

19th International Workshop on Algorithms in Bioinformatics

WABI 2019, September 8–10, 2019, Niagara Falls, NY, USA

Edited by

Katharina T. Huber

Dan Gusfield



Editors

Katharina T. Huber

University of East Anglia, Norwich, UK
K.Huber@uea.ac.uk

Dan Gusfield

University of California, Davis, California, USA
dmgusfield@ucdavis.edu

ACM Classification 2012

Applied computing → Bioinformatics; Theory of computation → Design and analysis of algorithms;
Mathematics of computing → Probabilistic inference problems; Computing methodologies → Machine
learning approaches

ISBN 978-3-95977-123-8

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/978-3-95977-123-8>.

Publication date

September, 2019

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>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):
<https://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work
under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.WABI.2019.0

ISBN 978-3-95977-123-8

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.

Editorial Board

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Katharina T. Huber and Dan Gusfield</i>	0:vii
Building a Small and Informative Phylogenetic Supertree	
<i>Jesper Jansson, Konstantinos Mampentzidis, and Sandhya T. P.</i>	1:1–1:14
Alignment- and Reference-Free Phylogenomics with Colored de Bruijn Graphs	
<i>Roland Wittler</i>	2:1–2:14
Quantified Uncertainty of Flexible Protein-Protein Docking Algorithms	
<i>Nathan L. Clement</i>	3:1–3:12
TRACTION: Fast Non-Parametric Improvement of Estimated Gene Trees	
<i>Sarah Christensen, Erin K. Molloy, Pranjal Vachaspati, and Tandy Warnow</i>	4:1–4:16
Better Practical Algorithms for rSPR Distance and Hybridization Number	
<i>Kohei Yamada, Zhi-Zhong Chen, and Lusheng Wang</i>	5:1–5:12
pClay: A Precise Parallel Algorithm for Comparing Molecular Surfaces	
<i>Georgi D. Georgiev, Kevin F. Dodd, and Brian Y. Chen</i>	6:1–6:13
Read Mapping on Genome Variation Graphs	
<i>Kavya Vaddadi, Rajgopal Srinivasan, and Naveen Sivadasan</i>	7:1–7:17
Finding All Maximal Perfect Haplotype Blocks in Linear Time	
<i>Jarno Alanko, Hideo Bannai, Bastien Cazaux, Pierre Peterlongo, and Jens Stoye</i>	8:1–8:9
A New Paradigm for Identifying Reconciliation-Scenario Altering Mutations Conferring Environmental Adaptation	
<i>Roni Zoller, Meirav Zehavi, and Michal Ziv-Ukelson</i>	9:1–9:13
Jointly Embedding Multiple Single-Cell Omics Measurements	
<i>Jie Liu, Yuanhao Huang, Ritambhara Singh, Jean-Philippe Vert, and William Stafford Noble</i>	10:1–10:13
Inferring Diploid 3D Chromatin Structures from Hi-C Data	
<i>Alexandra Gesine Cauer, Gürkan Yardımcı, Jean-Philippe Vert, Nelle Varoquaux, and William Stafford Noble</i>	11:1–11:13
Consensus Clusters in Robinson-Foulds Reticulation Networks	
<i>Alexey Markin and Oliver Eulenstein</i>	12:1–12:12
Weighted Minimum-Length Rearrangement Scenarios	
<i>Pijus Simonaitis, Annie Chateau, and Krister M. Swenson</i>	13:1–13:17
Fast and Accurate Structure Probability Estimation for Simultaneous Alignment and Folding of RNAs	
<i>Milad Miladi, Martin Raden, Sebastian Will, and Rolf Backofen</i>	14:1–14:13
Context-Aware Seeds for Read Mapping	
<i>Hongyi Xin, Mingfu Shao, and Carl Kingsford</i>	15:1–15:13
Bounded-Length Smith–Waterman Alignment	
<i>Alexander Tiskin</i>	16:1–16:12

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Validating Paired-End Read Alignments in Sequence Graphs <i>Chirag Jain, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru</i>	17:1–17:13
Detecting Transcriptomic Structural Variants in Heterogeneous Contexts via the Multiple Compatible Arrangements Problem <i>Yutong Qiu, Cong Ma, Han Xie, and Carl Kingsford</i>	18:1–18:5
Faster Pan-Genome Construction for Efficient Differentiation of Naturally Occurring and Engineered Plasmids with Plaster <i>Qi Wang, R. A. Leo Elworth, Tian Rui Liu, and Todd J. Treangen</i>	19:1–19:12
Rapidly Computing the Phylogenetic Transfer Index <i>Jakub Truszkowski, Olivier Gascuel, and Krister M. Swenson</i>	20:1–20:12
Empirical Performance of Tree-Based Inference of Phylogenetic Networks <i>Zhen Cao, Jiafan Zhu, and Luay Nakhleh</i>	21:1–21:13
A Combinatorial Approach for Single-cell Variant Detection via Phylogenetic Inference <i>Mohammadamin Edrisi, Hamim Zafar, and Luay Nakhleh</i>	22:1–22:13
Topological Data Analysis Reveals Principles of Chromosome Structure in Cellular Differentiation <i>Natalie Sauerwald, Yihang Shen, and Carl Kingsford</i>	23:1–23:16
Synteny Paths for Assembly Graphs Comparison <i>Evgeny Polevikov and Mikhail Kolmogorov</i>	24:1–24:14

■ Preface

This proceedings volume contains papers presented at the 19th Workshop on Algorithms in Bioinformatics (WABI 2019), which was held in Niagara Falls, New York, USA, September 8–10, 2019. WABI 2019 was held together with the 10th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB).

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 workshop is intended as a forum for 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 data-sets. The meeting's focus is on recent research results, including significant work-in-progress, as well as identifying and exploring directions of future research.

In 2019, a total of 47 manuscripts were submitted to WABI from which 24 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 24 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.

We thank all the authors of submitted papers and the members of the WABI Program Committee and their reviewers for their efforts that made this conference possible. We are also grateful to the WABI Steering Committee for their help and advice. We thank all the conference participants and speakers who contributed to a great scientific program. In particular, we are indebted to the keynote speaker of the conference, Nadia El-Mabrouk, for her presentation. WABI 2019 is grateful for the support of the sponsors of ACM BCB 2019, The ACM, and SIGBio. We thank Letu Qingge for setting up the WABI webpage, Michael Wagner for his assistance with putting together the WABI conference proceedings, and the ACM BCB Organizing Committee, especially Xinghua (Mindy) Shi for her efforts to coordinate WABI and ACM-BCB 2019. Finally, we also thank Dong Si, Michael Buck and Pierangelo Veltri for their hard work in making all the local arrangements to ensure an exciting and successful WABI and ACM BCB.



■ List of Authors


- Jarno Alanko  (8)
Department of Computer Science, University of Helsinki, Finland
- Srinivas Aluru (17)
School of Computational Science and Engineering, Georgia Institute of Technology, USA
- Rolf Backofen  (14)
Bioinformatics Group, Department of Computer Science, University of Freiburg, Germany; Signalling Research Centres BIOS and CIBSS, University of Freiburg, Germany
- Hideo Bannai  (8)
Department of Informatics, Kyushu University, Japan
- Zhen Cao (21)
Department of Computer Science, Rice University, 6100 Main Street, Houston, TX 77005, USA
- Alexandra Gesine Cauer (11)
Department of Genome Sciences, University of Washington, Seattle, WA, USA
- Bastien Cazaux  (8)
Department of Computer Science, University of Helsinki, Finland
- Annie Chateau (13)
LIRMM - Université Montpellier, France
- Brian Y. Chen (6)
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
- Zhi-Zhong Chen (5)
Division of Information System Design, Tokyo Denki University, Japan
- Sarah Christensen  (4)
University of Illinois at Urbana-Champaign, USA
- Nathan L. Clement (3)
Department of Computer Science, University of Texas at Austin, USA
- Alexander Dilthey (17)
Institute of Medical Microbiology, University Hospital of Düsseldorf, Germany
- Kevin F. Dodd (6)
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
- Mohammadamin Edrisi (22)
Department of Computer Science, Rice University, Houston, TX, USA
- R. A. Leo Elworth  (19)
Department of Computer Science, Rice University, Houston, TX 77005, USA
- Oliver Eulenstein (12)
Department of Computer Science, Iowa State University, Ames, IA, USA
- Olivier Gascuel (20)
Unité Bioinformatique Evolutive, Département de Biologie Computationnelle, USR 3756, Institut Pasteur et CNRS, Paris, France; LIRMM, CNRS, Université Montpellier, Montpellier, France
- Georgi D. Georgiev (6)
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
- Yuanhao Huang (10)
Department of Computational Medicine and Bioinformatics, University of Michigan, Ann Arbor, MI, USA
- Chirag Jain (17)
School of Computational Science and Engineering, Georgia Institute of Technology, USA
- Jesper Jansson (1)
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
- Carl Kingsford (15, 18, 23)
Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
- Mikhail Kolmogorov  (24)
Department of Computer Science and Engineering, University of California, San Diego, CA, USA
- Jie Liu (10)
Department of Computational Medicine and Bioinformatics, University of Michigan, Ann Arbor, MI, USA

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).
Editors: Katharina T. Huber and Dan Gusfield



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Tian Rui Liu  (19)
Department of Computer Science, Rice
University, Houston, TX 77005, USA
- Cong Ma (18)
Computational Biology Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, PA, USA
- Konstantinos Mampentzidis (1)
Department of Computer Science, Aarhus
University, Aarhus, Denmark
- Alexey Markin  (12)
Department of Computer Science, Iowa State
University, Ames, IA, USA
- Milad Miladi  (14)
Bioinformatics Group, Department of Computer
Science, University of Freiburg, Germany
- Erin K. Molloy  (4)
University of Illinois at Urbana-Champaign,
USA
- Luay Nakhleh  (21, 22)
Department of Computer Science, Rice
University, 6100 Main Street, Houston, TX
77005, USA
- William Stafford Noble (10, 11)
Department of Genome Sciences, University of
Washington, Seattle, WA, USA; Paul G. Allen
School of Computer Science and Engineering,
University of Washington, Seattle, WA, USA
- Pierre Peterlongo  (8)
Univ. Rennes, Inria, CNRS, Irista, France
- Evgeny Polevikov (24)
Bioinformatics Institute, Saint Petersburg,
Russia
- Yutong Qiu (18)
Computational Biology Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, PA, USA
- Martin Raden  (14)
Bioinformatics Group, Department of Computer
Science, University of Freiburg, Germany
- Natalie Sauerwald (23)
Computational Biology Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, 15213, USA
- Mingfu Shao (15)
Department of Computer Science and
Engineering, The Pennsylvania State University,
University Park, PA, USA
- Yihang Shen (23)
Computational Biology Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, 15213, USA
- Pijus Simonaitis  (13)
LIRMM - Université Montpellier, France
- Ritambhara Singh (10)
Department of Genome Sciences, University of
Washington, Seattle, WA, USA
- Naveen Sivadasan (7)
TCS Research, Hyderabad, India
- Rajgopal Srinivasan (7)
TCS Research, Hyderabad, India
- Jens Stoye  (8)
Faculty of Technology and Center for
Biotechnology (CeBiTec), Bielefeld University,
Germany
- Krister M. Swenson  (13, 20)
LIRMM, CNRS - Université Montpellier, France
- Sandhya T. P. (1)
The Hong Kong Polytechnic University, Hung
Hom, Kowloon, Hong Kong
- Alexander Tiskin  (16)
Department of Computer Science, University of
Warwick, Coventry CV4 7AL, United Kingdom
- Todd J. Treangen  (19)
Department of Computer Science, Rice
University, Houston, TX 77005, USA
- Jakub Truszkowski  (20)
LIRMM, CNRS, Université Montpellier,
Montpellier, France
- Pranjal Vachaspati  (4)
University of Illinois at Urbana-Champaign,
USA
- Kavya Vaddadi (7)
TCS Research, Hyderabad, India
- Nelle Varoquaux  (11)
Department of Statistics, UC Berkeley, CA,
USA
- Jean-Philippe Vert  (10, 11)
Google Brain, Paris, France; Centre for
Computational Biology, MINES ParisTech, PSL
University, Paris, France
- Lusheng Wang (5)
Department of Computer Science, City
University of Hong Kong, China

Qi Wang  (19)

Systems, Synthetic, and Physical Biology
(SSPB) Graduate Program, Rice University,
Houston, TX 77005, USA

Tandy Warnow  (4)

University of Illinois at Urbana-Champaign,
USA

Sebastian Will  (14)

Theoretical Biochemistry Group (TBI), Institute
for Theoretical Chemistry, University of Vienna,
Austria

Roland Wittler  (2)

Genome Informatics, Faculty of Technology,
Bielefeld University, Germany; Center for
Biotechnology, Bielefeld University, Germany

Han Xie (18)

Computational Biology Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, PA, USA

Hongyi Xin (15)

Computer Science Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, PA, USA

Kohei Yamada (5)

Division of Information System Design, Tokyo
Denki University, Japan

Gürkan Yardımcı (11)

Department of Genome Sciences, University of
Washington, Seattle, WA, USA

Hamim Zafar  (22)

Computational Biology Department, School of
Computer Science, Carnegie Mellon University,
Pittsburgh, PA, USA

Meirav Zehavi (9)

Ben Gurion University of the Negev, Israel

Haowen Zhang (17)

School of Computational Science and
Engineering, Georgia Institute of Technology,
USA

Jiafan Zhu (21)

Department of Computer Science, Rice
University, 6100 Main Street, Houston, TX
77005, USA

Michal Ziv-Ukelson (9)

Ben Gurion University of the Negev, Israel

Roni Zoller (9)

Ben Gurion University of the Negev, Israel

Building a Small and Informative Phylogenetic Supertree

Jesper Jansson

The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
jesper.jansson@polyu.edu.hk

Konstantinos Mampentzidis

Department of Computer Science, Aarhus University, Aarhus, Denmark
kmampent@cs.au.dk

Sandhya T. P.

The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
tp.sandhya@gmail.com

Abstract

We combine two fundamental, previously studied optimization problems related to the construction of phylogenetic trees called *maximum rooted triplets consistency* (MAXRTC) and *minimally resolved supertree* (MINRS) into a new problem, which we call *q-maximum rooted triplets consistency* (*q*-MAXRTC). The input to our new problem is a set \mathcal{R} of resolved triplets (rooted, binary phylogenetic trees with three leaves each) and the objective is to find a phylogenetic tree with exactly q internal nodes that contains the largest possible number of triplets from \mathcal{R} . We first prove that *q*-MAXRTC is NP-hard even to approximate within a constant ratio for every fixed $q \geq 2$, and then develop various polynomial-time approximation algorithms for different values of q . Next, we show experimentally that representing a phylogenetic tree by one having much fewer nodes typically does not destroy too much triplet branching information. As an extreme example, we show that allowing only nine internal nodes is still sufficient to capture on average 80% of the rooted triplets from some recently published trees, each having between 760 and 3081 internal nodes. Finally, to demonstrate the algorithmic advantage of using trees with few internal nodes, we propose a new algorithm for computing the *rooted triplet distance* between two phylogenetic trees over a leaf label set of size n that runs in $O(qn)$ time, where q is the number of internal nodes in the smaller tree, and is therefore faster than the currently best algorithms for the problem (with $O(n \log n)$ time complexity [SODA 2013, ESA 2017]) whenever $q = o(\log n)$.

2012 ACM Subject Classification Mathematics of computing → Trees

Keywords and phrases phylogenetic tree, supertree, rooted triplet, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.1

Funding Konstantinos Mampentzidis was funded by the Danish National Research Foundation, grant DNRF84, via the Center for Massive Data Algorithmics (MADALGO).

1 Introduction

Background. Phylogenetic trees are used in biology to represent evolutionary relationships. The leaves in such a tree correspond to species that exist today and internal nodes to ancestor species that existed in the past. An important problem when studying the evolution of species is, given some data describing the species, to construct a phylogenetic tree that supports the input data as much as possible. The supertree approach [3] deals with the challenging problem of constructing a reliable phylogenetic tree for a large set of species by combining several accurate trees for small, overlapping subsets of the species into one final tree. Depending on the type of data that is available and the type of trees that we want to construct, we obtain several variants of the same problem.



© Jesper Jansson, Konstantinos Mampentzidis, Sandhya T. P.;
licensed under Creative Commons License CC-BY

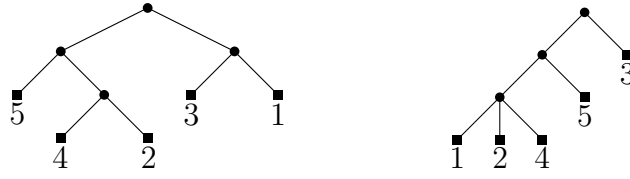
19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 1; pp. 1:1–1:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Let $L = \{1, 2, 3, 4, 5\}$ and $\mathcal{R} = \{45|3, 25|3, 13|5, 24|5, 23|1\}$. In this example no tree T such that $|\mathcal{R} \cap \text{rt}(T)| = 5$ exists. Left figure: optimal solution for MAXRTC with value 4. Right figure: optimal solution for 3-MAXRTC with value 3.

Problem Definition. A *rooted phylogenetic tree* is a rooted unordered tree in which every leaf has a distinct label and every internal node has at least two children. In this article, for simplicity we use the word “tree” to refer to a “rooted phylogenetic tree”. A *resolved triplet* is a binary tree with three leaves. The resolved triplet with leaf labels x, y , and z where z is closest to the root is denoted by $xy|z$. From now on when referring to a “triplet” we mean a “resolved triplet”. Let T be a tree on a leaf label set L of size n . For a node $u \in T$, $\text{deg}(u)$ is the number of children of u and $T(u)$ is the subtree induced by u and all the proper descendants of u . For two nodes u and v in T , $\text{lca}(u, v)$ is the lowest common ancestor node of u and v in T . We say that the triplet $xy|z$ is *induced by T* if $\text{lca}(x, z) = \text{lca}(y, z)$ and $\text{lca}(x, y) \neq \text{lca}(x, z)$. Let $\text{rt}(T)$ be the set of all triplets induced by T . Given a set \mathcal{R} of triplets, we say that \mathcal{R} is *consistent with T* , or equivalently T is *consistent with \mathcal{R}* , if $\mathcal{R} \subseteq \text{rt}(T)$.

Given a set \mathcal{R} of triplets on a leaf label set L of size n , in the *q -maximum rooted triplets consistency problem*, denoted q -MAXRTC, the goal is to find a tree T with exactly q internal nodes such that $|\mathcal{R} \cap \text{rt}(T)|$ is maximized, i.e., the total number of triplets induced by T that are also in \mathcal{R} is as large as possible. An example can be seen in Figure 1.

Let A be an algorithm for any maximization problem. Given an input instance I , let $\text{opt}(I)$ be the value of an optimal solution and $A(I)$ the value of the solution returned by A . Let $0 \leq r \leq 1$. We say that A is an r -approximation algorithm with *relative ratio* r , if $A(I) \geq r \cdot \text{opt}(I)$ for any I . Similarly, A is an r -approximation algorithm with *absolute ratio* r , if $A(I) \geq r \cdot |I|$ for any I . In particular, for q -MAXRTC we have that $A(I) \geq r \cdot |\mathcal{R}|$. From here on and unless otherwise stated, when we refer to any ratio r , we imply an absolute ratio.

Previous Work. Aho et al. [1] proposed a polynomial-time algorithm, called BUILD, that can determine if there exists a tree inducing all triplets from an input \mathcal{R} , and if such a tree exists, output it. As observed by Bryant [6], the BUILD algorithm does not always produce a tree with the minimal number of internal nodes. In fact, BUILD might return a tree with $\Omega(n)$ more internal nodes than needed [12], which is undesirable because unnecessary internal nodes may suggest false groupings of the leaves, also known as *spurious novel clades* [3]. Moreover, scientists typically look for the simplest possible explanation for some given observations and would prefer a tree that makes as few additional statements as possible about evolutionary relationships that are not supported by the input data. This motivates the *minimally resolved phylogenetic supertree* (MINRS) problem, where the output is a tree (if one exists) inducing all triplets from \mathcal{R} while having the minimum number of internal nodes. The decision version of MINRS is NP-complete for $q \geq 4$ and polynomial-time solvable for $q \leq 3$ [12], where q is the total number of internal nodes in the output tree. An exact exponential-time algorithm for MINRS and experimental results for the non-optimality of BUILD for MINRS were given in [14]. For the special case of *caterpillar trees* (trees in which every internal node has at most one non-leaf child), MINRS is polynomial-time solvable for any q [12].

The above problems only consider finding trees that induce all triplets from \mathcal{R} . However, in situations where such a tree cannot be constructed, e.g., due to a single error in the input triplets, it is still useful to build a tree that induces as many of the triplets

from \mathcal{R} as possible. This has been formalized as the *maximum rooted triplets consistency problem* (MAXRTC). Bryant [6] showed that MAXRTC is NP-hard and Gąsieniec et al. [10] proposed a polynomial-time top-down $\frac{1}{3}$ -approximation algorithm that always returns a caterpillar tree. Byrka et al. [8] showed that a bottom-up algorithm by Wu [21] can be modified to also obtain a polynomial-time $\frac{1}{3}$ -approximation algorithm. In [7], Byrka et al. gave a $\frac{1}{3}$ -approximation algorithm by derandomizing a randomized algorithm. In Section 3 below, we refer to the algorithm in [10] as *One-Leaf-Split* (OLS) and the algorithm in [8] as *Wu's algorithm* (WU). For more results related to the computational complexity of MAXRTC, see [8].

Motivation. The existing approximation algorithms for MAXRTC typically produce trees with an arbitrary number of internal nodes. For example, the algorithms in [7, 8] always produce a tree with $n - 1$ internal nodes and the algorithm in [10], $n - 1$ for certain \mathcal{R} . However, due to the issue of spurious novel clades [3] mentioned above, biologists may prefer to build a supertree with few internal nodes that is still consistent with a large number of input triplets, which leads to the new problem q -MAXRTC introduced in this paper. More precisely, q -MAXRTC can be regarded as a combination of MINRS and MAXRTC that models how well the triplet branching information contained in the set of input triplets can be preserved while forcing the size of the output tree to be bounded by a user-specified parameter q . On a high level, q -MAXRTC is related to the problem of compressing a large data file into a small data file; as an analogy, consider the widely used JPEG compression method for images. Both JPEG and q -MAXRTC are examples of *lossy compression* where the user controls a parameter yielding a trade-off between the size of the compressed data (the number of bits for JPEG and the number of internal nodes for q -MAXRTC) and the amount of preserved information (the image quality for JPEG and the number of induced triplets in \mathcal{R} for q -MAXRTC). Finally, in the design of phylogenetic tree comparison algorithms, trees with fewer internal nodes sometimes admit faster running times. For example, given two trees built on the same leaf label set of size n , the fastest known algorithms for computing the so-called *rooted triplet distance* between the two trees takes $O(n \log n)$ time [4, 5], but if at least one of the input trees has $O(1)$ internal nodes then the time complexity can be reduced to $O(n)$; see Section 5. As the available published trees get larger and larger (the total number of species on Earth was recently estimated to be 1 trillion [17]), to make their comparison practical, it may become necessary to approximate them using trees with fewer internal nodes while keeping enough triplet branching structure to represent them accurately.

New Results and Outline of the Article. Section 2 shows that q -MAXRTC is NP-hard for every fixed $q \geq 2$ and gives some inapproximability results. Section 3 describes our new approximation algorithms. Section 4 provides implementations and our experimental results. Section 5 presents a new algorithm for computing the rooted triplet distance between two trees. Finally, Section 6 contains some open problems. For a summary of previous and new results related to q -MAXRTC refer to the table below. Due to space constraints, some proofs and experimental results have been deferred to the journal version.

