and Cellular Biology*, 23(20):7391–7402, 2003.

22 Lars Relund Nielsen and Daniele Pretolani. A remark on the definition of a B-hyperpath. Technical report, Department of Operations Research, University of Aarhus, 2001.

23 Felipe Palacios, Jogender S. Tushir, Yasuyuki Fujita, and Crislyn D'Souza-Schorey. Lysosomal targeting of E-cadherin: a unique mechanism for the down-regulation of cell-cell adhesion during epithelial to mesenchymal transitions. *Molecular and Cellular Biology*, 25(1):389–402, 2005.

24 Emad Ramadan, Sudhir Perincheri, and David Tuck. A hyper-graph approach for analyzing transcriptional networks in breast cancer. In *Proceedings of the 1st ACM Conference on Bioinformatics and Computational Biology*, pages 556–562, 2010.

25 Emad Ramadan, Arijit Tarafdar, and A. Pothen. A hypergraph model for the yeast protein complex network. In *Proceedings of the 18th Parallel and Distributed Processing Symposium*, pages 189–196, 2004.

26 Anna Ritz, Brendan Avent, and T.M. Murali. Pathway analysis with signaling hypergraphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 14(5):1042–1055, 2017.

27 Anna Ritz and T.M. Murali. Pathway analysis with signaling hypergraphs. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 249–258, 2014.

28 Anna Ritz, Allison N. Tegge, Hyunju Kim, Christopher L. Poirel, and T.M. Murali. Signaling hypergraphs. *Trends in Biotechnology*, 32(7):356–362, 2014.

29 Carl F. Schaefer, Kira Anthony, Shiva Krupa, Jeffrey Buchoff, Matthew Day, Timo Hannay, and Kenneth H. Buetow. PID: the Pathway Interaction Database. *Nucleic Acids Research*, 37:D674–679, 2009.

30 Michael R. Schwob, Justin Zhan, and Aeren Dempsey. Modeling cell communication with time-dependent signaling hypergraphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, to appear 2019.

31 Roded Sharan and Trey Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427–433, 2006.

32 Marc Vidal, Michael E. Cusick, and Albert-Lásló Barabási. Interactome networks and human disease. *Cell*, 144(6):986–998, 2011.

33 Wanding Zhou and Luay Nakhleh. Properties of metabolic graphs: biological organization or representation artifacts? *BMC Bioinformatics*, 12(1):132, 2011.

A Appendix: Generating all source-sink hyperpaths

In this appendix, we give our algorithm for tractably generating all the hyperpaths in a hypergraph from a fixed source to a fixed sink.

The technique of inclusion and exclusion of Hamacher and Queyranne [13] provides a general method for generating all the solutions to any combinatorial optimization problem whose feasible solutions are subsets of a ground set – where in our context, hyperpaths are subsets of hyperedges from a hypergraph – but it relies on the ability to efficiently compute a feasible solution that is constrained to include a given *in-set* and exclude a given *out-set*. Interestingly, for hyperpaths, Carbonell et al. [5] have shown that just determining whether an s,t-hyperpath exists that contains a specified in-set of hyperedges (regardless of the length of the hyperpath) is already NP-complete. Consequently, we cannot generate all s,t-hyperpaths using the standard inclusion-exclusion technique, as we cannot tractably solve the resulting subproblem that has both in- and out-set constraints.

```
function AllHyperpaths (s, t, G) begin                  • Generate all s,t-hyperpaths in G

    Create queue Q                  • Initialize a queue of subproblems, and a set A of hyperpaths
    Q.Put((∅, ∅))
    A := ∅

    while not Q.Empty() do begin                        • Process all subproblems on the queue
        (Out, Keep) := Q.Get()

        P := OneHyperpath(s, t, Out, G)     • Find an s,t-hyperpath excluding edges in Out
        if P ≠ ∅ and P ∉ A then begin
            A ∪:= {P}                                   • Save the new hyperpath

            K := Keep                                   • Add all child subproblems to the queue
            for e ∈ P with e ∉ Keep do begin
                Q.Put((Out ∪ {e}, K))           • Children cannot exclude edges in Keep ...
                K ∪:= {e}                       • ... or edges excluded by prior siblings
            end
        end
    end

    return A                                            • Return the set A of all hyperpaths
end
```

Figure 7 Generating all source-sink hyperpaths. Function `AllHyperpaths`, given source vertex s, sink vertex t, and hypergraph G, returns the set of all s,t-hyperpaths in G. It calls a function `OneHyperpath` that returns an s,t-hyperpath not containing any hyperedge from a specified set Out, and which returns the empty path if no such hyperpath exists.

Instead, we generate all hyperpaths through a simple and practical algorithm that only involves out-sets, given in Figure 7. Function `AllHyperpaths` returns a list of all s,t-hyperpaths in hypergraph G, leveraging a function `OneHyperpath` that just has to return one s,t-hyperpath P in G that does not contain any hyperedges from set Out (so $P \cap \text{Out} = \emptyset$), or determine that no such hyperpath exists. This constrained hyperpath problem with only out-sets is easy to solve: remove all hyperedges in set Out from G, collect all vertices R and hyperedges F reachable from s in this reduced hypergraph, and if $t \in R$, then find any minimal subset $P \subseteq F$ in which t is still reachable from s; otherwise if $t \notin R$, no such hyperpath exists. Function `OneHyperpath` can efficiently find such an s,t-hyperpath P excluding set Out using repeated calls to `ForwardReachable` (given earlier in Figure 2).

Function `AllHyperpaths` uses a queue of subproblems. A subproblem is described by a pair (Out, Keep), which corresponds to finding an s,t-hyperpath excluding Out, where any subsequent subproblems that arise from this given subproblem must not exclude any hyperedges from set Keep (though their solutions are not required to actually use edges from Keep). The purpose of this set Keep is to ensure that all subproblems ever placed on the queue have distinct Out sets. (So any given subproblem described by an out-set is only ever solved once.) A subproblem that arises from a given one we call a *child* subproblem (as the entire collection of subproblems conceptually forms a tree that is explored breadth-first using the queue). Each child subproblem excludes one edge from the hyperpath found for its parent subproblem; in this way, the children will generate hyperpaths that are distinct from their parent hyperpath, if they have a solution. (Once a subproblem becomes infeasible due to its out-set eliminating any s,t-hyperpath as a solution, it also does not generate further subproblems.) Though the whole approach never repeatedly solves the same subproblem, in contrast to the inclusion-exclusion technique it can generate the same hyperpath from different subproblems, so we check whether hyperpath P is distinct from those already found before adding it to the list $\mathcal{A}$ of all hyperpaths.

We bound the running time for `AllHyperpaths` as follows. Let m be the number of hyperedges in hypergraph G, and ℓ be the size parameter for G defined in Section 3.1. Solving a given subproblem by `OneHyperpath` involves at most m calls to `ForwardReachable`, which takes $O(\ell\, m)$ time. Suppose `AllHyperpaths` terminates after solving k subproblems from the queue. Its total time is then $O(k\, \ell\, m)$.

In the worst-case, this is $O(2^m\, \ell\, m)$ time, since the out-sets of subproblems are all distinct. In practice, though, typically $k \ll 2^m$, so the running time is much faster than the worst-case suggests. Function `AllHyperpaths` can tractably generate all source-sink hyperpaths for large hypergraphs, as shown in Section 4, since many of its subproblems quickly become infeasible for real cell-signaling networks.