Year	Reference	Deterministic	q	Approximation	Type
1999	Gąsieniec et al. [10]	yes	unbounded	$1/3$	absolute
2010	Byrka et al. [7, 8]	yes	$n - 1$	$1/3$	absolute
2019	new [Section 3.1]	no	2	$1/2$	relative
2019	new [Section 3.1]	yes	2	$1/4$	relative
2019	new [Theorem 7]	yes	2	$4/27$	absolute
2019	new [Theorem 9]	yes	$q \geq 3$	$1/3 - 4/(3(q + q \bmod 2)^2)$	absolute

2 Computational Complexity of q -MAXRTC

In this section, we study the computational complexity of q -MAXRTC. We first address the NP-hardness of q -MAXRTC, and then present some inapproximability results.

► **Theorem 1.** q -MAXRTC is NP-hard for every fixed $q \geq 2$.

Proof. We consider the known NP-hard problem MAX q -CUT [15], in which the input is an undirected graph $G = (V, E)$ and the goal is to find a partition (A_1, A_2, \dots, A_q) of V such that the total number of edges connecting two nodes residing in different sets, i.e., *the size of the cut*, is maximized. We prove that q -MAXRTC is NP-hard by reducing MAX q -CUT to q -MAXRTC as follows: let $L = V \cup \{z\}$ and $\mathcal{R} = \{xz|y, yz|x : \{x, y\} \in E\}$. We claim that there exists a cut (A_1, A_2, \dots, A_q) of size k in G if and only if there exists a solution to q -MAXRTC that is consistent with k triplets from \mathcal{R} . We now prove the claim.

First, assume that there exists a cut (A_1, A_2, \dots, A_q) of size k in G . We construct a tree T that is rooted at the vertex a_1 with additional internal nodes a_2, \dots, a_q such that a_{i+1} is a child of a_i for $1 \leq i \leq q-1$. For $i \in \{1, 2, \dots, q\}$, we attach $|A_i|$ leaves bijectively labeled by A_i as children of a_i , and the vertex z is added as a child of a_q . Consider any edge $\{x, y\}$ in the cut. By the definition of a cut, $x \in A_i$ and $y \in A_j$ for two different $i, j \in \{1, 2, \dots, q\}$. If $i < j$, then $yz|x$ will be consistent with T , since $\text{lca}(y, z) = a_j$ is a proper descendant of $\text{lca}(x, y) = \text{lca}(x, z) = a_i$. Similarly, if $i > j$, then $xz|y$ will be consistent with T . For every edge in the cut, exactly one triplet will be consistent with T , so T will be consistent with exactly k triplets from \mathcal{R} .

Conversely, assume that there exists a tree T with q internal nodes a_1, a_2, \dots, a_q that is consistent with k triplets from \mathcal{R} . Let $A_i = \{x : x \text{ is a child of } a_i\} \setminus \{z, a_1, a_2, \dots, a_q\}$ for $1 \leq i \leq q$. Define $S = \mathcal{R} \cap \text{rt}(T)$. For each $xz|y \in S$, clearly x and y belong to different sets A_i and A_j for some $i, j \in \{1, 2, \dots, q\}$, and thus the corresponding edge $\{x, y\}$ contributes one to the size of the cut, making the size of the cut $|S| = k$. ◀

From the inapproximability of MAXCUT [11], we obtain the following corollary:

► **Corollary 2.** Unless $P=NP$, 2-MAXRTC cannot be approximated in polynomial time within a relative ratio of $16/17 + \epsilon$, for any constant $\epsilon > 0$.

From the inapproximability of MAX q -CUT [15], we obtain the following corollary:

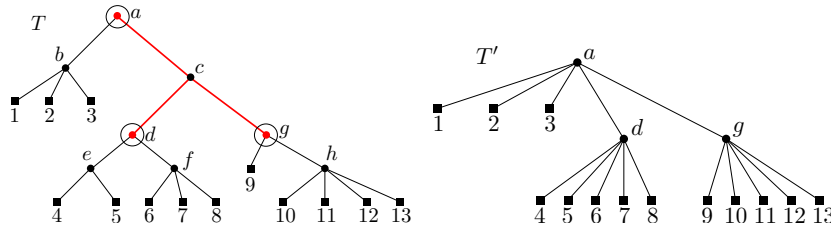
► **Corollary 3.** Unless $P=NP$, for any $q \geq 3$, it holds that q -MAXRTC cannot be approximated in polynomial time within a relative ratio of $1 - 1/(34q) + \epsilon$, for any constant $\epsilon > 0$.

3 Approximability of q -MAXRTC

Intuitively, a tree with a larger number of internal nodes should be able to induce more triplets from a given \mathcal{R} . The next lemma shows that this is indeed so, and upper bounds the total number of triplets that can be induced. Define $\text{opt}(q)$ to be the maximum number of triplets that can be consistent with a tree T with q internal nodes.

► **Lemma 4.** Let $2 \leq q' \leq q \leq n-1$. We have that $\text{opt}(q') \leq \text{opt}(q) \leq \lceil \frac{q-1}{q'-1} \rceil \text{opt}(q')$.

Proof. We start by showing that $\text{opt}(q') \leq \text{opt}(q)$. Let T' be the tree with q' internal nodes that induces $\text{opt}(q')$ triplets from \mathcal{R} . We can create a tree T with q internal nodes that induces at least as many triplets from \mathcal{R} as follows. Let $T = T'$. While T does not have q internal nodes, let $u \in T$ such that $\text{deg}(u) > 2$ and u_1, u_2 be two children of u . Create an internal node u_{12} , make u_1 and u_2 the children of u_{12} and u_{12} the child of u .



■ **Figure 2** An example. Let T be the tree on the left with 9 internal nodes. The tree T' on the right with 3 internal nodes is created by deleting all internal nodes in T except $W = \{a, d, g\}$.

To show the second half of the inequality, proceed as follows. Define the *delete operation* on any non-root node u in a tree as the operation of making the children of u become children of the parent of u , and then removing u and all edges incident to u . Let T be the tree that induces $opt(q)$ triplets from \mathcal{R} . Let $t = ab|c$ be a triplet induced in T that is also in \mathcal{R} . Anchor t in $lca(a, b)$. Let $W = \{u_1, u_2, \dots, u_{q'}\}$ be any set of q' internal nodes in T such that the root of T is included in W . Create a tree T' with q' internal nodes by letting T' be a copy of T and applying the delete operation to every internal node of T' not in W . Note that for a node u in T such that $u \in W$, every triplet anchored in u will also be induced by T' . An example can be found in Figure 2.

Let $T'_1, T'_2, \dots, T'_\lambda$ be trees that are built like T' , but in a way such that every internal node $u \in T$ except $r(T)$, corresponds to an internal node of exactly one such tree. Observe that $\lambda = \lceil \frac{q-1}{q'-1} \rceil$. We can create these trees with the following procedure:

- Store all internal nodes of T except $r(T)$ in the ordered set S , in any order from left to right and set $j = 1$.
- If $|S| \geq q' - 1$, pick and remove from S the first $q' - 1$ internal nodes to define W , and construct T'_j . Otherwise, pick the remaining nodes in S to define W and create T'_j just like T' but with $|S| = |W|$ nodes instead of $q' - 1$. Set $j = j + 1$.
- if $|S| = 0$ stop. Otherwise go to the previous step.

We then have: $|rt(T) \cap \mathcal{R}| = opt(q) \leq \sum_{i=1}^{\lambda} |rt(T'_i) \cap \mathcal{R}| \leq \lambda opt(q') = \lceil \frac{q-1}{q'-1} \rceil opt(q')$. ◀

3.1 Approximation Algorithms Based on MAX 3-CSP

In this subsection, we consider polynomial-time approximation algorithms. MAX 3-AND is a Boolean satisfiability problem in which we are given as input a logical formula consisting of a set of clauses, each being a conjunction (AND) of three literals formed from a set of Boolean variables, and the goal is to assign each Boolean variable a *True/False*-value so that the total number of satisfied clauses is maximized. Both MAX 3-AND and the well-known MAX 3-SAT problem are special cases of the MAX 3-CSP problem [22], where a clause can be an arbitrary function over three literals. The following lemma shows that 2-MAXRTC can be reduced to MAX 3-AND in polynomial time while preserving the approximation ratio.

► **Lemma 5.** *If MAX 3-AND can be approximated within a factor of r , then 2-MAXRTC can also be approximated within a factor of r .*

Lemma 5 allows every approximation algorithm for MAX 3-AND to be used to approximate 2-MAXRTC. For MAX 3-AND, Zwick [22] presented a randomized $\frac{1}{2}$ -approximation algorithm with relative ratio based on semi definite programming. Trevisan [19] presented a deterministic $\frac{1}{4}$ -approximation algorithm with relative ratio based on linear programming. A deterministic algorithm based on local search by Alimonti [2], would satisfy $\geq \frac{1}{8}|C|$ number of clauses, giving a $\frac{1}{8}$ -approximation ratio for 2-MAXRTC. Since this ratio is absolute, from Lemma 4 this algorithm also gives a $\frac{1}{8}$ -approximation ratio for q -MAXRTC, where $q \geq 3$.

3.2 Approximation Algorithms Based on Derandomization

This subsection also assumes that all approximation algorithms run in polynomial time. Reducing 2-MAXRTC to MAX 3-AND can produce a deterministic $\frac{1}{8}$ -approximation algorithm for q -MAXRTC, however from Lemma 4, we should be able to capture more triplets by allowing more internal nodes. The algorithms based on MAX 3-AND cannot be directly extended to support Lemma 4. We propose a new deterministic algorithm for q -MAXRTC that achieves a $\frac{4}{27}$ -approximation ratio, based on a randomized algorithm for 2-MAXRTC, and then show how to extend it to get better approximation ratios for higher values of q . Note that the only available related algorithm based on derandomization by Byrka et al. [7], always constructs a binary tree on n leaves, i.e. the case $q < n - 1$ is not considered. Moreover, as we will show below, our derandomization procedure is highly optimized for trees instead of the more complex *phylogenetic networks* (for a definition see Section 2 of [7]).

► **Lemma 6.** *There exists a randomized $\frac{4}{27}$ -approximation algorithm for q -MAXRTC.*

Proof. Let $\mathcal{R} = \{r_1, \dots, r_{|\mathcal{R}|}\}$ be the set of triplets and $L = \{x_1, \dots, x_n\}$ the leaf label set. Build a tree T with two internal nodes, with a being the root and b the child of the root. Make every leaf $x_i \in L$ with probability $\frac{2}{3}$ a child of b and probability $\frac{1}{3}$ a child of a . Let Y_j be a random variable that is 1 if $r_j \in rt(T)$ and 0 otherwise. Let $W = \sum_{j=1}^{|\mathcal{R}|} Y_j$. For the expected number of triplets consistent with T we have $E[W] = \sum_{j=1}^{|\mathcal{R}|} E[Y_j] = \sum_{j=1}^{|\mathcal{R}|} \frac{4}{27} = \frac{4}{27}|\mathcal{R}|$. ◀

► **Theorem 7.** *There exists a deterministic $\frac{4}{27}$ -approximation algorithm for q -MAXRTC that runs in $O(|\mathcal{R}|)$ time.*

Proof. We derandomize the algorithm in Lemma 6 with the method of conditional expectations [20] in a way that differs from Byrka et al. [7], where the main focus is the general case of phylogenetic networks. In our method, the leaves x_1, \dots, x_n are scanned from left to right, and each leaf is deterministically assigned to either be the child of b , denoted $x_i \leftarrow b$, or the child of a , denoted $x_i \leftarrow a$. The leaves are assigned in a way, such that after every assignment the expected value of the solution is preserved. From probability theory we have $E[W] = \frac{1}{3}E[W|x_1 \leftarrow a] + \frac{2}{3}E[W|x_1 \leftarrow b]$. We choose $n_1 = a$ or $n_1 = b$ such that $E[W|x_1 \leftarrow n_1] = \max(E[W|x_1 \leftarrow a], E[W|x_1 \leftarrow b])$. Then $E[W|x_1 \leftarrow n_1] \geq E[W] = \frac{4}{27}|\mathcal{R}|$. Suppose that the first i leaves have been assigned to n_1, \dots, n_i . Let N_i contain those assignments, i.e., $N_i = \{x_1 \leftarrow n_1, \dots, x_i \leftarrow n_i\}$. To find the assignment for x_{i+1} we follow the same approach as that for x_1 , i.e., we have $E[W|N_i] = \frac{1}{3}E[W|N_i, x_{i+1} \leftarrow a] + \frac{2}{3}E[W|N_i, x_{i+1} \leftarrow b]$ and then n_{i+1} is chosen so that $E[W|N_{i+1}] = \max(E[W|N_i, x_{i+1} \leftarrow a], E[W|N_i, x_{i+1} \leftarrow b])$. By induction, we then get that $E[W|N_{i+1}] \geq E[W|N_i] \geq \dots \geq \frac{4}{27}|\mathcal{R}|$.

To compute $E[W|N_i]$, we use the fact that $E[W|N_i] = \sum_{j=1}^{|\mathcal{R}|} Pr[r_j \in rt(T)|N_i]$, where $Pr[r_j \in rt(T)|N_i]$ can be computed in $O(1)$ time (see procedure PR2() of Algorithm 1). A trivial implementation that scans the leaves and for every possible assignment of a leaf x_i , computes the expected value $E[W|N_i]$ by scanning the entire set \mathcal{R} would require $O(n|\mathcal{R}|)$ time.

We can achieve a more efficient implementation (see procedure 2-MAXRTC-FAST() of Algorithm 1) that would require $O(|\mathcal{R}|)$ time, by maintaining for every leaf $x_i \in L$, a list of all the triplets that x_i is part of, denoted $\mathcal{R}[x_i]$. At the beginning of the i -th iteration of the first for loop in Algorithm 1, the value of the variable *prev* is $E[W|N_{i-1}]$. To determine the assignment for leaf x_i , we need to compute $E[W|N_{i-1}, x_i \leftarrow a]$ and $E[W|N_{i-1}, x_i \leftarrow b]$, and for this we use the second for loop. At the end of the execution of the second for loop, the value of $E[W|N_{i-1}, x_i \leftarrow a]$ will be stored in the variable *aValue* and the value of $E[W|N_{i-1}, x_i \leftarrow b]$ in the variable *bValue*. To compute *aValue* (resp. *bValue*), we initialize it

■ **Algorithm 1** $O(|\mathcal{R}|) \frac{4}{27}$ -approximation algorithm for q -MAXRTC based on 2-MAXRTC.

```

1: procedure PR2( $xy|z$ )                                ▷ Computing  $Pr[xy|z \in rt(T)|N_i]$ 
2:   if  $x \leftarrow a$  or  $y \leftarrow a$  or  $z \leftarrow b$  then return 0
3:    $p = 4/27$ 
4:   if  $x \neq \emptyset$  and  $x \leftarrow b$  then  $p = 3p/2$     ▷  $x \neq \emptyset$  meaning that  $x$  has been assigned
5:   if  $y \neq \emptyset$  and  $y \leftarrow b$  then  $p = 3p/2$ 
6:   if  $z \neq \emptyset$  and  $z \leftarrow a$  then  $p = 3p$ 
7:   return  $p$ 

8: procedure 2-MAXRTC-FAST( $\mathcal{R}$ )                          ▷ The main procedure
9:    $prev = 4|\mathcal{R}|/27$                                     ▷ Storing  $E[W|N_0]$ , where  $N_0 = \emptyset$ 
10:  for  $i = 1$  to  $n$  do
11:     $aValue = prev$                                        ▷ To compute  $E[W|N_{i-1}, x_i \leftarrow a]$ 
12:     $bValue = prev$                                        ▷ To compute  $E[W|N_{i-1}, x_i \leftarrow b]$ 
13:    for  $j = 1$  to  $|\mathcal{R}[x_i]|$  do
14:       $x_i \leftarrow \emptyset$ 
15:       $aValue = aValue - PR2(\mathcal{R}[x_i][j])$ 
16:       $bValue = bValue - PR2(\mathcal{R}[x_i][j])$ 
17:       $x_i \leftarrow a$ 
18:       $aValue = aValue + PR2(\mathcal{R}[x_i][j])$ 
19:       $x_i \leftarrow b$ 
20:       $bValue = bValue + PR2(\mathcal{R}[x_i][j])$ 
21:     $x_i \leftarrow b$ 
22:     $prev = bValue$ 
23:    if  $aValue > bValue$  then
24:       $x_i \leftarrow a$ 
25:       $prev = aValue$ 

```

to the value of $prev$, and then for every triplet in the list $\mathcal{R}[x_i]$, we subtract the contribution of that triplet to the value of $prev$ when $x_i \leftarrow \emptyset$, and add its new contribution by having $x_i \leftarrow a$ (resp. $x_i \leftarrow b$). Since every triplet in \mathcal{R} will be part of 3 lists, every triplet will induce $O(1)$ calls to the procedure PR2() of Algorithm 1, giving the $O(|\mathcal{R}|)$ final bound of the algorithm. ◀

In the following theorem, we prove that the best possible absolute approximation ratio for 2-MAXRTC is $\frac{4}{27}$, making the approximation algorithm in Theorem 7 optimal when considering algorithms with absolute approximation ratios.

► **Theorem 8.** *For any $\epsilon > 0$, there exists some n and set \mathcal{R} of triplets on a leaf label set of size n , such that the approximation ratio $\geq \frac{4}{27} + \epsilon$ for 2-MAXRTC is impossible.*

Proof. For any n , let $L_n = \{1, 2, \dots, n\}$ and $\mathcal{R}_n = \{ab|c, ac|b, bc|a : a, b, c \in L, |\{a, b, c\}| = 3\}$. Since $|L_n| = n$, we have $|\mathcal{R}_n| = 3\binom{n}{3}$. Next, we construct a tree T with two internal nodes, which is rooted at the vertex a with an internal node b (b is a child of a). Let $A = \{x : x \text{ is a child of } a\} \setminus \{b\}$ and $B = \{x : x \text{ is a child of } b\}$. Assume that $m = |A|$. Then $|B| = n - m$ and $|rt(T) \cap \mathcal{R}_n| = m\binom{m-1}{2}(n-m)$. By taking derivatives, we obtain that T is consistent with the largest number of triplets when $m = \frac{n+1+\sqrt{n^2-n+1}}{3}$. For that given m , we then have $|rt(T) \cap \mathcal{R}_n| = \left(\frac{n+1+\sqrt{n^2-n+1}}{3}\right)\left(\frac{n-2+\sqrt{n^2-n+1}}{6}\right)\left(\frac{2n-1-\sqrt{n^2-n+1}}{3}\right)$ and $\lim_{n \rightarrow \infty} \frac{|rt(T) \cap \mathcal{R}_n|}{|\mathcal{R}_n|} = \frac{4}{27}$. ◀

To obtain an algorithm that has a better approximation ratio for $q \geq 3$, we allow the output tree T to have q internal nodes $\{u_1, \dots, u_q\}$. Every internal node $u_j \in T$ has a probability $p(u_j)$, which is the probability of a fixed leaf being assigned to that node. Given that $\sum_{j=1}^q p(u_j) = 1$, we can obtain a randomized algorithm, the analysis of which follows from Lemma 6. Let $E[W]$ be the expected value of that randomized algorithm. Like in Theorem 7, we can derandomize the algorithm to obtain a $\frac{E[W]}{|\mathcal{R}|}$ -approximation ratio. The only difference in the proof is that the total number of possible assignments is q instead of 2, i.e., given N_i , we choose n_{i+1} for x_{i+1} such that $E[W|N_i, x_{i+1} \leftarrow n_{i+1}] = \max(E[W|N_i, x_{i+1} \leftarrow u_1, \dots, E[W|N_i, x_{i+1} \leftarrow u_q])$. The problem is thus reduced to finding a tree with q internal nodes and a choice of probabilities $p(u_1), \dots, p(u_q)$ such that $E[W] > \frac{4}{27}|\mathcal{R}|$.

► **Theorem 9.** *Given $q \geq 3$, there exists a randomized algorithm for q -MAXRTC that achieves a $(\frac{1}{3} - \frac{4}{3(q+q \bmod 2)^2})$ -approximation. The algorithm can be derandomized while preserving the approximation ratio. The running time of the deterministic algorithm is $O(q|\mathcal{R}|)$.*

4 Implementation and Experiments

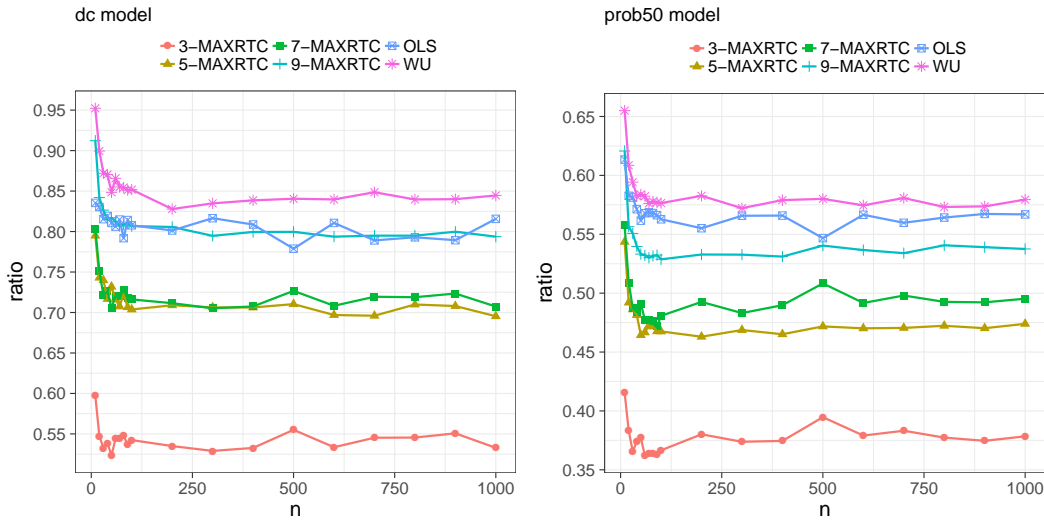
We used the C++ programming language to implement the algorithm from Theorem 7 for 2-MAXRTC, and the algorithm from Theorem 9 for q -MAXRTC when $q > 2$. The implementation is publicly available at <https://github.com/kmampent/qMAXRTC>. Below, we describe some experiments on both simulated and real datasets and the results.

Simulated Dataset. The input to q -MAXRTC is a set of triplets \mathcal{R} and a parameter q . We define the following types of sets for \mathcal{R} :

- *dense consistent* (abbreviated **dc**): if $|\mathcal{R}| = \binom{n}{3}$ and \mathcal{R} is consistent with a tree T containing $n - 1$ internal nodes. The tree T is created using the uniform model [18].
- *probabilistic*: if $|\mathcal{R}| = n^2$ and \mathcal{R} is a set of triplets on n leaf labels created as follows. After building a binary tree T on n leaves following the uniform model, start extracting triplets from T to add into \mathcal{R} . For every extracted triplet $xy|z$, permute the leaves uniformly at random with probability p . Depending on whether $p = 0.25$, $p = 0.50$ or $p = 0.75$ the abbreviations we use are **prob25**, **prob50**, and **prob75** respectively.

In the experiments of this dataset, the *performance* of an algorithm for any fixed q , n , and dataset model is defined as its mean approximation ratio, taken over 100 randomly generated instances of size n . Figure 3 compares the performance of q -MAXRTC, **WU**, and **OLS** in the **dc** and **prob50** models, for small values of q and n at most 1000. In both models, the larger the value of q , the better the performance of q -MAXRTC. Moreover, the improvement in performance decreases as the value of q increases, which is expected. For the **dc** dataset, which contains no conflicting triplets, the performance is much better. Significantly, when $q = 9$ we can capture close to 80% of the triplets even if the input tree contains as many as 1000 leaves. When compared against **WU** & **OLS**, we can see that while **WU** & **OLS** perform better, the difference in performance is small compared to the difference in the number of internal nodes used by the algorithms.

Real Dataset. We considered five trees from recently published papers ([9] and [16]). From [9] we used the trees from the supplementary datasets 2 and 4, denoted **nms2** and **nms4** respectively. From [16] we used the trees from the supplementary datasets 1, 2, and 4, denoted **pos1**, **pos2**, and **pos4** respectively. All trees are binary except **nms2** and **nms4**. However, we removed the leaf that is a child of **nms2**'s root to make **nms2** binary. Similarly,



■ **Figure 3** Performance of $\{3, 5, 7, 9\}$ -MAXRTC compared to WU and OLS in the `dc` and `prob50` models. Every data point corresponds to the mean of 100 runs. Observe that the performance of 9-MAXRTC is very close to that of WU & OLS, even though 9-MAXRTC uses only 9 internal nodes, while WU uses exactly $n - 1$ internal nodes and OLS at most $n - 1$.

■ **Table 1** Performance of q -MAXRTC on real datasets. Every cell corresponds to the best ratio (as defined below) over 100 runs. The size of each leaf label set is written inside the parenthesis.

q	poS1(761)	poS2(761)	poS4(841)	nmS4(1869)	nmS2(3082)	Average
2	0.27	0.36	0.43	0.41	0.29	0.35
3	0.67	0.54	0.48	0.41	0.46	0.51
5	0.77	0.81	0.67	0.66	0.72	0.73
7	0.82	0.75	0.76	0.62	0.73	0.74
9	0.86	0.71	0.87	0.80	0.79	0.81
11	0.91	0.89	0.87	0.79	0.87	0.87

we removed the two leaves that are children of `nmS4`'s root to make `nmS4` binary as well. The total number of leaves in `nmS2`, `nmS4`, `poS1`, `poS2`, and `poS4` is 1869, 3082, 761, 761, and 841. Since the trees are binary, the total number of internal nodes is 1868, 3081, 760, 760, and 840.

For a tree $T \in \{\text{nmS2}, \text{nmS4}, \text{poS1}, \text{poS2}, \text{poS4}\}$ with n leaf labels, let T_q be the tree produced by the new algorithm. Let $D(T, T_q)$ be the rooted triplet distance between T and T_q (for a definition see Section 5 below). The performance of q -MAXRTC in the experiments of this dataset is then defined by the ratio $S(T, T_q) / \binom{n}{3}$, where $S(T, T_q) = \binom{n}{3} - D(T, T_q)$. To compute this ratio efficiently, we used the rooted triplet distance implementation in [5]. We measured the performance of q -MAXRTC for $q \in \{2, 3, 5, 7, 9, 11\}$. Every experiment consisted of 100 runs, and in each run n^2 triplets were picked at random from the corresponding tree to define the set \mathcal{R} . We made sure that each leaf from a given tree appeared in \mathcal{R} so that the size of the leaf label set was as big as the leaf label set of the tree.

Table 1 shows the best ratios achieved, and the corresponding trees in Newick format can be found at <https://github.com/kmampent/qMAXRTC>. As can be seen from the results, larger number of internal nodes tend to improve performance. Significantly, with only 9 nodes we can capture between 71% and 86% of the triplets in each case, and with 11 nodes between 79% and 91%. When $q > 11$, we did not observe a significant improvement in performance.

5 Motivation for q -MAXRTC: Faster Computation of the Rooted Triplet Distance

Finally, we give an example of the algorithmic advantage of using phylogenetic trees with few internal nodes. More precisely, we develop an algorithm for computing the rooted triplet distance between two phylogenetic trees in $O(qn)$ time, where q is the number of internal nodes in the smaller tree and n is the number of leaf labels.

Problem Definition. The rooted triplet distance between two trees T_1 and T_2 built on the same leaf label set, is the total number of trees with three leaves that appear as embedded subtrees in T_1 but not in T_2 . Intuitively, two trees with very similar branching structure will share many embedded subtrees, so the rooted triplet distance between them will be small.

Formally, let T_1 and T_2 be two trees built on the same leaf label set of size n . We need to distinguish between two types of triplets. The first type is the *resolved triplet*, previously defined in Section 1. In addition, since T_1 and T_2 can be non-binary, we also need to define the *fan triplet*. We call $t = x|y|z$ a *fan triplet*, if t is a tree with the three leaves x , y , and z , and one internal node that is the root of t . The definition of when a resolved triplet is consistent with a tree T follows from Section 1. Similarly to a resolved triplet, we say that the fan triplet $x|y|z$ is consistent with a tree T , where x , y , and z are leaves in T , if $\text{lca}(x, y) = \text{lca}(x, z) = \text{lca}(y, z)$. In this section only, we use the word *triplet* to refer to both fan and resolved triplets. Moreover, when we refer to a fan triplet $x|y|z$ or a resolved triplet $xy|z$ induced by a tree T , there exists a left to right ordering of x , y , and z in T .

Let $D(T_1, T_2)$ be the rooted triplet distance between T_1 and T_2 . Define $S(T_1, T_2)$ to be the total number of triplets that are consistent with both T_1 and T_2 , commonly referred to as *shared triplets*. For the rooted triplet distance we then have $D(T_1, T_2) = \binom{n}{3} - S(T_1, T_2)$.

The Algorithm. It is known how to compute $D(T_1, T_2)$ in $O(n \log n)$ time [4, 5]. Below, we show how to compute $D(T_1, T_2)$ in $O(qn)$ time, which is faster than [4, 5] when $q = o(\log n)$. There is a preprocessing step and a counting step.

Preprocessing. The leaves in T_2 are relabeled according to their discovery time by a depth first traversal of T_2 , in which the children of a node are discovered from left to right. Notice that for a node v in T_2 , the labels of the leaves in $T_2(v)$ will correspond to a continuous range of numbers. Afterwards, we transfer the new labels of the leaves in T_2 to the leaves in T_1 . For T_1 , we define the $q \times n$ table A such that for a node u in T_1 we have $A[u][\ell] = 1$ if ℓ is a leaf in $T_1(u)$, and $A[u][\ell] = 0$ otherwise. We construct another table C to answer one dimensional range queries as follows. For $1 \leq i \leq n$ we have $C[u][i] = \sum_{j=1}^i A[u][j]$ and $C[u][0] = 0$. The C table will be used to answer queries asking for the total number of leaves in $T_2(v)$ that are also in $T_1(u)$ in $O(1)$ as follows. Let $[l, \dots, r]$ be the continuous range of leaf labels in $T_2(v)$. The answer to the query will be exactly $C[u][r] - C[u][l - 1]$.

Counting. We extend the technique introduced in [5]. Let $t = xy|z$ or $t = x|y|z$ be a triplet induced by a tree T , which in our problem can be either T_1 or T_2 . We anchor t in the edge $\{v, c\}$, where $v = \text{lca}(x, y)$ and c is the child of v such that $T(v)$ contains y . The following lemma shows that every triplet induced by T is anchored in exactly one edge of T .

► **Lemma 10.** *Let T be a tree in which every triplet t with the three leaves x , y , and z is anchored in the edge $\{u, c\}$, such that $u = \text{lca}(x, y)$ and $T(c)$ contains y . Every triplet induced by T is anchored in exactly one edge of T .*

Suppose that a node v in T_2 has the children $v_1, \dots, v_j, \dots, v_i$ where $1 < j \leq i$. To capture all triplets anchored in edge $\{v, v_j\}$ of T_2 , we color the leaves of T_2 as follows. Let

every leaf in $T_2(v_1), \dots, T_2(v_{j-1})$ have the color red, every leaf in $T_2(v_j)$ have the color blue, every leaf in $T_2(v_{j+1}), \dots, T_2(v_i)$ have the color green and every other leaf in T_2 have the color white. The red, blue, and green colors will be used to capture fan triplets and the red, blue, and white colors, resolved triplets. By the relabeling scheme of the leaves, we have that the red, blue, and green colors correspond to exactly one continuous range of leaf labels each. Let those ranges be $R = [a_{\text{red}}, \dots, a'_{\text{red}}]$, $B = [a_{\text{blue}}, \dots, a'_{\text{blue}}]$, and $G = [a_{\text{green}}, \dots, a'_{\text{green}}]$, for the colors red, blue, and green respectively. Note that $a_{\text{blue}} = a'_{\text{red}} + 1$ and if G is non-empty, $a_{\text{green}} = a'_{\text{blue}} + 1$. Finally, note that a leaf has the color white if and only if it does not have any other color.

We are now going to describe how to compute the total number of triplets anchored in some edge $\{v, v'\}$ in T_2 , where v is the parent of v' , that are also consistent with T_1 , denoted $S^{\{v, v'\}}(T_1, T_2)$. Let $S_r^{\{v, v'\}}(T_1, T_2)$ denote the shared resolved triplets anchored in $\{v, v'\}$ and similarly let $S_f^{\{v, v'\}}(T_1, T_2)$ denote the shared fan triplets. Note that we have $S^{\{v, v'\}}(T_1, T_2) = S_r^{\{v, v'\}}(T_1, T_2) + S_f^{\{v, v'\}}(T_1, T_2)$. The following lemma gives an algorithm for computing $S^{\{v, v'\}}(T_1, T_2)$ efficiently.

► **Lemma 11.** *Given the ranges R , B , and G that define a coloring of the leaves in T_2 according to an edge $\{v, v'\}$ of T_2 , there exists a $O(q)$ -time algorithm for computing $S^{\{v, v'\}}(T_1, T_2)$.*

Proof. Since both T_1 and T_2 are built on the same leaf label set, a coloring of the leaves of T_2 defines a coloring of the leaves of T_1 . Suppose that a node u in T_1 has the m children u_1, \dots, u_m , where $m \geq 2$. Some children could be leaves and others, internal nodes. Let I denote the set containing the children that are internal nodes and L the children that are leaves. Let $T(I) = \{T(u) : u \in I\}$. Define the following counters:

1. u_{white} : total number of leaves with the white color in T_1 but not in $T_1(u)$.
2. u_i , for $i \in \{\text{red}, \text{blue}, \text{green}\}$: total number of leaves with color i in $T_1(u)$.
3. u_{iI} , for $i \in \{\text{red}, \text{blue}, \text{green}\}$: total number of leaves with color i in $T(I)$.
4. u_{iL} , for $i \in \{\text{red}, \text{blue}, \text{green}\}$: total number of leaves with color i in L .
5. $u_{i,j}$, for $(i, j) \in \{(\text{red}, \text{blue}), (\text{red}, \text{green}), (\text{blue}, \text{green})\}$: total number of pairs of leaves in $T(I)$, such that one has color i , the other has color j , and both come from different subtrees attached to u .
6. $u_{\text{red}, \text{blue}, \text{green}}$: total number of leaf triples in $T(I)$, such that one leaf has the color red, another the color blue, another the color green, and they all come from different subtrees attached to u .

To compute these counters for every internal node of T_1 efficiently, a depth first traversal is applied on T_1 while making sure that we only visit internal nodes. For every such internal node u visited, a simple dynamic programming procedure is used to compute the counters of u in $O(|I|)$ time, thus making the total time required to compute all counters $O(q)$.

Algorithm 2 shows how to compute $S_f^{\{v, v'\}}(T_1, T_2)$ and $S_r^{\{v, v'\}}(T_1, T_2)$ in $O(q)$ time as well. It counts shared triplets by considering for every internal node u in T_1 , all possible cases for the location of the leaves of a shared triplet anchored in any edge $\{u, u'\}$ in T_1 , where u is the parent of u' . More precisely, for the leaves of a fan triplet anchored in any edge $\{u, u'\}$ in T_1 , we have the following cases: (1) all three leaves come from $T(I)$, (2) two leaves come from $T(I)$ and one from L , (3) one leaf comes from $T(I)$ and two from L , and (4) all three leaves come from L . Similarly, for the leaves of a resolved triplet we have the following cases: (1) two leaves come from $T(I)$ and one not from $T_1(u)$, (2) one leaf comes from $T(I)$, one from L , and one not from $T_1(u)$, and (3) two leaves come from L and one not from $T_1(u)$. Since $S^{\{v, v'\}}(T_1, T_2) = S_r^{\{v, v'\}}(T_1, T_2) + S_f^{\{v, v'\}}(T_1, T_2)$, the statement follows. ◀

■ **Algorithm 2** Computing $S_f^{\{v,v'\}}(T_1, T_2)$ and $S_r^{\{v,v'\}}(T_1, T_2)$ in $O(q)$ time.

```

1: procedure  $S_f^{\{v,v'\}}(T_1, T_2)$ 
2:   fans = 0
3:   for every internal node  $u$  in  $T_1$  do
4:     fans = fans +  $u_{\text{red,blue,green}}$ 
5:     fans = fans +  $u_{\text{red,blue}} \cdot u_{\text{greenL}} + u_{\text{red,green}} \cdot u_{\text{blueL}} + u_{\text{blue,green}} \cdot u_{\text{redL}}$ 
6:     fans = fans +  $u_{\text{redI}} \cdot u_{\text{blueL}} \cdot u_{\text{greenL}} + u_{\text{blueI}} \cdot u_{\text{redL}} \cdot u_{\text{greenL}} + u_{\text{greenI}} \cdot u_{\text{redL}} \cdot u_{\text{blueL}}$ 
7:     fans = fans +  $u_{\text{redL}} \cdot u_{\text{blueL}} \cdot u_{\text{greenL}}$ 
8:   return fans
9: procedure  $S_r^{\{v,v'\}}(T_1, T_2)$ 
10:  resolved = 0
11:  for every internal node  $u$  in  $T_1$  do
12:    resolved = resolved +  $u_{\text{red,blue}} \cdot u_{\text{white}}$ 
13:    resolved = resolved +  $u_{\text{redI}} \cdot u_{\text{blueL}} \cdot u_{\text{white}} + u_{\text{blueI}} \cdot u_{\text{redL}} \cdot u_{\text{white}}$ 
14:    resolved = resolved +  $u_{\text{redL}} \cdot u_{\text{blueL}} \cdot u_{\text{white}}$ 
15:  return resolved

```

■ **Algorithm 3** $O(qn)$ -time algorithm for computing $D(T_1, T_2)$.

```

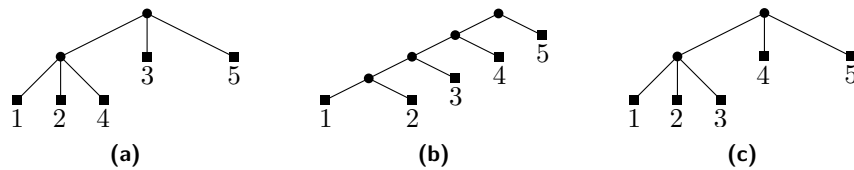
1: procedure  $D(T_1, T_2)$ 
2:   Compute the  $q \times n$  table  $C$ .
3:   For every  $u$  in  $T_1$  compute the parameter  $u_i$ , which is the number of leaves in  $T(u)$ .
4:   sharedTriplets = 0
5:   for every internal node  $v$  in  $T_2$  do
6:     for every child  $v'$  of  $v$  do
7:       Let  $R$ ,  $B$ , and  $G$  be the color ranges defined by edge  $\{v, v'\}$ 
8:       Given  $C$ ,  $R$ ,  $B$ , and  $G$ , compute the counters of  $T_1$  according to Lemma 11
9:       sharedTriplets = sharedTriplets +  $S_f^{\{v,v'\}}(T_1, T_2) + S_r^{\{v,v'\}}(T_1, T_2)$ 
10:  return  $\binom{n}{3} - \text{sharedTriplets}$ 

```

In Algorithm 3 we show how to compute $D(T_1, T_2)$. From the preprocessing step, line 2 requires $O(qn)$ time. Line 3 is performed by a depth first traversal of T_1 , thus requiring $O(n)$ time. From Lemma 11, lines 7-9 require $O(q)$ time. Since we also have that $\sum_{v \in T_2} \text{deg}(v) = O(n)$, the total time required to compute $D(T_1, T_2)$ is $O(qn)$. The correctness is ensured by Lemma 10, thus we obtain the following theorem:

► **Theorem 12.** *The rooted triplet distance between two rooted phylogenetic trees T_1 and T_2 built on the same leaf label set of size n , can be computed in $O(qn)$ time, where q is the total number of internal nodes in T_1 .*

An implementation of the algorithm in C++ is available at <https://github.com/kmampent/qtd>. Preliminary experiments indicate that our prototype implementation uses less space and is faster than the state-of-the-art, optimized implementation of the $O(n \log n)$ -time algorithm from [5] for large inputs, e.g., when $n = 1,000,000$ and $q \leq 50$. Details will be reported in the full version of the paper.



■ **Figure 4** Let $\mathcal{R} = \{12|3, 13|4, 24|5\}$. (a) The optimal tree for 2-MAXRTC induces 2 triplets from \mathcal{R} . (b) The tree returned by the BUILD algorithm from [1]. (c) The best tree obtainable by contracting all internal edges except one in the tree from (b) induces only 1 triplet from \mathcal{R} , so this method is not optimal for 2-MAXRTC.

6 Open Problems

The optimal polynomial-time approximation ratio for any fixed $q \geq 3$ is an open problem, as well as the existence of algorithms achieving that ratio. Moreover, for the special case where all the triplets in \mathcal{R} are consistent with a tree T , the computational complexity of q -MAXRTC is an open problem as well. Note that just applying BUILD [1] to obtain such a T and then trying every bipartition of L induced by an edge of T fails to produce an optimal solution to 2-MAXRTC (see Figure 4 for a counterexample). Another open problem is the existence of approximation algorithms for q -MAXRTC in the weighted case, where each triplet in \mathcal{R} has a weight and the objective is to build a tree that maximizes the total weight of the triplets induced from \mathcal{R} . This addresses the case where some triplets in \mathcal{R} are more important than others. Moreover, another open problem is the following: given a set of triplets \mathcal{R} on a leaf label set of size n and a parameter ℓ , build a tree T with ℓ leaves such that $|rt(T) \cap \mathcal{R}|$ is maximized. Just like q -MAXRTC is a combination of MINRS and MAXRTC, this new problem is a combination of the *maximum agreement supertree problem* studied in [13] and MAXRTC. Finally, for the rooted triplet distance computation, a major open problem [4, 5] is whether it can be computed in $O(n)$ time. When $q = O(1)$, our proposed algorithm runs in $O(n)$ time. If q_1 is the total number of internal nodes of one tree and q_2 of the other, is it possible to obtain an algorithm with a $O(q_1 q_2 + n)$ running time?

References

- 1 A. V. Aho, Y. Sagiv, T. G. Szlymanski, and J. D. Ullman. Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.
- 2 P. Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. *Information Processing Letters*, 57(3):151–158, 1996.
- 3 O. R. P. Bininda-Emonds. The evolution of supertrees. *Trends in Ecology & Evolution*, 19(6):315–322, 2004.
- 4 G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, T. Mailund, and A. Sand. Efficient Algorithms for Computing the Triplet and Quartet Distance Between Trees of Arbitrary Degree. In *Proc. SODA 2013*, pages 1814–1832, 2013.
- 5 G. S. Brodal and K. Mampentzidis. Cache Oblivious Algorithms for Computing the Triplet Distance between Trees. *Proc. ESA 2017*, pages 21:1–21:14, 2017.
- 6 D. Bryant. *Building Trees, Hunting for Trees, and Comparing Trees - Theory and Methods in Phylogenetic Analysis*. PhD thesis, University of Canterbury, Christchurch, NZ, 1997.
- 7 J. Byrka, P. Gawrychowski, K. T. Huber, and S. Kelk. Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks. *Journal of Discrete Algorithms*, 8(1):65–75, 2010.

- 8 J. Byrka, S. Guillelot, and J. Jansson. New Results on Optimizing Rooted Triplets Consistency. *Discrete Appl. Math.*, 158(11):1136–1147, 2010.
- 9 L. A. Hug et al. A new view of the tree of life. *Nature Microbiology*, 1, 2016.
- 10 L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the Complexity of Constructing Evolutionary Trees. *Journal of Combinatorial Optimization*, 3(2):183–197, 1999.
- 11 Johan Håstad. Some Optimal Inapproximability Results. *J. ACM*, 48(4):798–859, 2001.
- 12 J. Jansson, R. S. Lemence, and A. Lingas. The Complexity of Inferring a Minimally Resolved Phylogenetic Supertree. *SIAM Journal on Computing*, 41(1):272–291, 2012.
- 13 J. Jansson, Joseph H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted Maximum Agreement Supertrees. *Algorithmica*, 43(4):293–307, 2005.
- 14 J. Jansson, R. Rajaby, and W.-K. Sung. Minimal Phylogenetic Supertrees and Local Consensus Trees. *AIMS Medical Science*, 5:181, 2018.
- 15 V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the Hardness of Approximating MAX k -CUT and Its Dual. *Chicago Journal of Theoretical Computer Science*, 1997.
- 16 J. M. Lang, A. E. Darling, and J. A. Eisen. Phylogeny of bacterial and archaeal genomes using conserved genes: supertrees and supermatrices. *PLoS ONE*, 8(4), 2013.
- 17 K. J. Locey and J. T. Lennon. Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences*, 2016.
- 18 A. McKenzie and M. Steel. Distributions of cherries for two models of trees. *Mathematical Biosciences*, 164(1):81–92, 2000.
- 19 L. Trevisan. Parallel Approximation Algorithms by Positive Linear Programming. *Algorithmica*, 21(1):72–88, 1998.
- 20 D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*, pages 108–109. Cambridge University Press, New York, NY, USA, 1st edition, 2011.
- 21 B. Y. Wu. Constructing the Maximum Consensus Tree from Rooted Triples. *Journal of Combinatorial Optimization*, 8(1):29–39, 2004.
- 22 U. Zwick. Approximation Algorithms for Constraint Satisfaction Problems Involving at Most Three Variables Per Constraint. *Proc. SODA 1998*, pages 201–210, 1998.

Alignment- and Reference-Free Phylogenomics with Colored de Bruijn Graphs

Roland Wittler 

Genome Informatics, Faculty of Technology, Bielefeld University, Germany

Center for Biotechnology, Bielefeld University, Germany

<https://www.cebitec.uni-bielefeld.de/~roland>

roland.wittler@uni-bielefeld.de

Abstract

We present a new whole-genome based approach to infer large-scale phylogenies that is alignment- and reference-free. In contrast to other methods, it does not rely on pairwise comparisons to determine distances to infer edges in a tree. Instead, a colored de Bruijn graph is constructed, and information on common subsequences is extracted to infer phylogenetic splits. Application to different datasets confirms robustness of the approach. A comparison to other state-of-the-art whole-genome based methods indicates comparable or higher accuracy and efficiency.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Molecular sequence analysis

Keywords and phrases Phylogenomics, phylogenetics, phylogenetic splits, colored de Bruijn graphs

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.2

Acknowledgements I thank Guillaume Holley for support on Bifrost, Nina Luhmann for pointers to data sets, and Andreas Rempel for programming assistance.

1 Introduction

A common task in comparative genomics is the reconstruction of the evolutionary relationships of species or other taxonomic entities, their *phylogeny*. Today's wealth of available genome data enables large-scale comparative studies, where phylogenetics is faced with the following problems: first, the sequencing procedure itself is becoming cheaper and faster, but finishing a genome sequence remains a laborious step. Thus, more and more genomes are published in an unfinished state, i.e., only assemblies (composed of contigs), or raw sequencing data (composed of read sequences) are available. Hence, traditional approaches for phylogenetic inference can often not be applied, because they are based on the identification and comparison of marker sequences, which relies on computing multiple alignments – an NP-hard task in theory, and in practice even heuristics are often too slow. Second, the low sequencing cost allow new large-scale studies of certain niches and/or aloof from model organisms, where reference sequences would be too distant or not available at all.

Whole-genome approaches that are usually alignment- and reference-free solve these problems, see e.g. [5, 7, 13, 18, 19, 22]. However, the sheer number of genomes to be analysed is still posing limits in large-scale scenarios as almost all whole-genome approaches are based on a pairwise comparison of some characteristics of the genomes (e.g. occurrences or frequencies of k -mers or other patterns) to define distances which are then used to reconstruct a tree (e.g. by using neighbor joining [15]). This means, for n genomes, $O(n^2)$ comparisons are performed in order to infer $O(n)$ edges. To the best of our knowledge, only MultiSpaM [4] follows a different approach by sampling small, gap-free alignments involving *four* genomes each, which are used to infer a super tree on quartets. According to our experiments, this method is not suitable for large-scale settings (see Results), though.



© Roland Wittler;

licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 2; pp. 2:1–2:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Apart from computational issues, the actual objective of phylogenetic inference in terms of how to represent a phylogeny is not obvious in the first place. Taking only intra-genomic mutations into account, i.e., assuming a genome mutating independently of others, genomes would have unique lines of ancestors and their phylogeny would thus be a tree. Several reasons however conflict this simple tree model. Inter-genomic exchange of genomic segments such as crossover in diploid or polyploid organisms, lateral gene transfer in bacteria, or introgression in insects contradict the assumption of unique ancestry. Furthermore, incomplete, ambiguous, or even misleading information can hamper resolving a reliable phylogenetic tree.

Here, we propose a new methodology that is whole-genome based, alignment- and reference-free, and does not rely on a pairwise comparison of the genomes or their characteristics. An implementation called SANS (“Symmetric Alignment-free phylogeNomic Splits”) is available at <https://gitlab.ub.uni-bielefeld.de/gi/sans>. The k -mers of all genomic sequences (assemblies or reads) are stored in a colored de Bruijn graph, which is then traversed to extract phylogenetic signals. The reconstructed phylogenies are not restricted to trees. Instead, the generalized model of *phylogenetic splits* [2] is used to infer phylogenetic networks that can indicate a tree structure and also point to ambiguity in the reconstruction.

In the following Section 2, we will first introduce two building blocks of our approach, *splits* and *colored de Bruijn graphs*. Then, we will describe and motivate our method in Section 3. After an evaluation on several real data sets in Section 4, we will give a brief summary and an outlook in Section 5.

2 Background

Before presenting our method in Section 3, we will introduce two basic concepts it builds upon. Firstly, as motivated above, our phylogenies will be represented by sets of *splits*, a generalization of trees. Secondly, to extract phylogenetic signals from the given genomes in the first place, they are stored in a *colored de Bruijn graph*.

2.1 Phylogenetic splits

In the following, we briefly recapitulate some notions and statements from the split decomposition theory introduced by Bandelt and Dress [2], and put them into context.

► **Definition 1** (Unordered split). *Given a set O , if for two subsets $A, B \subseteq O$, both $A \cap B = \emptyset$ and $A \cup B = O$, then the unordered pair $\{A, B\}$ is a bipartition or (unordered) split of O . If either A or B is empty, a split is called trivial.*

We extend the above commonly used terminology of (unordered) splits to *ordered splits* – a central concept in our approach.

► **Definition 2** (Ordered split). *If $\{A, B\}$ is an unordered split of O , the ordered pairs (A, B) and (B, A) are ordered splits. (B, A) is called the inverse (split) of (A, B) and vice versa.*

Note that one unordered split $\{A, B\} = \{B, A\}$ corresponds to two ordered splits $(A, B) \neq (B, A)$. Our method will first infer ordered splits and their inverse, which will then be combined to form unordered splits. If clear from the context, we may denote an ordered split (A, B) by simply A .

A set of splits \mathcal{S} may be supplemented with weights $w : \mathcal{S} \rightarrow \mathbb{R}$, e.g., in [2], splits are weighted by a so-called *isolation index*. Strong relations between metrics and sets of weighted unordered splits have been shown. In particular, one can canonically decompose any distance d into a set of weighted splits \mathcal{S}_d that is *weakly compatible* in the following sense.

► **Definition 3** (Weak compatibility [2]). *A set of unordered splits \mathcal{S} on O is weakly compatible if for any three splits $\{A_1, B_1\}, \{A_2, B_2\}, \{A_3, B_3\} \in \mathcal{S}$, there are no elements $a, a_1, a_2, a_3 \in O$ with $\{a, a_1, a_2, a_3\} \cap A_i = \{a, a_i\}$ for $i = 1, 2, 3$.*

Then, $d(a, b) = \sum_{\{A, B\} \in \mathcal{S}_d} w(\{A, B\}) \delta_A(a, b) + d_0$ where $\delta_A(a, b) := 1$ if either a or b is in A , but not both, and $\delta_A(a, b) := 0$ otherwise, i.e., the weights of all splits separating a from b are accumulated, and where d_0 is a so-called *split prime* residue that cannot be decomposed further.

As a peculiarity of our approach is being *not* distance-based, we mention the above relation of weakly compatible splits and distances only for the sake of completeness. We will make use of the above property to filter a general set of splits such that it can be displayed as a – in most cases planar – network.

A distance d is a tree metric (also called *additive*), if and only if there is a set of splits \mathcal{S}_d with $d(a, b) = \sum_{\{A, B\} \in \mathcal{S}_d} w(\{A, B\}) \delta_A(a, b)$ that is *compatible* in the following sense.

► **Definition 4** (Compatibility [2]). *A set of unordered splits \mathcal{S} on O is compatible if for any two splits $\{A, B\}$ and $\{A', B'\}$, one of the four intersections $A \cap A', A \cap B', B \cap A',$ and $B \cap B'$ is empty.*

We will make use of the implied one to one correspondence of edges in a tree and compatible splits: an edge of length w whose removal separates a tree into two trees with leaf sets A and B , respectively, corresponds to a split $\{A, B\}$ of weight w .

2.2 Colored de Bruijn graphs

A *string* s is a sequence of characters over a finite, non-empty set, called *alphabet*. Its *length* is denoted by $|s|$, the character at position i by $s[i]$, and the substring from position i through j by $s[i..j]$. A string of length k is called *k-mer*.

We consider a *genome* as a set of strings over the DNA alphabet $\{A, C, G, T\}$. The *reverse complement* of a string s is $\bar{s} := \overline{s[|s|]} \cdots \overline{s[1]}$, where $\overline{A} := T, \overline{C} := G, \overline{G} := C, \overline{T} := A$.

An abstract data structure that is often used to efficiently store and process a collection of genomes is the *colored de Bruijn graph (C-DBG)* [11]. It is a node-labeled graph (V, E, col) where each vertex $v \in V$ represents a k -mer associated with a set of colors $col(v)$ representing the set of genomes the k -mer occurs in. A directed edge from v to v' exists if and only if for the corresponding k -mers x and x' , respectively, $x[2..k] = x'[1..k-1]$. We call a path $p = v_1, \dots, v_l$ of length $|p| = l$ in a C-DBG *non-branching* if all contained vertices have an in- and outdegree of one with the possible exception of v_1 having an arbitrary indegree and v_l having an arbitrary outdegree, and it has the same set of colors assigned to all its vertices. A maximal non-branching path is a *unitig*. In a *compact C-DBG*, all unitigs are merged into single vertices.

In practice, since a genomic sequence can be read in both directions, and the actual direction of a given sequence is usually unknown, a string and its reverse complement are assumed equivalent. Thus, in many C-DBG implementations, both a k -mer and its reverse complement are represented by the same vertex. In the following, we will assume this being internally handled by the data structure.

3 Method

The basic idea of our new approach is that a sequence which is contained as substring in a subset A of all genomes G but not contained in any of the other genomes is interpreted as a signal that A should be separated from $G \setminus A$ in the phylogeny. The more of those sequences exist and the longer they are, the stronger is the signal for separation.

■ **Algorithm 1** SANS: Symmetric, Alignment-free phylogenomic Splits.

```

INPUT: List of genomes G
OUTPUT: Weighted splits over G
T := empty trie // initialize T[S] := (0,0) on first access by S
C-DBG := colored de Bruijn graph of G
foreach unitig U in C-DBG:
  S := color list of U (sublist of G)
  // add ordered split S or its inverse G\S to trie
  if |S| < |G|/2 or ( |S| == |G|/2 and S[0] == G[0] ) then:
    increase first element of T[S] by length of U
  else:
    increase second element of T[G\S] by length of U
foreach entry S in T with values (w,w'):
  output unordered split {S,G\S} of weight sqrt(w*w')

```

To efficiently extract common sequences, we first construct a C-DBG of all given genomes. Then, we collect all separation signals as ordered splits, where any unitig u contributes $|u|$ to the weight of an ordered split $col(u)$. Since both an ordered split (A, B) and its inverse (B, A) indicate that A and B should be separated in the phylogeny, we combine them to one unordered split $\{A, B\}$ with an overall weight that is a combination of the individual weights. The individual steps will be explained in more detail next.

C-DBG

Among several available implementations of C-DGBs (e.g. [1, 9, 11, 14]), we decided to use Bifrost [8] (<https://github.com/pmelsted/bifrost>) for the following reasons: it is easy to install and use; it is efficiently implemented; it can process full genome sequences, assemblies, read data or even combinations of these; for read data as input, it offers some basic assembly-like filtering of k -mers; and it realizes a compacted C-DBG and provides a C++ API such that a traversal of the unitigs could be easily and efficiently implemented – only unitigs with heterogeneous color sets had to be split, because colors are not considered during compaction.

Like other implementations of DBGs on the DNA alphabet, Bifrost saves space by not storing edges explicitly – with the trade-off of having to determine neighboring vertices by querying the graph for all possible preceding and succeeding k -mers. Since we do not make use of the topology of the C-DBG, this common design decision accommodates our needs.

Accumulating split weights

Because splits often share many genomes, we use a trie data structure to store a split (as key) as path from the root to a terminal vertex, along with its weight (as value) assigned to the terminal vertex. We represent the set of genomes G as a list with some fixed order, and any subset of G as sublist of G , i.e., with the same relative order. For a split (A, B) and its inverse (B, A) , we take as key the shorter of A and B , breaking ties by selecting that split containing $G[0]$, and as value the pair of weights (w, w') , where w is the accumulated weight of the key, and w' the accumulated weight of its inverse. When the trie is accessed for a key the first time, the value is initialized with $(0, 0)$.

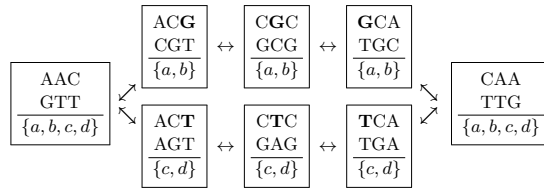
The overall method SANS is shown in Algorithm 1, the very last step of which will be motivated in the following.

Combining splits and their inverses

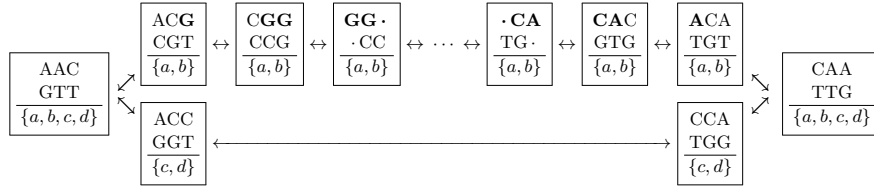
To combine an ordered split (A, B) of weight w_A and its inverse (B, A) of weight w_B , a naive argument would be: both indicate the same separation, so they should be taken into account equivalently, and thus take the sum $w_A + w_B$ or arithmetic mean $(w_A + w_B)/2$. However, in our evaluation, this weighting scheme often assigned higher weight to wrong splits than to correct splits (compared to reliable reference trees; exemplified in Section 4.1). Instead, we revert the above argument: consider a mutation on a (true) phylogenetic branch separating the set of genomes into subgroups A and B . The corresponding two variants of the affected segment will induce two unitigs with color sets A and B , respectively. Under the infinite sites assumption, these unitigs would not be affected by other events. So, each mutation on a branch in the phylogeny contributes to *both* splits (A, B) and (B, A) . We hence take the geometric mean $\sqrt{w_A \cdot w_B}$ such that in case of asymmetric splits, the lower weight diminishes the total weight, and only symmetric splits receive a high overall weight.

Considering different scenarios that would affect the observation of common substrings in the C-DBG, some of which are illustrated in Figure 1, we observe beneficial behavior of the weighting scheme in almost all cases: A **single nucleotide variation** would cause a bubble in the C-DBG composed of two unitigs of similar length k each – a symmetric scenario in accordance with the above weighting scheme. Both an **insertion or deletion** of length l would cause an asymmetric bubble and thus asymmetric weights $k - 1$ and $l + k - 1$. Here, the geometric mean has the positive effect to weaken the impact of the length of the event on the overall split weight. E.g., the total weight for x deletions of length l would increase linearly with x whereas those for one deletion of length $x \cdot l$ would increase with \sqrt{x} . For both a **transposition or inversion** of arbitrary length, the color set of the segment itself remains the same, and only those k -mers spanning the breakpoint regions would be affected, inducing symmetric bubbles in accordance with the weighting scheme. **Lateral gene transfer** is challenging phylogenetic reconstruction, because a subsequence of length l that is contained in both the group A containing the donor genome as well as the target genome b from the other genomes $B := G \setminus A$ can easily be misinterpreted as a signal to separate $A \cup \{b\}$ from the remainder $B \setminus \{b\}$ instead of separating A from B , where the strength of this erroneous signal grows with l . Our approach will be affected only little: on the one hand, the unitig corresponding to the copied subsequence has color set $A \cup \{b\}$ and thus contributes to an ordered split $(A \cup \{b\}, B \setminus \{b\})$ of weight $l - k + 1$. On the other hand, because the transfer does not remove any subsequence in the donor sequence, only those k k -mers spanning the breakpoint region will be affected, inducing a unitig with color set $B \setminus \{b\}$ whose length is independent of l . **Missing or additional data** may arise from genomic segments that are difficult to sequence or assemble and might thus be missing in some assemblies, due to the usage of different sequencing protocols, assembly tools, or filter criteria, or simply because some input files contain plasmid or mitochondrial sequences and others do not. This does not affect our approach, because additional sequence induces unitigs and thus an ordered split, but the absence of sequence does not induce any split, not even due to breakpoint regions, because in such cases usually whole reads, contigs or chromosomes are involved. Thus, the weight of the additional ordered split would be multiplied by zero for the absent split, resulting in a total weight of zero. **Copy number changes** can only be detected if the change is from one to two or *vice versa*, adding or removing k -mers spanning the juncture of the two copies. Beyond that, because the k -mer counts are not captured, our approach is not sensible for copy number changes.

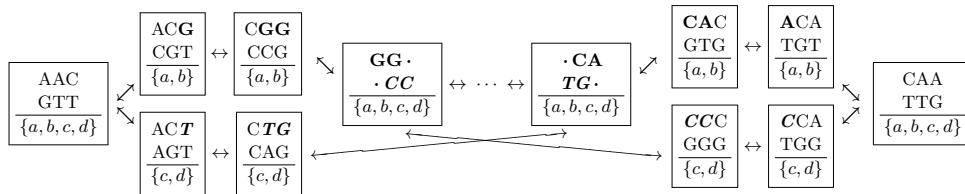
In practice, the structure of a C-DBG is much more complex than the simplified picture we draw above. Nevertheless, using the geometric yields high accuracy of the approach compared to other methods.



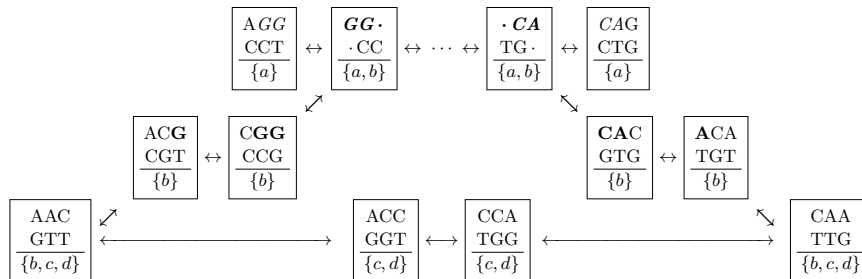
(a) Single nucleotide variation in genomes $a = b = \text{AACGCAA}$ and $c = d = \text{AACTCAA}$. The induced ordered split $\{a, b\}$ and its inverse $\{c, d\}$ of weight $k = 3$ each yield a corresponding unordered split $\{\{a, b\}, \{c, d\}\}$ of weight $\sqrt{k}k = k = 3$.



(b) Insertion/deletion of length $l = 4$ (or longer, indicated by dots) in genomes $a = b = \text{AACGG}\dots\text{CACAA}$ and $c = d = \text{AACCAA}$. The induced ordered split $\{a, b\}$ of weight $l + k - 1 = l + 2$ and its inverse $\{c, d\}$ of constant weight $k - 1 = 2$ yield a corresponding unordered split $\{\{a, b\}, \{c, d\}\}$ of weight $\sqrt{(l + k - 1)(k - 1)} = \sqrt{2(l + 2)}$.



(c) Inversion of length $l = 4$ (or longer, indicated by dots) between genomes $a = b = \text{AACGG}\dots\text{CACAA}$ and $c = d = \text{AACTG}\dots\text{CCCAA}$. The induced ordered split $\{a, b\}$ and its inverse $\{c, d\}$ of constant weight $2(k - 1) = 4$ each yield a corresponding unordered split $\{\{a, b\}, \{c, d\}\}$ of constant weight $\sqrt{2(k - 1)2(k - 1)} = 2(k - 1) = 4$.



(d) Lateral gene transfer of length $l = 4$ (or longer, indicated by dots) from genome $a = \text{AGG}\dots\text{CAG}$ to $b = \text{AACGG}\dots\text{CACAA}$ but not to $c = d = \text{AACCAA}$. Apart from mutation-independent splits for the boundaries, and the trivial split $\{b\}$ (without its inverse), the split $\{a, b\}$ of weight $l - k + 1 = l - 2$ and its inverse $\{c, d\}$ of constant length $k - 1 = 2$ are induced, yielding a corresponding unordered split $\{\{a, b\}, \{c, d\}\}$ of weight $\sqrt{(l - k + 1)(k - 1)} = \sqrt{2(l - 2)}$.

■ **Figure 1** Toy examples for different mutations within four genomes a, b, c and d to illustrate their effect on a C-DBG with $k = 3$. Each vertex of the C-DBG is labelled with both its k -mer and the reverse complement (in arbitrary order), as well as its color set. Due to the small value of k , the C-DBG contains edges corresponding to pairs of overlapping k -mers that are not contained in the given strings. For the purpose of clarity, these are not drawn. Mutations are highlighted in bold and/or italics.

Postprocessing

Even though the geometric mean filters out many asymmetric splits, the total number of positively weighted splits can be many-fold higher than $2n - 3$, the number of edges in a fully resolved tree for n genomes. Unfortunately, the observed distribution of split weights did not indicate any obvious threshold to separate high-weighted splits from low-weighted noise. Nevertheless, a rough cutoff can safely be applied by keeping only the t highest weighting splits, e.g., in our evaluation $t = 10n$ has been used for all datasets. Additionally, we evaluated two filtering approaches: *greedy weakly*, i.e., greedily approximating a maximum weight subset that is weakly compatible and can thus be displayed as a network, and *greedy tree*, i.e., greedily approximating a maximum weight subset that is compatible and thus corresponds to a tree. To this end, we used the corresponding options of the software tool SplitsTree [10, 12]. As we will demonstrate in the Results section, in particular the tree filter proved to be very effective in practice.

Run time complexity

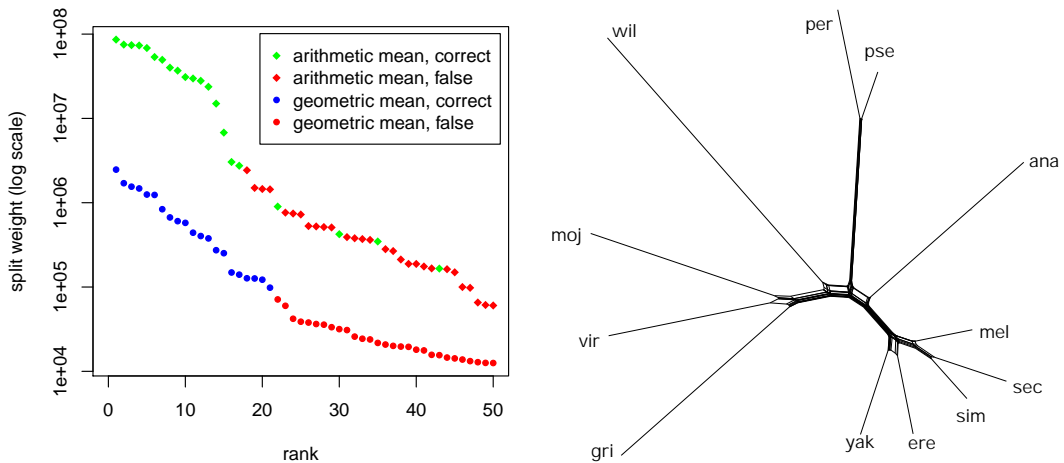
Consider n genomes of length $O(m)$ each. In Bifrost, the compacted C-DBG is built by indexing a k -mer by its *minimizer*, i.e., a substring with the smallest hash value among all substrings of length g in a k -mer. According to the developers of Bifrost (personal communication), inserting a k -mer and its color takes $O(4^{(k-g)} \log(n))$ time in the worst case. In practice, however, each of the $O(mn)$ k -mers can be inserted in $O(\log(n))$ time, and hence, building the complete C-DGB takes $O(mn \log(n))$ time. While iterating over all positions in the graph, we verify whether a unitig has to be split due to a change in the color set. Because each of the n genomes adds $O(m)$ color assignments to the graph, we have to do $O(mn)$ color comparisons in total, which does not increase the overall complexity.

Each genome contributes to at most $O(m)$ ordered splits. So the sum of the cardinality of all ordered splits, i.e., the total length of all splits in Algorithm 1, is $O(mn)$. Hence, the insertion and lookups of all \mathbf{S} in trie \mathbf{T} takes $|\mathbf{S}|$ time each and $O(mn)$ in total, and the number of terminal vertices of \mathbf{T} , i.e., the final number of unordered splits, is in $O(mn)$, too. For ease of postprocessing, splits are ordered by decreasing weight, increasing the run time for split extraction to $O(mn \log(mn))$, or $O(mn \log(n))$ to output only the t , $t \in O(n)$, highest weighting splits, respectively.

4 Results

In this section, we present several use cases in order to exemplify robustness and different other characteristics of our approach SANS. We compare to the following other whole-genome based reconstruction tools.

MultiSpaM [4] samples a constant, high number of small, gap-free alignments of four genomes. The implied quartet topologies are combined to an overall tree topology. To the best of our knowledge, all other tools are distance-based and rely on pairwise comparisons. Interestingly, although all methods are based on lengths or numbers of common subsequences or patterns, their results differ considerably from those of SANS. Co-phylog [18] analyses each genome in terms of certain patterns (*C-grams*, *O-grams*) and compares their characteristics (*context*). In andi [7], enhanced suffix arrays are used to detect pairs of maximal unique matches that are used to anchor ungapped local alignments, based on which pairwise distances are computed. CVTree3 [22] corrects k -, $k-1$, and $k-2$ -mer counts by subtracting random background of neutral mutations using a $(k-2)$ -th Markov assumption. In FSWM [13], matches of patterns including match and *don't-care* position are scored and filtered to estimate evolutionary distances.



(a) Comparison of accuracy for using arithmetic or geometric mean for combining weights of splits and their inverse each. Splits have been sorted by the combined weight and the 50 highest weighting splits are shown. Color indicates whether a split agrees with the reference [17].

(b) Visualization of greedily extracted weakly compatible subset of splits using SplitsTree [10, 12]. As by default, geometric mean has been used for combining weights of splits and their inverse each.

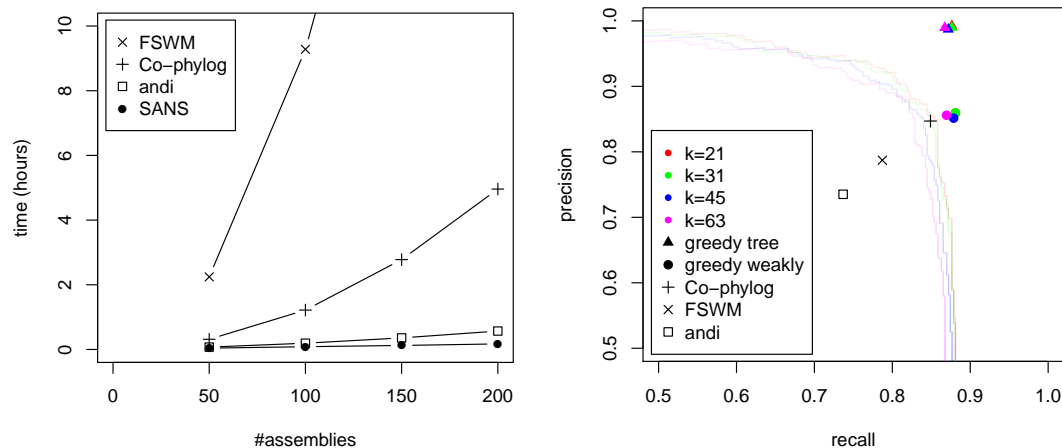
■ **Figure 2** Reconstructed phylogenetic splits on the *Drosophila* dataset [17].

Unless stated otherwise, a k -mer length of 31 (Bifrost default) has been used for constructing the C-DBG for SANS. All tools have been run on a single 2 GHz processor and times are given in CPU hours (user time). Accuracy has been measured in terms of topological Robinson-Foulds distance, i.e., a predicted edge (split) is correct if and only if the reference tree contains an edge that separates the same two sets of leaves. As *recall*, we count the number of correct edges (splits) divided by the total number of edges in the reference, and as *precision*, we count the number of correct edges (splits) divided by the total number of predicted edges (splits).

4.1 *Drosophila*

This dataset comprises assemblies from 12 species of the genus *drosophila* obtained from the database FlyBase (flybase.org, latest release before Feb. 2019 of all-chromosome-files each) [17]. As reference, we consider the commonly accepted phylogeny published by the FlyBase consortium [3, Figure 2] also shown on the database website.

Although being “simple” in the sense that it contains only a small number of genomes, its analysis exemplifies the following aspects: (i) The effectiveness of our method for medium sized input files: for a total of more than 2 161 Mbp (180 Mbp on average), SANS inferred the correct tree within 168 minutes and using up to 25 GB of memory. We ran CVTree3 with various values of k . In the best cases ($k = 12$ and 13), 7 of 9 internal edges have been inferred correctly taking 95 and 162 minutes, and up to 26 and 87 GB of memory, respectively. (For $k = 11$, only 4 internal edges were correct, and for $k > 13$, the computation ran out of memory.) Both Co-phylog and FSWM did not finish within 48 hours, and both MultiSpaM and andi could not process this dataset successfully. (ii) As can be seen in Figure 2a, combining splits and their inverse using the geometric instead of the arithmetic mean strengthens the tendency of correct splits having a high weight. (iii) Even though the reconstruction shown in Figure 2b contains 45 splits – in comparison to 21 edges in a binary tree –, the visualization is close to a tree structure.



(a) Running time for computing phylogenies on random subsamples. Times for SANS include DBG construction with $k = 31$, split extraction and agglomeration.

(b) For different values of k , weakly compatible subsets (bullets) and trees (triangles) have been greedily extracted. Each point on a line corresponds to a different threshold to discard low weighting splits.

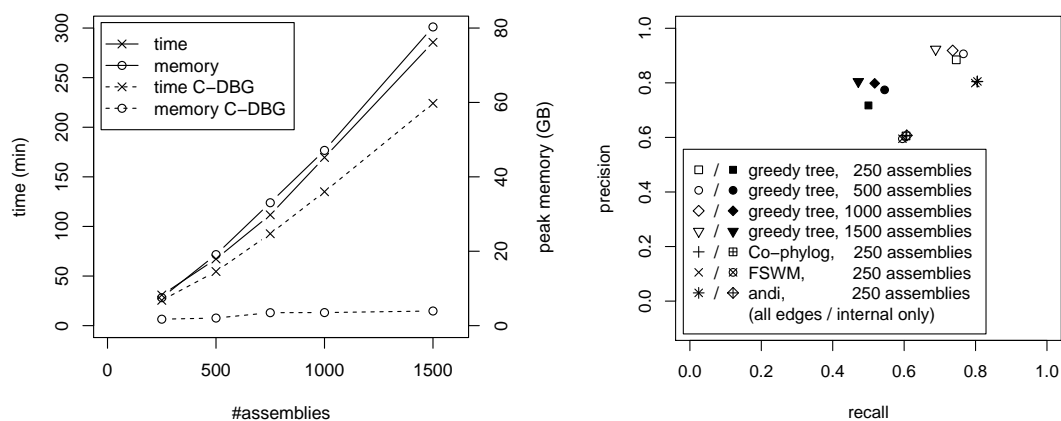
■ **Figure 3** Comparison of running time and accuracy of different methods on the ParaC dataset [20] comprising assemblies of $n = 220$ genomes.

4.2 *Salmonella enterica* Para C

This dataset is of special interest as the contained assemblies from 220 genomes of different serovars within the *Salmonella enterica* Para C lineage include that of an ancient Paratyphi C genome obtained from 800 year old DNA [20], the placement of which is especially difficult due to missing data. As reference, we consider a maximum-likelihood based tree on nonrecombinant SNP data [20, Figure 5a].

We studied the running time behaviour of the different methods for random subsamples of increasing size. As shown in Figure 3a, for this high number of closely related genomes, we observed a super-linear running time of up to 41 minutes for andi, about 5 hours for Co-phylog, and up to 43 hours for FSWM, whereas the reconstruction of SANS shows a linear increase (Pearson correlation coefficient 0.9994) to about 10 minutes. The memory requirement of both SANS and Co-phylog remained below 0.5 GB, whereas andi required about 1 GB, and FSWM required up to about 17 GB. We ran CVTree3 with ten values of k between 5 and 27, but none of the resulting trees contained more than 5 correct internal edges. For MultiSpaM, we increased the number of sampled quartets from the default of 10^6 to up to 10^8 , which increased the running time from about one hour to about 66 hours. Both recall and precision improved but were still below 0.2 for internal edges.

The accuracy of the reconstructions with respect to the reference is visualized in Figure 3b. In particular, we observe: (i) the split reconstruction by SANS and the tree inferred by Co-phylog are comparably accurate and both are more accurate than the FSWM and andi tree, (ii) greedily extracting high weighting splits to filter for a tree selects correct splits while discarding false splits with very high precision, (iii) greedily extracting high weighting splits to filter for a weakly compatible subset also selects correct splits, but, as expected, has a lower precision than the tree filter, because more splits are kept than there are edges in a tree, and (iv) the results of SANS are robust for a wide range of k from 21 to 63.



(a) Running time and peak memory usage of SANS. Values including C-DBG construction, split extraction and agglomeration, as well as C-DBG construction only are given.

(b) Accuracy with respect to the reference phylogeny [21, Fig. 2A].

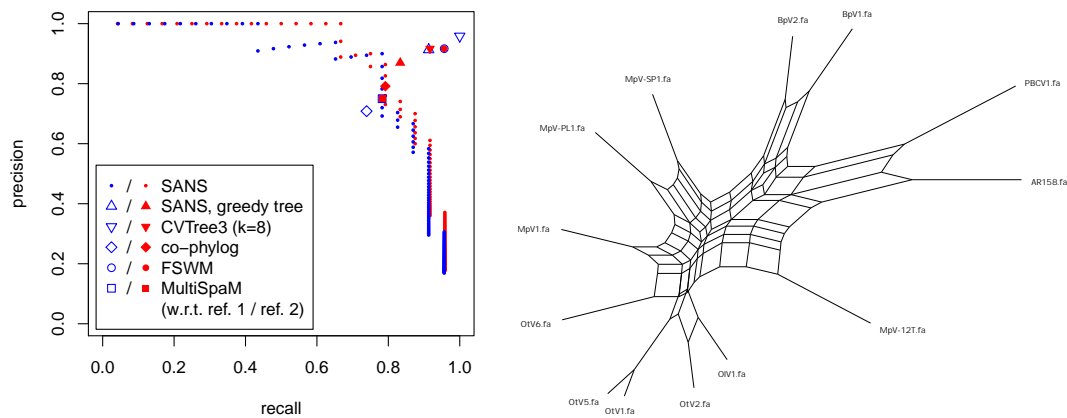
■ **Figure 4** Efficiency and accuracy on the *Salmonella enterica* dataset [21]. Values have been averaged over processing two random subsamples each.

4.3 *Salmonella enterica* subspecies *enterica*

In comparison to the ParaC dataset, the 2964 genomes studied by Zhou et al. [21] are not only a larger but also a more diverse selection of *Salmonella enterica* strains. As reference, we consider a maximum-likelihood based tree on 3002 concatenated core genes [21, Figure 2A, supertree 3].

The probability to observe long k -mers that are conserved in such a high number of more diverse genomes is lower than for the previous datasets. Hence, we selected a smaller k -mer length of $k = 21$. To assess the efficiency and accuracy for increasing number of genomes, we sampled subsets of up to 1500 assemblies. To process the smallest considered subsample of size 250, andi took about 110 minutes, whereas Co-phylog and FSWM took already more than 9 and 50 hours, respectively, and MultiSpam was not able to process this dataset at all. We ran CVTree3 with all values of k between 6 and 14, but in the best case ($k = 8$), the resulting tree contained only 33 (of 247) correct internal edges such that we did not further consider CVTree3 in our evaluation.

The memory usage for split extraction and agglomeration clearly dominates those of the C-DBG construction by Bifrost such that processing the complete dataset was not possible with our current implementation of SANS. Figure 4a shows a slightly super-linear runtime and memory consumption of up to about 300 minutes and 80 GB for processing 1500 assemblies. As can be seen in Figure 4b, both precision and recall vary only slightly for this wide range of input size. Keeping in mind that a final split of high weight strictly requires the observation of *both* unordered pairs, this is a quite promising result for this first investigation of the methodology. In particular, whereas for distance-based methods, all leaf-edges are inferred by construction and can never be false, a trivial split separating a leaf from the remaining tree, requires not only some sequence(s) unique to the leaf but also sequences that are unique to all other $n-1$ genomes. Also note that measuring accuracy by counting correct and false splits corresponding to the topological Robinson-Foulds distance has to be interpreted with care. A single misplaced leaf breaks all splits between its correct and actual location. However, this is a desired behaviour in this context, because, in a phylogeny of several hundred genomes, each genome should at least be located in the correct area, whereas the complete misplacement even of a single genome can easily lead to wrong biological conclusions.



(a) Accuracy of different tools w.r.t. two reference trees [6, Figures 3 and 4] shown in blue and red, respectively. For SANS, each point corresponds to a different threshold to discard low weighting splits.

(b) Visualization of greedily extracted weakly compatible subset of splits using SplitsTree [10, 12].

■ **Figure 5** Reconstruction results on the prasinovirus dataset [6].

4.4 Prasinoviruses

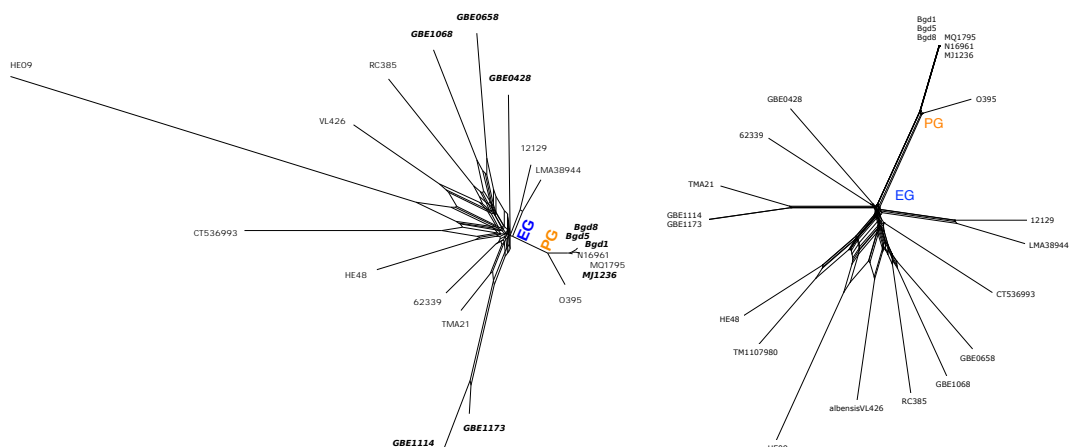
Viral genomes are short and highly diverse – posing the limits of phylogenetic reconstruction based on sequence conservation. Here we consider complete genomes of 13 prasinoviruses, which are relatively large (213 Kbp on average) [6]. As references, we consider two trees reported in the original study, one of which is based on the presence and absence of shared putative genes [6, Figure 3], and the other is a maximum likelihood estimation based on a marker gene (DNA polymerase B) [6, Figure 4].

Due to the small size of the input, it could be processed by all tools, where time and memory consumption were negligible. Only andi could not process this dataset successfully (“very little homology was found”). Results are shown in Figure 5a. The visualization of the predicted splits in Figure 5b exemplifies the explanatory power of the split framework. While main separations supported by both reference trees are recognizable as strong splits in the net, separations in which the two reference trees disagree are also shown as weakly compatible splits.

4.5 *Vibrio cholerae*

The dataset comprises 22 genomes from the species *Vibrio cholerae*, 7 of which have been sequenced from clinical samples and are labelled “pandemic genome” (PG), and the remaining 15 have been sequenced from non-clinical samples and are labelled “environmental genome” (EG) [16, primary dataset]. As already observed in the original study, for these genomes, it is difficult to reconstruct a reliable, fully resolved tree. Nevertheless, representing the phylogeny in form of splits shows a strong separation of the pandemic from the environmental group. The phylogeny presented by the authors of the original study [16, Supplementary Figure 1a] is based on 126 099 sites extracted from alignment blocks.

Comparing our reconstruction results to the reference, both shown in Figure 6, we make two observations. (i) Our reconstruction also separates the pandemic from the environmental group, and agrees to the reference in further sub-groups. (ii) When collecting the sequence data, for some of the genomes, we found assemblies, whereas for others, only read data was



(a) Visualization of greedily extracted weakly compatible subset of splits. For taxa highlighted in bold, only read data was available on NCBI (input option `-s` of Bifrost has been used); for Taxon TM1107980, no data was available on NCBI (February 2019).

(b) Reference phylogeny. Figure reprinted from Shapiro et al. [16, Supplementary Figure 1a].

■ **Figure 6** Splits reconstructed for the *V. cholerae* dataset [16] by SANS (left) and by Shapiro et al. [16] (right) visualized with SplitsTree [10, 12].

available. Because the used C-DBG implementation Bifrost supports a combination of both types as input, we were able to reconstruct a joint phylogeny without extra effort or obvious bias in the result.

5 Discussion and Outlook

We proposed a new k -mer based method for phylogenetic inference that neither relies on alignments to a reference sequence nor on pairwise or multiple alignments to infer markers. Prevailing whole-genome approaches perform pairwise comparisons to determine a quadratic number of distances to finally infer a linear number of tree edges. In contrast, in our approach, the length of conserved sequences is extracted from a colored de Bruijn graph to first infer signals for phylogenetic sub-groups. These signals are then combined with a symmetry assumption to weighted phylogenetic splits. Evaluations on several real datasets have proven comparable or better efficiency and accuracy compared to other whole-genome approaches. Our results indicate robustness in terms of k -mer length, as well as the taxonomic order, size and number of the genomes. The analysis of a dataset composed of both assembly and read data indicated also robustness in this regard – an important characteristic, which we want to investigate further.

A distinctive feature of the proposed methodology is the direct association of a phylogenetic split to the conserved subsequences it has been derived from, which is not possible for distance-based methods. We plan to enrich our implementation with this valuable possibility to allow the analysis of characteristic subsequences of identified subgroups, or subsequences inducing phylogenetic splits off the main tree, e.g. horizontal gene transfer. Here, the applied generalization of trees plays an important role, e.g., circular split systems are more strict than weakly compatible sets and might thus be a promising alternative to be studied further.

Another direction of future work is the incorporation of the topology of the de Bruijn graph. Currently, it is simply used as a collection of units. But specific substructures, in particular with regard to the colors in the graph, could be used to identify phylogenetic events.

Finally, we want to emphasize the simplicity of the new approach as presented here. At its current state, apart from iterating a colored de Bruijn graph and agglomerating unitig lengths, the only elaborate ingredient so far is the symmetry assumption realized by applying the geometric mean. We believe that the general approach still harbors much potential to be further refined by, e.g., statistical models, advanced data structures, pre- or postprocessing, to further increase its accuracy and efficiency.

References

- 1 Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: a succinct colored de Bruijn graph representation. In *International Workshop on Algorithms in Bioinformatics (WABI 2017)*, volume 88, pages 18:1–18:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
- 2 Hans-Jürgen Bandelt and Andreas WM Dress. Split decomposition: a new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*, 1(3):242–252, 1992.
- 3 Madeline A Crosby, Joshua L Goodman, Victor B Strelets, Peili Zhang, William M Gelbart, and the FlyBase Consortium. FlyBase: genomes by the dozen. *Nucleic Acids Research*, 35(suppl_1):D486–D491, 2006.
- 4 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 *Proc. of RECOMB Comparative Genomics*, pages 227–241. Springer, 2018.
- 5 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, 2015.
- 6 Jan Finke, Danielle Winget, Amy Chan, and Curtis Suttle. Variation in the genetic repertoire of viruses infecting *Micromonas pusilla* reflects horizontal gene transfer and links to their environmental distribution. *Viruses*, 9(5):116, 2017.
- 7 Bernhard Haubold, Fabian Klötzl, and Peter Pfaffelhuber. andi: Fast and accurate estimation of evolutionary distances between closely related genomes. *Bioinformatics*, 31(8):1169–1175, 2014.
- 8 Guillaume Holley and Páll Melsted. Bifrost—Highly parallel construction and indexing of colored and compacted de Bruijn graphs. *BioRxiv*, page 695338, 2019.
- 9 Guillaume Holley, Roland Wittler, and Jens Stoye. Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage. *Algorithms for Molecular Biology*, 11(1):3, 2016.
- 10 Daniel H Huson, Tobias Klopper, and David Bryant. SplitsTree 4.0-computation of phylogenetic trees and networks. *Bioinformatics*, 14:68–73, 2008.
- 11 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, 2012.
- 12 Tobias H Klopper and Daniel H Huson. Drawing explicit phylogenetic networks and their integration into SplitsTree. *BMC Evolutionary Biology*, 8(1):22, 2008.
- 13 Chris-André Leimeister, Salma Sohrabi-Jahromi, and Burkhard Morgenstern. Fast and accurate phylogeny reconstruction using filtered spaced-word matches. *Bioinformatics*, 33(7):971–979, 2017.
- 14 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.
- 15 Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

- 16 B Jesse Shapiro, Ines Levade, Gabriela Kovacicova, Ronald K Taylor, and Salvador Almagro-Moreno. Origins of pandemic *Vibrio cholerae* from environmental gene pools. *Nature Microbiology*, 2(3):16240, 2017.
- 17 Jim Thurmond, Joshua L Goodman, Victor B Strelets, Helen Attrill, L Sian Gramates, Steven J Marygold, Beverley B Matthews, Gillian Millburn, Giulia Antonazzo, Vitor Trovisco, Thomas C Kaufman, Brian R Calvi, and the FlyBase Consortium. FlyBase 2.0: the next generation. *Nucleic Acids Research*, 47(D1):D759–D765, 2018.
- 18 Huiguang Yi and Li Jin. Co-phylog: an assembly-free phylogenomic approach for closely related organisms. *Nucleic Acids Research*, 41(7):e75–e75, 2013.
- 19 Xiaoyu Yu and Oleg N Reva. SWPhylo—a novel tool for phylogenomic inferences by comparison of oligonucleotide patterns and integration of genome-based and gene-based phylogenetic trees. *Evolutionary Bioinformatics*, 14:1176934318759299, 2018.
- 20 Zheming Zhou, Nabil-Fareed Alikhan, Martin J Sergeant, Nina Luhmann, Cátia Vaz, Alexandre P Francisco, João André Carriço, and Mark Achtman. GrapeTree: visualization of core genomic relationships among 100,000 bacterial pathogens. *Genome Research*, 28(9):1395–1404, 2018.
- 21 Zheming Zhou, Inge Lundstrøm, Alicia Tran-Dien, Sebastián Duchêne, Nabil-Fareed Alikhan, Martin J Sergeant, Gemma Langridge, Anna K Fotakis, Satheesh Nair, Hans K Stenøien, Stian S. Hamre, Sherwood Casjens, Axel Christophersen, Christopher Quince, Nicholas R. Thomson, François-Xavier Weill, Simon Y.W. Ho, M. Thomas P. Gilbert, and Mark Achtman. Pan-genome analysis of ancient and modern *Salmonella enterica* demonstrates genomic stability of the invasive para C lineage for millennia. *Current Biology*, 28(15):2420–2428, 2018.
- 22 Guanghong Zuo and Bailin Hao. CVTree3 web server for whole-genome-based and alignment-free prokaryotic phylogeny and taxonomy. *Genomics, Proteomics & Bioinformatics*, 13(5):321–331, 2015.

Quantified Uncertainty of Flexible Protein-Protein Docking Algorithms

Nathan L. Clement

Department of Computer Science, University of Texas at Austin, USA
nclement@cs.utexas.edu

Abstract

The strength or weakness of an algorithm is ultimately governed by the confidence of its result. When the domain of the problem is large (e.g. traversal of a high-dimensional space), an exact solution often cannot be obtained, so approximations must be made. These approximations often lead to a reported quantity of interest (QOI) which varies between runs, decreasing the confidence of any single run. When the algorithm further computes this QOI based on uncertain or noisy data, the variability (or lack of confidence) of the QOI increases. Unbounded, these two sources of uncertainty (algorithmic approximations and uncertainty in input data) can result in a reported statistic that has low correlation with ground truth.

In molecular biology applications, this is especially applicable, as the search space is generally large and observations are often noisy. This research applies *uncertainty quantification* techniques to the difficult protein-protein docking problem, where uncertainties arise from the explicit conversion from continuous to discrete space for protein representation (introducing some uncertainty in the input data), as well as discrete sampling of the conformations. It describes the variability that exists in existing software, and then provides a method for computing *probabilistic certificates* in the form of Chernoff-like bounds. Finally, this paper leverages these probabilistic certificates to accurately bound the uncertainty in docking from two docking algorithms, providing a QOI that is both robust and statistically meaningful.

2012 ACM Subject Classification Applied computing → Molecular structural biology; Mathematics of computing → Hypothesis testing and confidence interval computation; Computing methodologies → Uncertainty quantification

Keywords and phrases protein-protein docking, uncertainty quantification, protein flexibility, low-discrepancy sampling, high-dimensional sampling

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.3

Related Version A full version of the paper is available at <http://arxiv.org/abs/1906.10253>.

Acknowledgements I would like to thank all those who have supported and helped advise on this work, for their valuable feedback and suggestions for improvement.

1 Introduction

Predicting the bound conformation of two proteins (protein-protein docking) has many applications in medicine and biology [26, 19]. The simpler form of this problem is the so-called “bound-bound” case, where the 3-dimensional coordinates of the *in situ* protein complex is resolved (via e.g. X-ray crystallography, NMR, etc.), and atoms corresponding to individual proteins are then extracted from the complex. The more difficult version is the “unbound-unbound” case, where each protein in the pair is imaged in its separate native state, and the algorithm must predict the correct *in situ* bound complex [18]. Importantly, the final quantity of interest (QOI) in many cases is the change in binding free energy: protein complexes with a high change in free energy are more likely to be found as a bound complex, and are likely good targets for drug discovery pathways. The difficulty of the unbound-unbound case then arises from the inherent flexibility of proteins: large-scale movements



© Nathan L. Clement;

licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 3; pp. 3:1–3:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

may occur along the pathway from a closed conformation (unbound) to an open (bound) one, or visa versa. If docking is performed on only the unbound complexes, the final delta energy could be completely misleading. (To aid in discussion, here and through the paper we will refer to one protein, typically the bigger, as the *receptor*, and the other as the *ligand*.)

A subsequent difficulty of the unbound-unbound docking problem is that computational approaches must search two high-dimensional spaces. The first is that of possible protein structures, a naive description of which is $\mathbb{R}^{3m} \times \mathbb{R}^{3n}$, where m and n are the number of atoms in the ligand and receptor, respectively. The second is the space of possible docked conformations. In rigid-body docking (each protein is static), this is the 6-dimensional real space of 3 rotational + 3 translational degrees of freedom, $SE(3) = SO(3) \times \mathbb{R}^3$ [1, 25]. Without approximation, searching this high-dimensional space is computationally intractable. To achieve meaningful results, successful algorithms must employ some sort of simplification.

One of the biggest issues that arises from these simplifications is *uncertainty propagation*. A computational representation of a protein is, by nature, an approximation (discrete representation of a continuous space). Computing a simple statistic, or *quantity of interest* (QOI), on these representations is then by nature uncertain [27]. Algorithmic approximations (due to randomness or variations in the inputs) in one stage of a protein docking pipeline lead to uncertainty in the input for the next stage. If these uncertainties are not *quantified* at each stage, the uncertainties propagate to future levels of the pipeline, leading to a result or QOI that is unbounded, and may contain little valuable information.

This paper provides a framework for bounding the uncertainty of protein-protein docking. For a docking procedure where the QOI, $f(\mathbf{X})$, is some complicated function or optimization functional involving noisy data \mathbf{X} , we seek to provide a *probabilistic certificate* as a function of parameter t that the computed value $f(\mathbf{X})$ is not more than t away from the true value, with high probability. This certificate is expressed as a Chernoff-Hoeffding like bound [8]:

$$\Pr [|f(X) - E[f]| > t] \leq \epsilon, \quad (1)$$

where $E[f]$ is the expectation of f , computed over all permutations of X . The primary QOI we are interested in bounding is the change in Gibbs free energy, or ΔG , as this is the metric most useful for real-world experiments. However, we also consider the interface RMSD (iRMSD), which is defined as the RMSD between $C\alpha$ atoms on the interface of the bound pair.

Instead of providing a new docking algorithm as a solution to bounding the above certificate, this research instead considers the docking algorithm, $f(\mathbf{X})$, as a *black box*, exploring the landscape of possible structures, $X \in \mathbf{X}$, as inputs to f and computing the certificates from the output. This then provides a framework by which any two algorithms can be compared, and by which conclusive results can be reported.

In this work, we expand upon our previous research [27, 10] in the following manner. First, the model used in the previous research was simplistic, and, while useful for modeling small uncertainties, does not provide insight into uncertainty of large-scale protein movement. Second, we consider the impact of this conformational uncertainty to provide certificates for black-box docking functions. This second contribution can be used when trying to interpolate results of a given docking algorithm to biological equivalents.

The only known research that applies uncertainty quantification to protein-protein docking is a recent preprint by Cau and Shen [5]. The authors use Bayesian active learning to explore protein-protein docking samples using a black-box energy function. Once the energy landscape has been sufficiently sampled, they provide posterior distributions of the desired QOI, which enables computing confidence intervals for each model. The major differences between this

work and our own work is 1) the treatment of the entire *docking algorithm* as a black box (instead of just the energy function), and 2) the use of a hierarchical model convolved with a von Mises distribution to generate samples local to the unbound input.

The paper is organized as follows. First, we provide the theoretical and technical details of our approach, including probabilistic certificates through effective sampling, protein representation, sampling protocol, and benchmark dataset. Second, we show that the protein sampling protocol used improves upon the results of both rigid-body and flexible docking metrics, computing the probabilistic certificates for the change in Gibbs free energy for sets of docked proteins. Finally, we discuss the importance of these results both in terms of UQ for docking algorithms and biological relevance.

2 Materials and methods

2.1 Computing Chernoff-like bounds

Our primary motivation in this work is to compute a *probabilistic certificate* to bound the uncertainty in a computed statistics. We are most concerned with providing the Chernoff-Hoeffding like bound expressed in Equation 1, which provides a probabilistic guarantee for the moments of a QOI computed on noisy data.

We can provide a theoretic bound for the uncertainty by using the McDiarmid inequality, defined in [22] and extended to support summations of decaying kernels such as the Leonard-Jones potential in [27]. Let (X_i) be independent random variables with discrete space A_i , let $f : \prod_i A_i \rightarrow \mathbb{R}$, and let $|f(x_1, \dots, x_k, \dots, x_n) - f(x_1, \dots, x'_k, \dots, x_n)| \leq c_k$, or c_k is the degree of change influenced on f over all variations of x_k . Then, for any $t > 0$:

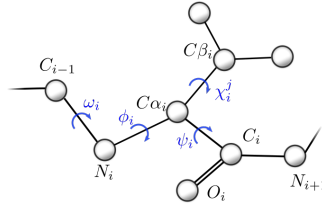
$$\Pr [|f(\mathbf{X}) - \mathbb{E}[f]| > t] < 2 \exp \left(-2t^2 / \sum_k c_k^2 \right).$$

Thus, to provide theoretic bounds, all that is required is to determine the value of c_k for each x_k . However, computing c_k analytically may be difficult, and even if it were possible, theoretical bounds these often overestimate the error. An alternate approach is then to empirically compute these certificates using quasi-Monte Carlo (QMC) methods [24, 16]. Assuming the distribution of (X_i) is known, we sample this space and evaluate f at each sample. This leads to an estimate of the distribution of f over the joint space of all A_i , which provides sufficient data to compute certificates on the uncertainty, as defined in Equation 1.

Correctness of this approach relies on the correctness of the QMC methods and the description of the joint sampling space. For this reason, we will spend the next section describing our protein representation and the corresponding sampling space. In the Results and discussion section we will show that our sampling space is accurate (e.g. a good representation of the distribution of (X_i)), and thus the provided certificates are also sound.

2.2 Protein representations

The base structure of a protein is a linear chain of *amino acids* (also called “residues”). Each amino acid consists of a set of atoms, and all the atoms connected by covalent bonds into a single 3-dimensional structure. Such atoms divide into two groups: *backbone* atoms: two carbons, one nitrogen, and one oxygen; and zero or more *side-chain* atoms. The carbon connecting the backbone to the side-chain atoms is called the $C\alpha$ atom, and the first side-chain carbon (if it exists) is called the $C\beta$ atom (see Figure 1). The *native representation* of a protein is thus a graph in 3-dimensional Cartesian space, where each node of the graph



■ **Figure 1** Torsion angles for a protein chain. The backbone atoms are labeled C_i , O_i , C_{α_i} , and N_i . For a constrained internal coordinate representation, only the ψ_i , ϕ_i , and potentially χ_i torsion angles are considered (ω_i is fixed at 180°).

represents atoms and edges represent bonds. The position of each node/atom is represented by a vector in \mathbb{R}^3 , requiring three parameters for each atom. If \hat{t} is the average number of side-chain atoms per residue for a given protein, then this representation requires a total of $n = 3(\hat{t} + 4)N$ parameters (3 degrees of freedom each for the \hat{t} side-chain and 4 backbone atoms), or degrees of freedom (DOFs), for a protein with N amino acids.

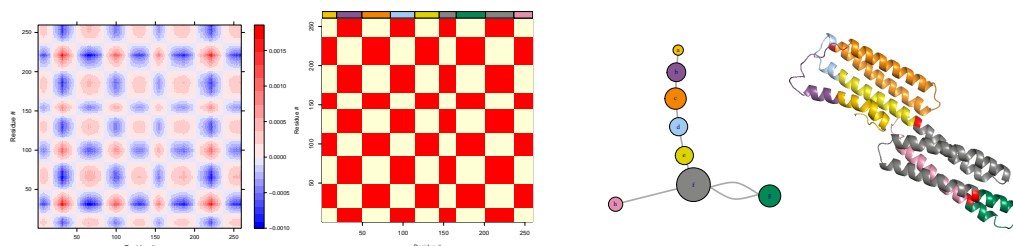
An alternate representation of proteins, employed by most sampling protocols (e.g. [23, 12] and others) is the *internal coordinates* representation. Under this representation, the position of each atom is only defined *in relation* to the atoms around it, and the free variables are bond angles, bond lengths, and torsion angles (the degree of “twist” defined by 2 planes or 4 atoms, see Figure 1). This does not immediately reduce the total degrees of freedom (since in general, each *atom* needs to be described by bond angles, bond lengths, and torsion angles); however, if small-scale atomic vibrations are ignored, then bond lengths and angles can be approximated as constant, leaving the only DOFs as the ψ , ϕ , and χ_i torsion angles (the ω torsion angle on the backbone is held at $\sim 180^\circ$ by the sp^2 partial double bond [4]). If \hat{k} is the average number of χ_i angles for a given residue (k varies from 0 to 5 in the standard 20 amino acids), then the number of DOFs for this representation for a protein with N amino acids is $m = (\hat{k} + 2)N$. Since in most cases $\hat{k} + 2 \ll 3(\hat{t} + 4)$, this *constrained* internal coordinate method allows for a lower-dimensional specification of the protein conformational space without a loss in representation [13].

2.3 Hierarchical domain decomposition and motion graph

Roughly speaking, proteins decompose into rigid and flexible parts. Rigid contiguous parts are called domains, which exhibit little movement in several conformations. In turn, flexible parts, also known as hinges, interconnect domains. These flexible parts show three types of motion: shearing or gliding (i.e. a lateral movement along domain interfaces), bending (i.e. an angular movement between axes of two connected domains), and twisting (i.e. a rotational movement around the longitudinal axis of a domain).

When representing large-scale protein motion, we are primarily interested in hinges, or flexible regions connecting large mostly-rigid bodies or domains. However, since there may be multiple levels of motion, we use a *hierarchical representation* of the constrained internal angles representation of the protein. The hierarchical representation is not a recursive subdivision of the protein, but rather a description of (possibly overlapping) protein motions. This allows us to represent motions at one level that consist of atoms from different domains in the previous level.

To obtain this hierarchical domain decomposition for a given protein, we model the protein as a Ca (one node per residue) GNM (Gaussian network model), and compute the NMA (normal modes analysis) decomposition of the protein (in this work, we use the



■ **Figure 2** The NMA decomposition of 1RKE receptor, for the second non-trivial mode. From left to right: the cross-correlation fluctuation matrix, $[F]_2$; the sign of entries of $[F]_2$, with short domains removed; the domain graph representation of the protein, where the size of each node represents the number of residues in that domain; and the domain graph representation mapped onto the 3d structure of the protein, colored according to domain with hinge residues colored red. Hinges that are also flexible connectors separate all domains but f (gray) and g (green), which are connected by segments (hinges) that would not form a cut in the domain graph representation.

implementation from the R Bio3D package [14]). Each of the k modes represents a separate direction of motion, from large-scale motions (the smallest eigenvalues) to the high-frequency vibrations of hydrogen atoms. Each mode corresponds to a different level in our hierarchical representation; that is, each hierarchical level corresponds to a distinct rigidity threshold.

Hinges are obtained in a similar fashion to that demonstrated by HingeProt[11], as follows. For each mode, i , we compute the mean square fluctuation matrix as follows:

$$[F]_i = \frac{3k_B T}{\gamma} \lambda_i^{-1} \mathbf{u}_i \mathbf{u}_i^T, \quad (2)$$

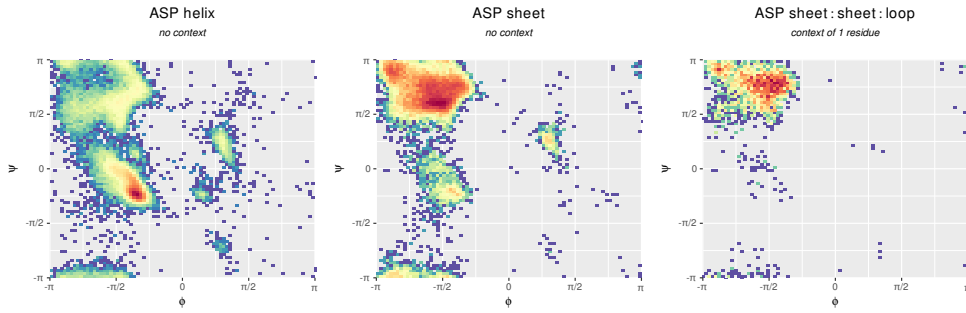
where λ_i and \mathbf{u}_i is the eigenvalue and eigenvector of mode i , and k_B , T , and γ are the Boltzmann constant, temperature, and uniform force constant, respectively. Regions of this matrix with the same sign form the rigid domains, and individual residues where the sign changes (from positive to negative) become hinges. For practical purposes, we collapse domains with only a small number of residues.

The final stage at a given level is to construct a domain graph representation, where nodes in the graph represent rigid domains and edges in the graph exist wherever two domains are in contact with each other, i.e. any atom from one rigid domain is within r_c of any atom from another domain (see Figure 2 for the decomposition a single level in the hierarchy). From this graph, we categorize each hinge as *flexible connector* if the removal of the hinge would form a cut of the domain graph representation, i.e. its removal would result in two disjoint subgraphs.

Once we have obtained the domain graph representation of the protein for each of the k NMA modes, we construct a multi-graph of the domain hierarchy for the entire protein [2]. At the top level of the hierarchy are the hinges and domains computed by the first non-trivial mode (i.e. with the smallest eigenvalue), representing more broad, global motions. The next level of the hierarchy is represented by the second smallest eigenvalue, and so on until all k modes have been used. We also assign a weight, w_k , to all hinges at level k of the hierarchy, arising from Equation 2:

$$w_k = 3k_B T \lambda_k^{-1}. \quad (3)$$

The final dimension of the product space of sampling is then $K_R + K_L$, where K_R (K_L) is the number of hinges from all k levels for the receptor (ligand), creating a product space of $SO(3)^{K_R+K_L}$ ($SO(3)$ is the special orthogonal group of rotations about a fixed axis). For



■ **Figure 3** Ramachandran distributions for aspartic acid under different parameterizations. Left: ss_i is a helix; middle: ss_i is a loop; right: ss_{i-1} , ss_i , and ss_{i+1} are respectively sheet-sheet-loop. Note that the distributions are more tightly clustered with the gain of additional context.

the dataset used in this paper, the value of $K_R + K_L$ range from 21 hinge residues (3FN1) to 70 (1BKD). It is well known that generating a small number of good (i.e. low discrepancy) samples is difficult in high dimensions, so to overcome this issue, we use the low-discrepancy sampling protocol developed by [3] when generating samples.

2.4 Sampling protocol and Ramachandran distributions of amino acids

Based on the hierarchical protein decomposition described above, we now describe how to obtain a set of representative samples of the protein. Even with a good low-discrepancy sampling, this high-dimensional product space still requires a large number of samples to completely cover the product space. However, most of these samples will lead to physically impossible protein structures: clashes between nearby atoms, steric strain, or even a protein that is no longer biologically active. We would like to reduce the sampling space for a given torsion angle from all of $SO(3)$ to only the relevant, low-energy regions.

To establish a set of generic neighbor-dependent Ramachandran probability distribution, we compute the torsion angles from a set of $\sim 15k$ high-quality, non-homologous protein structures obtained from the Pisces server [29]. From this set, we generate discrete probability distributions for each backbone torsion angle pair, conditioned on the amino acid type and secondary structure type of the previous and following residues. In other words,

$$Prob_N(\phi, \psi, i) = \Pr(\phi, \psi | ss_{i-1}, ss_i, ss_{i+1}, aa_i), \quad (4)$$

where ss_i and aa_i are the secondary structure and amino acid types of residue i , respectively, and ϕ and ψ are the backbone torsion angles (see Figure 1). Figure 3 shows the conditional distributions for aspartic acid.

To generate samples of a given protein, we would like to draw samples for each flexible residue from the neighbor-dependent Ramachandran distributions. However, we also recognize that the input protein has important structural elements that should be preserved. For this reason, we convolve the discrete Ramachandran distribution with a bivariate von Mises distribution (the two-dimensional variant of the approximately-Gaussian distribution on a unit circle, e.g. $[-\pi, \pi]^2$ [21]), centered at the given backbone torsion angle. The cosine variant of the bivariate von Mises distribution is given as follows:

$$\Pr(\phi, \psi) = Z_c(\kappa_1, \kappa_2, \kappa_3) \exp(\kappa_1 \cos(\phi - \mu) + \kappa_2 \cos(\psi - \nu) + \kappa_3 \cos(\phi - \mu - \psi + \nu)), \quad (5)$$

where μ and ν describe the mean for ϕ and ψ , κ_1 and κ_2 describe their concentration, and κ_3 describes their correlation. If κ_3 is zero and $\kappa_1 = \kappa_2 = \sigma$, then σ can be used to increase or

decrease the amount of bias the input structure has on the Ramachandran distributions. Lower values of σ (lower concentration) bias more toward the general Ramachandran distributions, while higher values of σ bias more towards the input protein structure.

The final probability of a given (ϕ, ψ) pair at position i , $Prob(i, \phi, \psi)$, is the convolution of the neighbor-dependent Ramachandran distribution with the specific von-Mises distribution:

$$Prob(\phi, \psi, i) \propto \Pr(\phi, \psi | ss_{i-1}, ss_i, ss_{i+1}, aa_i) * \exp \left[\sigma \cos(\phi - \hat{\phi}_i) + \sigma \cos(\psi - \hat{\psi}_i) \right], \quad (6)$$

where $\hat{\phi}_i$ and $\hat{\psi}_i$ are the values of ϕ and ψ for residue i in the input protein.

With the internal angles representation and hierarchical decomposition of the protein as input, we perform the following importance sampling protocol on each level, l :

1. For each hinge at level l , $h_j^{(l)}$, let i be the index of the residue corresponding to this hinge.
 - a. Generate the pair $(\hat{\phi}, \hat{\psi})$, drawn from the von Mises-convolved neighbor-dependent Ramachandran distributions
 - b. Let ρ_l be the probability of a given hinge residue changing, arising from w_l in Equation 3: $\rho_l = \min(1, w_l)$
 - c. If $h_j^{(l)}$ is a cut or no other non-cut hinges have been sampled at level l , set (ϕ_i, ψ_i) to $(\hat{\phi}, \hat{\psi})$ with probability ρ_l ; otherwise, keep the original (ϕ_i, ψ_i) pair
2. From the internal angle sample, generate the explicit structure in \mathbb{R}^3
3. Compute the number of clashes caused by hinges at level l , and accept the torsion angle changes for level l if the number of clashes are less than some parameter c . We define a *clash* as two atoms occupying the same space in \mathbb{R}^3 .

As we are most interested in modeling the large-scale uncertainty that arises from domain movements, we then find the optimal placement of side-chain atoms using SCWRL4 [17], followed by a brief energy minimization step with Amber16 [6] to remove any steric strain. Finally, we rank each sample by free energy, and keep only the samples with the lowest energy. These final two steps (minimization and ranking by energy) prevent us from using samples that are biologically irrelevant.

2.5 Benchmark dataset

In this research, we are interested in 1) modeling the uncertainty of a given protein-protein docking algorithm, but also 2) improving the existing docking results in the unbound-unbound case. The Zlab benchmark 5 [28] contains a set of proteins that have had the X-ray structure determined both in isolation and together, and consist of 254 protein pairs classified as either *difficult*, *medium difficulty*, or *rigid-body*, depending on the interface RMSD (iRMSD). The difficult class of proteins have an iRMSD of $> 2.2\text{\AA}$, which means there is typically some movement between bound and unbound conformations.

To select our set of input structures, we docked each protein classified as “difficult” in both the bound and unbound conformations with F2Dock [1, 9], a rigid-body docking algorithm. We selected those proteins that performed well in the bound structure but poorly with the unbound structure as candidates in our benchmark. The criteria we use for differentiating between success and failure is whether there exists a “hit” in the top 1000 reported poses. We define a “hit” as a bound pair with iRMSD within 5\AA of the actual bound conformation.

Since we are primarily interested in the single-body docking problem (and not the multi-body docking problem), we only kept the single-chain proteins for our experiment, which led to 10 single-chain proteins that perform well when using the bound conformation but

■ **Table 1** Protein structures used in dataset, labeled according to the ID from the ZLab benchmark 5 [28]. The top section contain those that performed well when bound, the bottom section containing those that did not.

ID	# Residues			iRMSD (Å)	Δ Energy (J)
	receptor	ligand	contact		
1ATN	372	258	36	42	131
1F6M	320	108	62	16	87
1FQ1	183	295	53	53	367
1BKD	439	166	97	20	425
1R8S	160	187	61	28	439
1RKE	262	176	68	52	524
1ZLI	306	74	77	13	212
2COL	292	122	92	17	366
2I9B	265	122	101	29	387
2J7P	292	265	80	20	370
2OT3	253	157	69	17	428
3FN1	160	90	38	14	315
1H1V	368	327	74	33	180
1Y64	411	357	66	39	192
3AAD	264	153	42	37	66

not when unbound. In addition, we also included the 3 single-chain proteins that performed poorly when both the bound and unbound conformation were used. Statistics on the size and free energy of each protein are given in Table 1.

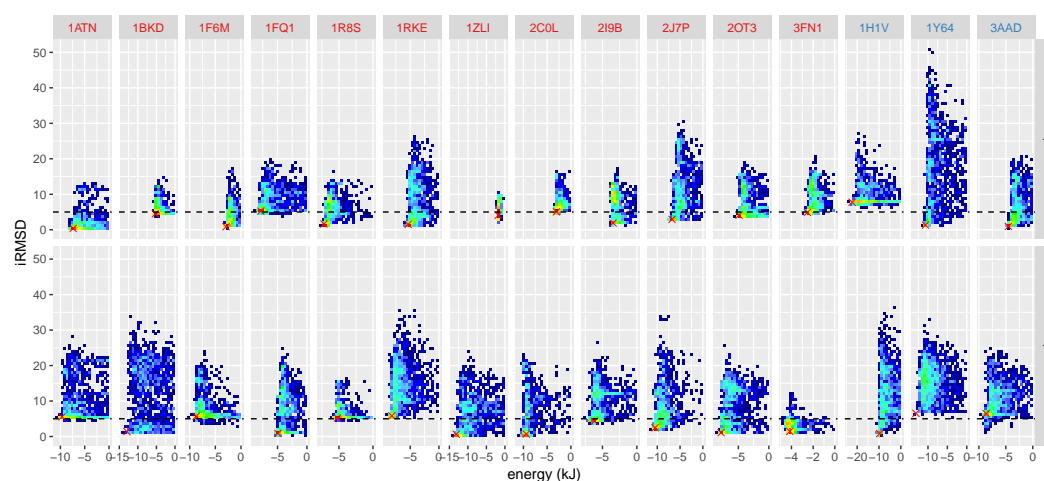
3 Results and discussion

3.1 Conformational sampling distributions

Our primary concern for generating a good set of samples is that the samples cover a good portion of the feasible set of the protein conformational space. We consider two metrics for measuring coverage: 1) the free energy of individual proteins, and 2) the iRMSD from the sample to the bound conformation. The first of these metrics is an unbiased measure of protein stability: if all samples have abnormally high energy, they are unlikely to be biologically feasible. However, it is possible that the bound conformation lies in an energy well made more available when in combination with the second protein. For this reason, we are not interested in only finding the energy minimum, but also the distance from the bound conformation. Figure 4 shows the energy vs iRMSD for 1000 samples of each protein.

3.2 Improvements in unbound-unbound docking

By generating a set of proteins that have a closer iRMSD to the bound conformation, we are able to improve on the blind unbound-unbound docking protocol, for both rigid-body and flexible docking algorithms. We compare the results for the bound and unbound case for F2Dock (a rigid-body docking algorithm) [1], Rosetta (a semi-flexible docking algorithm) [15, 7], and SwarmDock (a flexible docking algorithm) [20]. We perform bound-bound and unbound-unbound docking for each program, and compute the iRMSD on the reported poses. For F2Dock and Rosetta, the number of reported poses is variable, which we set to 1000. SwarmDock reports a fixed number of results, so this number varies from 465–548 poses.



■ **Figure 4** Plot of iRMSD (against the bound conformation) vs energy for samples generated from the unbound conformation. Proteins are separated by ligand (top) and receptor (bottom) to show the difference in individual protein movement. The black dashed line shows iRMSD= 5, or the value at which a match is considered a “hit,” and the red “X” marks the spot of the original unbound protein. For all proteins, there exist some samples that improve on both the iRMSD and energy; some of the proteins, such as 3AAD receptor and 1ZLI ligand, improve upon the iRMSD greatly. Strong convergence is shown by a funnel-shaped energy landscape, and is seen for many protein pairs. Protein labels are colored red (good when bound) and blue (bad when bound).

Since F2Dock and Rosetta both have command-line interfaces, we also perform docking 50 samples of the unbound conformation of each protein. The minimum iRMSD for each protein (bound, unbound, and samples for F2Dock and Rosetta) are found in Table 2.

The results from the iRMSD statistics suggest a few findings. First, the flexible algorithms (Rosetta and SwarmDock) are better at docking the bound-bound conformations than the rigid-body one (F2Dock). This is potentially due to the fact that clashes in side-chain atoms prevent the rigid body docking algorithm from correctly identifying the best conformation, but also could be due to the fact that each program uses a different energy function, and may be better tuned for these specific proteins in the flexible programs. The most important observation, however, is the huge difference in iRMSD between the bound and unbound pairs. This suggests that the input to the algorithm (e.g. unbound or bound) is an important characteristic of the docking result, and variations in input structure must be accounted for and described in the output QOI as empirical certificates. Finally, we also note that for each protein, using many different sampled proteins always improves the iRMSD of the docking result (sometimes drastically), suggesting that the sampling protocol is sound (leading to better results).

3.3 Probabilistic certificates from Quasi-Monti Carlo samples

To describe the uncertainty of the results of the docking algorithm, we compute the probabilistic certificates arising from the Chernoff-like bounds of the sampled algorithms, given in Equation 1. This provides a metric that can compare across proteins (for the same docking algorithm) and across docking algorithms (for the same protein, or over all proteins). We could provide probabilistic certificates for any QOI; however, we are primarily interested in bounding the binding free energy. If the reported free energy is tightly bound by a probabilistic certificate, we are more confident that we have identified the correct free energy.

■ **Table 2** Best RMSD (over top 1000 poses for F2Dock and Rosetta and all poses for SwarmDock) for proteins included in this dataset. A single asterisk marks proteins not in the bound form with at least one hit (iRMSD $< 5\text{\AA}$) in the top poses. F2Dock and Rosetta statistics for sampled proteins are also included. Note the great improvement on Rosetta docking when using the sampled proteins, and that the sampled proteins are always better than the unbound case. The large differences between bound and unbound input suggests the output QOI is highly dependent on the input to the model.

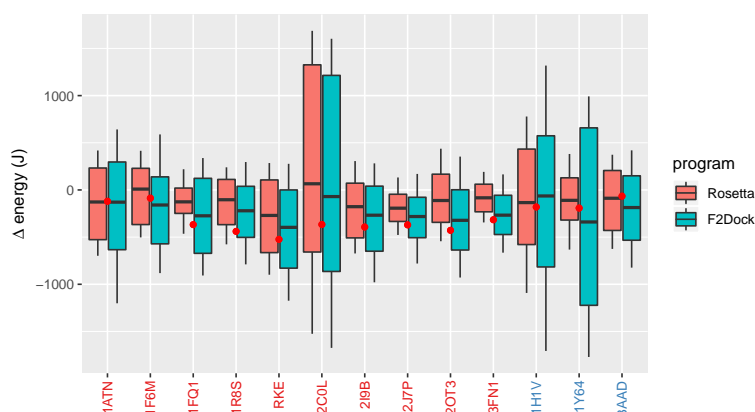
ID	F2Dock			Rosetta			SwarmDock	
	bound	unbound	sampled	bound	unbound	sampled	bound	unbound
1ATN	1.2	7.2	*4.6	0.08	8.6	6.8	0.98	*4.6
1BKD	1.3	8.7	*4.9	0.23	16.9	5.4	0.68	8.7
1F6M	1.2	8.1	*5.0	0.11	17.9	13.4	0.69	5.6
1FQ1	2.3	6.4	*4.4	0.48	15.0	8.2	3.55	5.6
1R8S	1.7	9.4	6.2	0.22	14.5	6.2	0.72	5.1
1RKE	0.8	7.1	5.8	0.15	15.1	12.8	0.65	5.4
1ZLI	0.6	10.0	*4.6	0.13	10.6	7.1	0.71	9.0
2C0L	0.5	*4.8	*4.0	0.30	12.2	8.2	0.75	*3.8
2I9B	1.1	8.4	*4.7	0.09	13.6	8.8	7.93	6.5
2J7P	1.4	7.2	*3.4	1.12	17.2	14.9	0.60	6.6
2OT3	1.1	5.1	*3.9	0.16	15.6	6.8	0.92	6.0
3FN1	1.0	5.5	*4.9	0.10	9.9	*4.8	0.53	*4.1
1H1V	8.8	11.0	8.0	0.27	18.8	13.5	0.68	9.1
1Y64	9.3	11.4	10.7	1.9	35.0	15.6	1.37	11.7
3AAD	7.0	8.2	5.3	0.39	22.0	9.2	2.36	7.1

Figure 5 shows a comparison of the certificate for ΔG of each protein (at $\text{Pr} = 0.9$, see Equation 1), and includes the true QOI (red dot), computed on the bound-bound conformation. For some of the proteins (e.g. 3FN1), the provided certificate is much tighter than others (e.g. 2C0L). This also allows us to directly compare the two different programs in terms of docking uncertainty. While the rigid F2Dock algorithm occasionally has higher bounds, with high probability the true statistic lies within the $\text{Pr} = 0.9$ certificate range. The Rosetta results usually contain the true QOI, but is not contained within the min/max range for 3FN1.

4 Conclusion

In this work, we provide a framework for providing probabilistic certificates on uncertainty in a docking algorithm. Fundamental to these certificates is the contribution of a low-discrepancy hierarchical sampling protocol that includes general amino acid information in the form of Ramachandran plots, but also structural information that is specific to the input protein in the form of a bivariate von Mises distribution. We show that the low-discrepancy samples generated by this protocol explore the energy landscape for the unbound protein, which includes samples closer to the bound conformation.

With these samples, we compare three different docking algorithms, ranging from rigid-body to completely flexible. We show that docking results vary substantially depending on the input protein structure – even for the flexible docking algorithms – further substantiating our claim that uncertainty quantification is essential to protein-protein docking.



■ **Figure 5** Probabilistic bounds compared to ground truth for values of ΔG . Protein labels are colored red (good when bound) and blue (bad when bound). The box shows the value of the certificate at $\text{Pr} = 0.9$ and the tails show the min/max values, and the red point shows the true statistic, computed on the bound-bound form of the protein.


Finally, we repeat the protein-protein docking experiments with different structures from our hierarchical sampling protocol and assess the variations in reported binding free energy. We compute a probabilistic certificate for the binding free energy, and compare the 90% confidence interval with the value computed on the bound complex. This provides a tool for comparing not only uncertainty across proteins, but also across docking algorithms.

References

- 1 C. Bajaj, R. Chowdhury, and V. Siddahanavalli. F2Dock: Fast Fourier Protein-protein Docking. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(1):45–58, 2011.
- 2 Chandrajit Bajaj, Rezaul Alam Chowdhury, and Vinay Siddavanahalli. F3Dock: A fast, flexible and Fourier based approach to protein-protein docking. *The University of Texas at Austin, ICES Report*, pages 08–01, 2008.
- 3 Chandrajit L. Bajaj, Abhishek Bhowmick, Eshan Chattopadhyay, and David Zuckerman. On Low Discrepancy Samplings in Product Spaces of Motion Groups. *arXiv e-prints*, page arXiv:1411.7753, November 2014. [arXiv:1411.7753](https://arxiv.org/abs/1411.7753).
- 4 Ian David Brown. Recent developments in the methods and applications of the bond valence model. *Chemical reviews*, 109(12):6858–6919, 2009.
- 5 Yue Cao and Yang Shen. Bayesian active learning for optimization and uncertainty quantification in protein docking. *arXiv preprint*, 2019. [arXiv:1902.00067](https://arxiv.org/abs/1902.00067).
- 6 D.A. Case, R.M. Betz, D.S. Cerutti, et al. *AMBER 2016*. University of California, San Francisco, 2016.
- 7 Sidhartha Chaudhury, Monica Berrondo, Brian D Weitzner, Pravin Muthu, Hannah Bergman, and Jeffrey J Gray. Benchmarking and analysis of protein docking performance in Rosetta v3. 2. *PLoS One*, 6(8):e22477, 2011.
- 8 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.
- 9 R. Chowdhury, D. Keidel, M. Moussalem, M. Rasheed, A. Olson, M. Sanner, and C. Bajaj. Protein-Protein Docking with F²Dock 2.0 and GB-rerank. *Biophys. J.*, 8(3):1–19, 2013.
- 10 Nathan Clement, Muhibur Rasheed, and Chandrajit Lal Bajaj. Viral capsid assembly: A quantified uncertainty approach. *Journal of Computational Biology*, 25(1):51–71, 2018.

- 11 Ugur Emekli, Dina Schneidman-Duhovny, Haim J Wolfson, Ruth Nussinov, and Turkan Haliloglu. HingeProt: automated prediction of hinges in protein structures. *Proteins: Structure, Function, and Bioinformatics*, 70(4):1219–1227, 2008.
- 12 Vamshi K. Gangupomu, Jeffrey R. Wagner, In-Hee Park, Abhinandan Jain, and Nagarajan Vaidehi. Mapping Conformational Dynamics of Proteins Using Torsional Dynamics Simulations. *Biophysical Journal*, 104(9):1999–2008, 2013.
- 13 Bryant Gipson, David Hsu, Lydia E Kavragi, and Jean-Claude Latombe. Computational models of protein kinematics and dynamics: Beyond simulation. *Annual Review of Analytical Chemistry*, 5:273–291, 2012.
- 14 Barry J Grant, Ana PC Rodrigues, Karim M ElSawy, J Andrew McCammon, and Leo SD Caves. Bio3d: an R package for the comparative analysis of protein structures. *Bioinformatics*, 22(21):2695–2696, 2006.
- 15 Jeffrey Gray, Stewart Moughon, Chu Wang, Ora Schueler-Furman, Brian Kuhlman, Carol Rohl, and David Baker. Protein–protein docking with simultaneous optimization of rigid-body displacement and side-chain conformations. *Journal of Molecular Biology*, 331(1):281–299, 2003.
- 16 Fred James, Jiri Hoogland, and Ronald Kleiss. Quasi-Monte Carlo, discrepancies and error estimates. *Methods*, page 9, 1996. [arXiv:physics/9611010](https://arxiv.org/abs/physics/9611010).
- 17 Georgii G Krivov, Maxim V Shapovalov, and Roland L Dunbrack. Improved prediction of protein side-chain conformations with SCWRL4. *Proteins: Structure, Function, and Bioinformatics*, 77(4):778–795, 2009.
- 18 Daisuke Kuroda and Jeffrey J Gray. Pushing the backbone in protein-protein docking. *Structure*, 24(10):1821–1829, 2016.
- 19 Loren M LaPointe, Keenan C Taylor, Sabareesh Subramaniam, Ambalika Khadria, Ivan Rayment, and Alessandro Senes. Structural organization of FtsB, a transmembrane protein of the bacterial divisome. *Biochemistry*, 52(15):2574–2585, 2013.
- 20 Xiaofan Li, Iain H Moal, and Paul A Bates. Detection and refinement of encounter complexes for protein–protein docking: taking account of macromolecular crowding. *Proteins: Structure, Function, and Bioinformatics*, 78(15):3189–3196, 2010.
- 21 Kanti Mardia, Charles Taylor, and Ganesh Subramaniam. Protein bioinformatics and mixtures of bivariate von Mises distributions for angular data. *Biometrics*, 63(2):505–512, 2007.
- 22 C McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, 141(141):148–188, 1989.
- 23 R. J. Milgram, G. Liu, and J. C. Latombe. On the structure of the inverse kinematics map of a fragment of protein backbone. *Journal of Computational Chemistry*, 29(1):50–68, 2008.
- 24 Harald Niederreiter. Quasi-Monte Carlo methods. *Encyclopedia of Quantitative Finance*, 24(1):55–61, 1990.
- 25 Dzmityr Padhorny, Andrey Kazennov, Brandon S Zerbe, Kathryn A Porter, Bing Xia, Scott E Mottarella, Yaroslav Kholodov, David W Ritchie, Sandor Vajda, and Dima Kozakov. Protein–protein docking by fast generalized Fourier transforms on 5D rotational manifolds. *Proceedings of the National Academy of Sciences*, 113(30):E4286–E4293, 2016.
- 26 Muhibur Rasheed, Radhakrishna Bettadapura, and Chandrajit Bajaj. Computational Refinement and Validation Protocol for Proteins with Large Variable Regions Applied to Model HIV Env Spike in CD4 and 17b Bound State. *Structure*, 23(6):1138–1149, 2015.
- 27 Muhibur Rasheed, Nathan Clement, Abhishek Bhowmick, and Chandrajit L Bajaj. Statistical framework for uncertainty quantification in computational molecular modeling. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017.
- 28 Thom Vreven, Iain H Moal, Anna Vangone, Brian G Pierce, et al. Updates to the Integrated Protein–Protein Interaction Benchmarks: Docking Benchmark Version 5 and Affinity Benchmark Version 2. *Journal of Molecular Biology*, 427(19):3031–3041, 2015.
- 29 Guoli Wang and Roland L Dunbrack Jr. PISCES: a protein sequence culling server. *Bioinformatics*, 19(12):1589–1591, 2003.

TRACTION: Fast Non-Parametric Improvement of Estimated Gene Trees

Sarah Christensen 

University of Illinois at Urbana-Champaign, USA
sac2@illinois.edu

Erin K. Molloy 

University of Illinois at Urbana-Champaign, USA
emolloy2@illinois.edu

Pranjal Vachaspati 

University of Illinois at Urbana-Champaign, USA
vachasp2@illinois.edu

Tandy Warnow¹ 

University of Illinois at Urbana-Champaign, USA
<http://tandy.cs.illinois.edu>
warnow@illinois.edu

Abstract

Gene tree correction aims to improve the accuracy of a gene tree by using computational techniques along with a reference tree (and in some cases available sequence data). It is an active area of research when dealing with gene tree heterogeneity due to duplication and loss (GDL). Here, we study the problem of gene tree correction where gene tree heterogeneity is instead due to incomplete lineage sorting (ILS, a common problem in eukaryotic phylogenetics) and horizontal gene transfer (HGT, a common problem in bacterial phylogenetics). We introduce TRACTION, a simple polynomial time method that provably finds an optimal solution to the RF-Optimal Tree Refinement and Completion Problem, which seeks a refinement and completion of an input tree t with respect to a given binary tree T so as to minimize the Robinson-Foulds (RF) distance. We present the results of an extensive simulation study evaluating TRACTION within gene tree correction pipelines on 68,000 estimated gene trees, using estimated species trees as reference trees. We explore accuracy under conditions with varying levels of gene tree heterogeneity due to ILS and HGT. We show that TRACTION matches or improves the accuracy of well-established methods from the GDL literature under conditions with HGT and ILS, and ties for best under the ILS-only conditions. Furthermore, TRACTION ties for fastest on these datasets. TRACTION is available at <https://github.com/pranjalv123/TRACTION-RF> and the study datasets are available at https://doi.org/10.13012/B2IDB-1747658_V1.

2012 ACM Subject Classification Applied computing → Molecular evolution; Applied computing → Population genetics

Keywords and phrases Gene tree correction, horizontal gene transfer, incomplete lineage sorting

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.4

Funding Sarah Christensen: Ira & Debra Cohen Fellowship

Erin K. Molloy: NSF Graduate Research Fellowship Grant Number DGE-1144245 and Ira & Debra Cohen Fellowship

Pranjal Vachaspati: NSF Graduate Research Fellowship Grant Number DGE-1144245

Tandy Warnow: NSF CCF-1535977

Acknowledgements We thank Mike Steel for encouragement and the members of the Warnow lab for valuable feedback. This study was performed on the Illinois Campus Cluster and Blue Waters, a computing resource that is operated and financially supported by UIUC in conjunction with the National Center for Supercomputing Applications.

¹ Corresponding author



1 Introduction

Reconstructing the evolutionary history of a gene is a core task in phylogenetics, and our ability to infer these evolutionary relationships accurately can have important implications for a variety of downstream analyses. For example, estimated gene trees are used in the inference of adaptation, evolutionary event detection (such as gene loss, gene duplication, and horizontal gene transfer), ortholog identification, analysis of functional trait evolution, and species tree estimation. However, unlike species tree estimation techniques that leverage information encoded across the entire genome, gene tree estimation based on a single locus may not contain enough signal to determine the correct gene tree topology with high confidence [27]. Indeed, many phylogenomic datasets have gene trees with average branch support well below 75%, which is a common lower bound for branches to be considered reliable. For example, the Avian Phylogenomic Project [17] reported average branch support values below 30%, and many other studies (surveyed in [25]) have had similar challenges. Estimating gene and species trees is further complicated by biological processes such as gene duplication/loss (GDL), incomplete lineage sorting (ILS), and horizontal gene transfer (HGT), that create heterogeneous tree topologies across the genome [21]. HGT has long been known to cause problems for bacterial phylogenetics, and ILS by itself has emerged as a major issue in phylogenomics, affecting most, if not all, genome-scale datasets [10].

Because gene trees often have low accuracy, a natural problem is to try to improve gene tree estimation using an estimated or known species tree. An approach from the gene duplication and loss literature is to modify estimated gene trees with respect to a reference species tree, which may either be an established tree from prior literature or an estimated species tree (e.g., based on an assembled multi-locus dataset). Some of these methods use the available sequence data as well as the estimated gene tree and species tree, and are referred to as “integrative methods”; examples include ProfileNJ [27], TreeFix [33], and TreeFix-DTL [2]. Other methods, called “gene tree correction methods”, use just the topologies of the gene tree and species tree, and are typically based on parametric models of gene evolution; Notung [6, 9] and ecceTERA [16] are two of the well known methods of this type. Integrative methods are generally expected to be more accurate than gene tree correction methods when gene tree heterogeneity is due to GDL, but as a result of using likelihood calculations they are also more computationally intensive. See [4, 26, 31, 18, 15, 16, 34] for an entry into the vast literature on this subject.

Here, we examine the problem of gene tree correction for the case where gene tree heterogeneity is due to ILS or HGT, and where each gene tree has at most one copy of each species. We present a new approach to gene tree correction that is based on a very simple *non-parametric* polynomial-time method, TRACTION, that is *agnostic* to the cause of gene tree heterogeneity. In addition to correcting gene trees, TRACTION is also capable of completing gene trees that do not contain all the species present in the reference species tree, a condition that may occur in a multi-locus study as a result of taxon sampling strategies or unavailable data (such as when not all genomes have been sequenced and assembled). The input to TRACTION is a pair (t, T) of unrooted phylogenetic trees. The leaf set of t is a subset of the leaf set of T , tree T is binary, and tree t will generally be non-binary. We seek a tree T' created by refining t and adding any missing leaves so that T' has the minimum Robinson-Foulds (RF) [28] distance to T . We call this optimization problem the “RF-Optimal Tree Refinement and Completion Problem” (RF-OTRC). We show that TRACTION finds an optimal solution to this problem in $O(n^{1.5} \log n)$ time, where n is the number of leaves in the species tree T . To use TRACTION for gene tree correction, we

assume we are given an estimated gene tree with branch support values and an estimated (or known) binary species tree, which may have additional species. The low support branches in the gene tree are collapsed, forming the (unresolved) tree t . TRACTION has two steps: first it refines the input gene tree t into a binary tree t' , and then it adds the missing species to t' . Although the algorithm is quite simple, the proof of correctness is non-trivial. We present the results of an extensive simulation study (on 68,000 gene trees, each with up to 51 species) in which gene tree heterogeneity is either due to only ILS or to both ILS and HGT. We explore TRACTION for gene tree correction with estimated species trees in comparison to Notung, ecceTERA, ProfileNJ, TreeFix, and TreeFix-DTL, evaluating the accuracy of the corrected gene trees by computing the RF distance to the true gene tree. Overall, while many methods (including TRACTION) tie for best on the ILS-only data, TRACTION dominates the other gene tree correction methods with respect to topological accuracy on the HGT+ILS data, and ties for fastest. Importantly, TRACTION provides good accuracy even when the estimated species tree is far from the true gene tree. The simplicity of the approach and its good accuracy under a range of different model conditions indicates that non-parametric approaches to gene tree correction may be promising, and encourages future research.

2 TRACTION

2.1 Terminology and Basics

Each edge e in an unrooted phylogenetic tree defines a *bipartition* π_e (also referred to as a *split*) on the leaves of the tree induced by the deletion of e (but not its endpoints). Each bipartition of the leaf set into two non-empty disjoint parts, A and B , is denoted by $A|B$. The set of bipartitions of a tree T is given by $C(T) = \{\pi_e : e \in E(T)\}$, where $E(T)$ is the edge set for T . Tree T' is a *refinement* of T if T can be obtained from T' by contracting a set of edges in $E(T')$. A tree T is *fully resolved* (i.e., binary) if there is no tree that refines T other than itself.

A set Y of bipartitions on some leaf set S is *compatible* if there exists an unrooted tree T leaf-labelled by S such that $Y \subseteq C(T)$; furthermore, when a set of bipartitions is compatible then there is a *unique* tree such that $Y = C(T)$. In addition, pairwise compatibility of a set of bipartitions ensures setwise compatibility [11, 12]. A bipartition π_e of a set S is said to be compatible with a tree T with leaf set S if and only if there is a tree T' such that $C(T') = C(T) \cup \{\pi\}$ (i.e., T' is a refinement of T that includes the bipartition π). Similarly, two trees on the same leaf set are said to be compatible if they share a common refinement; it then follows that two trees are compatible if and only if the union of their sets of bipartitions is compatible.

Given a phylogenetic tree T on taxon set S , T *restricted to* $R \subseteq S$ is the minimal subgraph of T connecting elements of R and suppressing nodes of degree two. We denote this as $T|_R$. If T and T' are two trees with R as the intersection of their leaf sets, their *shared edges* are edges whose bipartitions restricted to R are in the set $C(T|_R) \cap C(T'|_R)$. Correspondingly, their *unique edges* are edges whose bipartitions restricted to R are not in the set $C(T|_R) \cap C(T'|_R)$. See Figure 1 for a pictorial depiction of unique and shared edges.

The *Robinson-Foulds* (RF) distance [28] between two trees T and T' on the same set of leaves is the number of bipartitions present in only one tree; equivalently, the RF distance is equal to the total number of unique edges in both trees. The normalized RF distance is the RF distance divided by $2n - 6$, where n is the number of leaves in each tree; this produces a value between 0 and 1 since the two trees can only disagree with respect to internal edges and $n - 3$ is the maximum number of internal edges in an unrooted tree with n leaves.

2.2 RF-Optimal Tree Refinement and Completion (RF-OTRC) Problem

Input: An unrooted binary tree T on S and an unrooted tree t on $R \subseteq S$

Output: An unrooted binary tree T' on S with two key properties: (1) T' contains all the leaves of S and is compatible with t (i.e., $T'|_R$ is a refinement of t) and (2) T' minimizes the RF distance to T among all binary trees satisfying condition (1).

If the trees t and T have the same set of taxa, then the RF-OTRC problem becomes the RF-OTR (RF-Optimal Tree Refinement) problem, while if t is already binary but can be missing taxa, then the RF-OTRC problem becomes the RF-OTC (RF-Optimal Tree Completion) problem. OCTAL, presented in [7], solves the RF-OTC problem in $O(n^2)$ time, and an improved approach presented by Bansal [1] solves the RF-OTC problem in linear time. We refer to this faster approach as **Bansal's algorithm**. In this paper we present an algorithm that solves the RF-OTR problem exactly in polynomial time and show that the combination of this algorithm with Bansal's algorithm solves the RF-OTRC problem exactly in $O(n^{1.5} \log n)$ time, where T has n leaves. We refer to the two steps together as TRACTION (Tree Refinement And CompleTION).

2.3 TRACTION Algorithm

The input to TRACTION is a pair of unrooted trees (t, T) , where t is the estimated gene tree on set R of species and T is the binary reference tree on S , with $R \subseteq S$. We note that t may not be binary (e.g., if low support edges have already been collapsed) and may be missing species (i.e., $R \subset S$ is possible).

- Step 1: Refine t so as to produce a binary tree t^* with as many shared bipartitions with T as possible (using a polynomial time algorithm described below).
- Step 2: Add the missing species from T into t^* , minimizing the RF distance.

2.3.1 Step 1: Greedy Refinement of t

To compute t^* , we first refine t by adding all bipartitions from $T|_R$ that are compatible with t ; this produces a unique tree t' . If t' is not fully resolved, then there are multiple optimal solutions to the RF-OTR problem, as we will later prove. The algorithm selects one of these optimal solutions as follows. First, we add edges from t that were previously collapsed (if such edges are available). Next, we randomly refine the tree until we obtain a fully resolved refinement, t^* . Note that if t' is not binary, then t^* is not unique. We now show that the first step of TRACTION solves the RF-OTR problem.

► **Theorem 1.** *Let T be an unrooted binary tree on leaf set S , and let t be an unrooted tree on leaf set $R \subseteq S$. A fully resolved (i.e. binary) refinement of t minimizes the RF distance to $T|_R$ if and only if it includes all compatible bipartitions from $T|_R$.*

Proof. Let C_0 denote the set of bipartitions in $T|_R$ that are compatible with t . By the theoretical properties of compatible bipartitions (Section 2.1), this means the set $C_0 \cup C(t)$ is a compatible set of bipartitions that define a unique tree t' where $C(t') = C_0 \cup C(t)$. We now prove that for any binary tree B that refines t , B minimizes the RF distance to $T|_R$ if and only if B refines t' .

Consider a sequence of trees $t = t_0, t_1, t_2, \dots, t_k$, each on leaf set R , where t_i is obtained from t_{i-1} by adding one edge to t_{i-1} , and thus adds one bipartition to $C(t_{i-1})$. Let $\delta_i = RF(t_i, T|_R) - RF(t_{i-1}, T|_R)$, so that δ_i indicates the change in RF distance produced by adding a specific edge to t_{i-1} to get t_i . Hence,

$$RF(t_i, T|_R) = RF(t_0, T|_R) + \sum_{j \leq i} \delta_j.$$

Since T is fully resolved, a new bipartition π_i added to $C(t_{i-1})$ is in $C(T|_R)$ if and only if $\pi_i \in C_0$. If this is the case, then the RF distance will decrease by one (i.e., $\delta_i = -1$). Otherwise, $\pi_i \notin C_0$, and the RF distance to $T|_R$ will increase by one (i.e., $\delta_i = 1$).

Now suppose B is a binary refinement of t . We split the bipartitions in $C(B) \setminus C(t)$ into two sets, X and Y , where X are bipartitions in C_0 and Y are bipartitions not in C_0 . By the argument just provided, it follows that $RF(B, T|_R) = RF(t, T|_R) - |X| + |Y|$. Note that $|X \cup Y|$ must be the same for all binary refinements of t , because all binary refinements of t have the same number of edges. Thus, $RF(B, T|_R)$ is minimized when $|X|$ is maximized, so B minimizes the RF distance to $T|_R$ if and only if $C(B)$ contains all the bipartitions in C_0 . In other words, $RF(B, T|_R)$ is minimized if and only if B refines t' . ◀

► **Corollary 2.** *TRACTION finds an optimal solution to the RF-OTR problem.*

Proof. Given input gene tree t and reference tree T , both on the same leaf set, TRACTION produces a tree t'' that refines t and contains every bipartition in T that is compatible with t ; hence by Theorem 1, TRACTION solves the RF-OTR problem. ◀

2.3.2 Step 2: Adding in missing species

The second step of TRACTION can be performed using OCTAL or Bansal's algorithm, each of which finds an optimal solution to the RF-OTC problem in polynomial time. More generally, we show that any method that optimally solves the RF-OTC problem can be used as an intermediate step to solve the RF-OTRC problem.

To prove this, we first restate several prior theoretical results. In the proof of correctness for OCTAL, [7] showed the minimum achievable RF distance between T and T' is given by:

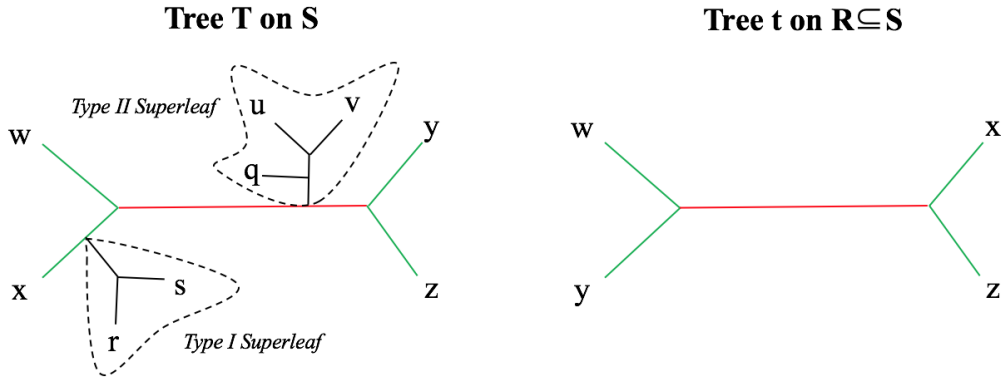
$$RF(T, T') = RF(T|_R, t) + 2m \tag{1}$$

where m is the number of Type II superleaves in T relative to t , which we now define.

► **Definition 3.** *Let T be a binary tree on leaf set S and t be a tree on leaf set $R \subseteq S$. The superleaves of T with respect to t are defined as follows (see Figure 1). The set of edges in T that are on a path between two leaves in R define the backbone; when this backbone is removed, the remainder of T breaks into pieces. The components of this graph that contain vertices from $S \setminus R$ are the superleaves. Each superleaf is rooted at the node that was incident to one of the edges in the backbone, and is one of two types:*

- *Type I superleaves: the edge e in the backbone to which the superleaf was attached is a shared edge in $T|_R$ and t*
- *Type II superleaves: the edge e in the backbone to which the superleaf was attached is a unique edge in $T|_R$ and t*

► **Theorem 4** (Restatement of Theorem in [7]). *Given unrooted binary trees t and T with the leaf set of t a subset of the leaf set S of T , $OCTAL(T, t)$ solves the RF-OTC problem and runs in $O(n^2)$ time, where T has n leaves.*



■ **Figure 1** Type I and Type II superleaves of a tree T with respect to t . Edges in the backbone (defined to be the edges on paths between nodes in the common leaf set) are colored green for shared and red for unique; all other edges are colored black. The deletion of the backbone edges in T defines the superleaves; one is a Type I superleaf because it is attached to a shared (green) edge and the other is a Type II superleaf because it is attached to a unique (red) edge. This figure is taken from [7].

2.4 Proof of correctness for TRACTION

► **Theorem 5.** *Let T be an unrooted binary tree on leaf set S with $|S| = n$, and let t be an unrooted tree on leaf set $R \subseteq S$. TRACTION returns a binary unrooted tree T' on leaf set S such that $\text{RF}(T', T)$ is minimized subject to $T'|_R$ refines t . The first stage (refining t) takes $O(|S| + |R|^{1.5} \log(|R|))$ time. Hence, TRACTION runs in $O(n^{1.5} \log n)$ time if used with Bansal's algorithm and $O(n^2)$ time if used with OCTAL.*

Proof. By construction TRACTION outputs a tree T' that, when restricted to the leaf set of t , is a refinement of t . Hence, it is clear that $T'|_R$ refines t . Now, it is only necessary to prove that $\text{RF}(T', T)$ is minimized by TRACTION. Since the intermediate tree t^* produced in the first step of TRACTION is binary, Theorem 4 gives that TRACTION using OCTAL (or any method exactly solving the RF-OTC problem) will add leaves to t^* in such a way as to minimize the RF distance to T .

As given in Equation 1, the optimal RF distance between T' and T is the sum of two terms: 1) $\text{RF}(t^*, T|_R)$ and 2) the number of Type II superleaves in T relative to t^* . Theorem 1 shows that TRACTION produces a refinement t^* that minimizes the first term. All that remains to be shown is that t^* is a binary refinement of t minimizing the number of Type II superleaves in T relative to t^* .

Consider a superleaf X in T with respect to t . If t were already binary, then every superleaf X is either a Type I or a Type II superleaf. Also, note that every Type I superleaf in T with respect to t will be a Type I superleaf for any refinement of t . However, when t is not binary, it is possible for a superleaf X in T to be a Type II superleaf with respect to t but a Type I superleaf with respect to a refinement of t . This happens when the refinement of t introduces a new shared edge with T to which the superleaf X is attached in T . Notice that since the set of all possible shared edges that could be created by refining t is compatible, any refinement that maximizes the number of shared edges with T also minimizes the number of Type II superleaves. Theorem 1 shows that TRACTION produces such a refinement t^* of t . Thus, TRACTION finds a binary unrooted tree T' on leaf set S such that $\text{RF}(T', T)$ is minimized subject to the requirement that $T'|_R$ refine t .

We now analyze the running time, focusing on the first stage. Constructing $T|_R$ takes $O(|S|)$ time. Checking compatibility of a single bipartition with a tree on K leaves, and then adding the bipartition to the tree if compatible, can be performed in only $O(|K|^{0.5} \log(|K|))$ after a fast preprocessing step (see Lemmas 3 and 4 from [14]). Hence, determining the set of edges of $T|_R$ that are compatible with t takes only $O(|S| + |R|^{1.5} \log(|R|))$ time. Therefore, the first stage of TRACTION takes $O(|S| + |R|^{1.5} \log(|R|))$ time. Hence, if used with OCTAL, TRACTION takes $O(|S|^2)$ time and if used with Bansal’s algorithm TRACTION takes $O(|S|^{1.5} \log |S|)$ time. ◀

3 Evaluation

Overview. We evaluated TRACTION in comparison to Notung, ecceTERA, profileNJ, TreeFix, and TreeFix-DTL on estimated gene trees under two different model conditions (ILS-only and ILS+HGT), using estimated species trees. In total, we analyzed 68,000 genes: 8,000 with 26 species under ILS-only models and 60,000 with 51 species under ILS+HGT models. All estimated gene trees we correct in these experiments were complete (i.e., have all the species). The motivation for this is two-fold. First, the methods we benchmark against do not provide an option for completing gene trees with missing data. This is understandable since these methods were developed for GDL, where missing species in a gene tree are interpreted as true loss events rather than incomplete sampling. Second, an experimental evaluation of OCTAL, the algorithm that performs the completion step of TRACTION, was previously performed in [7].

Datasets. We briefly describe the datasets used in this study (all are from prior studies [8, 7] and available online). The datasets include single copy genes with 26 or 51 species (each with a known outgroup), and were generated under model conditions where true gene trees and true species trees differed due to only ILS (two levels of ILS) or to both ILS and HGT (one ILS level but two levels of HGT). The true gene tree heterogeneity (**GT-HET**, the topological distance between true species trees and true gene trees) ranged from 10% (for the ILS-only condition with moderate ILS) to as high as 68% (for the ILS+HGT condition with high HGT). Each model condition has 200 genes, and we explore multiple replicates per model condition with different sequence lengths per gene. See Table 1 for details.

Estimated gene trees and estimated reference species trees. For each gene, we used RAxML v8.2.11 [29] under the GTRGAMMA model to produce maximum likelihood gene trees, with branch support computed using bootstrapping. Because sequence lengths varied, this produced estimated gene trees with different levels of gene tree estimation error (**GTEE**) (defined to be the average RF distance between the true gene tree and the estimated gene tree), ranging from 32% to 63% as defined by the missing branch rate (see Table 1). We estimated a species tree using ASTRID v1.4 [32] on the RAxML gene trees. Since we know the true outgroup for all species trees and gene trees, we used this in our performance study to root the input species tree and the estimated gene trees, given as input to all methods.

The gene trees given as input to the different methods were computed as follows. Each edge in each RAxML gene tree we computed was annotated with its bootstrap branch support, and we identified all the branches with bootstrap support less than 75% (the standard threshold for “low support”). Low support branches were collapsed in the gene trees before being given to TRACTION, Notung, and ProfileNJ. When we ran ecceTERA, we gave the binary gene trees with the threshold value (i.e., minimum required bootstrap support

■ **Table 1** Empirical properties of the simulated datasets used in this study: GT-HET (gene tree heterogeneity, the average normalized RF distance between true gene trees and true species trees); GTEE (average gene tree estimation error); and the average distance of the ASTRID reference tree to the true gene trees. The publications from which the simulated datasets are taken are also indicated. In total we analyzed 68,000 genes with varying levels and causes of true gene tree heterogeneity (to the true species tree) and gene tree estimation error. The ILS-only conditions each had 20 replicates, and the ILS+HGT conditions each had 50 replicates.

	GT-HET	GTEE	Distance ASTRID to true gene trees
			ILS-only, Low ILS, 26 species [7]
# sites varies	0.10	0.32	0.08
			ILS-only, High ILS, 26 species [7]
# sites varies	0.36	0.40	0.33
			ILS+HGT, Moderate HGT (m5), 51 species [8]
100 sites	0.54	0.63	0.55
250 sites	0.54	0.47	0.55
500 sites	0.54	0.47	0.54
			ILS+HGT, High HGT (m6), 51 species [8]
100 sites	0.68	0.62	0.68
250 sites	0.68	0.46	0.68
500 sites	0.68	0.38	0.68

value) of 75%; ecceTERA then collapses all branches that have support less than 75%, and explores the set of refinements. Thus, the protocol we followed ensures that ecceTERA, ProfileNJ, Notung, and TRACTION all used the same set of collapsed gene trees. TreeFix and Treefix-DTL used the uncollapsed gene trees.

Gene tree correction and integrative methods. The RAxML gene trees were corrected using TRACTION v1.0, Notung v2.9, ecceTERA v1.2.4, ProfileNJ (as retrieved from GitHub after the March 20, 2018 commit with ID 560b8b2) [27], TreeFix v1.1.10 (for the ILS-only datasets), and TreeFix-DTL v1.0.2 (for the HGT+ILS datasets), each with a species tree estimated using ASTRID v1.4 [32] as the reference tree rooted at the outgroup. The integrative methods (TreeFix, TreeFix-DTL, and ProfileNJ) also required additional input data related to the gene alignments, which we detail in the appendix. All estimated gene trees are complete (i.e., there are no missing taxa), so TRACTION only refines the estimated gene tree and does not add any taxa.

Evaluation criteria. We use RF tree error (the standard criterion in performance studies evaluating phylogeny estimation methods) to quantify error in estimated and corrected gene trees as compared to the known true gene tree (as defined in the simulation protocol) and the impact of TRACTION, Notung, ecceTERA, and TreeFix-DTL, on these errors. Note that although we use the RF distance within the OTR optimization criterion, there it refers to the distance between the corrected gene tree and the reference tree (which is an *estimated species tree*); in contrast, when we use the RF error rate in the evaluation criterion, in that context it refers to the distance between the corrected gene tree and the true *gene tree*. Since

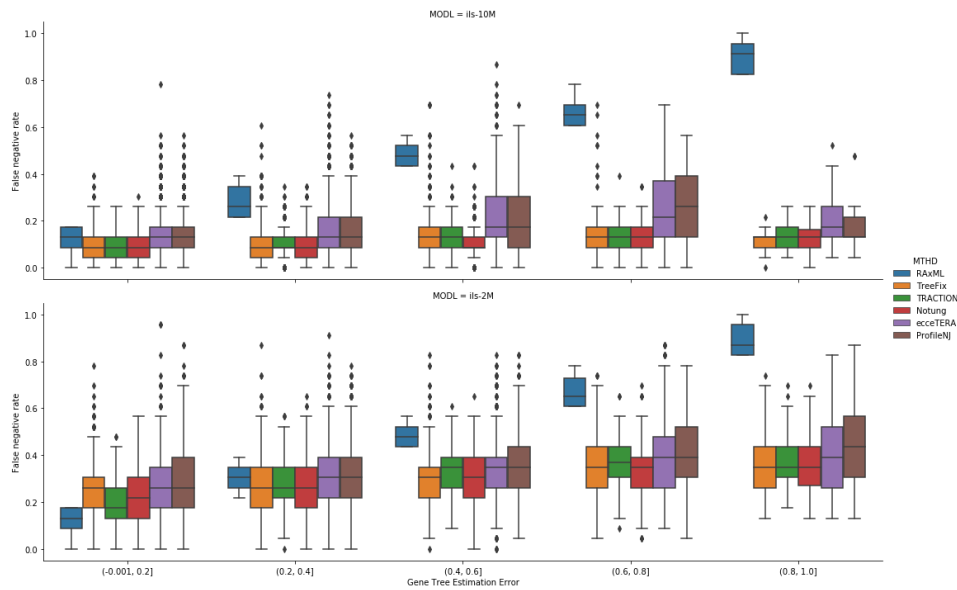


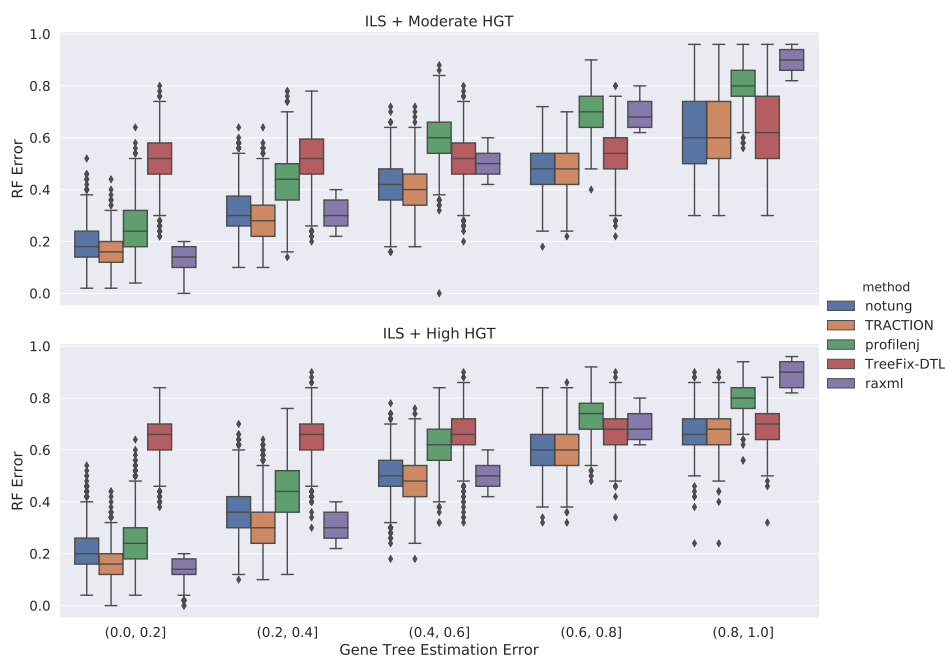
Figure 2 Comparison of methods on the ILS-only datasets with respect to average RF error rate, as a function of GTEE. Results are only shown for those datasets on which all methods complete. Each model condition (characterized by ILS level) has 20 replicate datasets, each with 200 genes.

the reference trees used in our experiments are typically very topologically different from the true gene tree (8% RF distance for the moderate ILS condition, 33% for the high ILS condition, 54% to 68% for the ILS+HGT conditions, see Table 1), optimizing the RF distance to the reference tree is quite different from optimizing the RF distance to the true gene tree.

Experiments. We performed two main experiments: one in which we explored performance on ILS-only datasets and the other in which we explored performance on datasets with HGT and ILS. In each case, we directly explored how the GTEE level impacted absolute and relative accuracy of the gene tree correction methods. We also indirectly explored how GT-HET affects relative and absolute accuracy. The heterogeneity is higher on the HGT+ILS datasets than on the ILS-only datasets, as HGT adds to the heterogeneity between gene trees and species trees (see Table 1).

4 Results and Discussion

Experiment 1: Comparison of methods on ILS-only datasets. Not all methods completed on all datasets: ecceTERA failed to complete on 67 datasets, profileNJ failed to complete on two datasets, and all other methods completed on all datasets. Results shown in Figure 2 are restricted to those datasets on which all methods completed; see the Appendix for additional results. For the moderate ILS condition (Figure 2 (top)), all methods are able to improve on RAxML, and the degree of improvement increases with GTEE. For the high ILS condition (Figure 2 (bottom)), methods improve on RAxML only when GTEE is at least 20%. Thus, GTEE and ILS level both impact whether methods can improve on RAxML. Furthermore, the methods group into two sets: TRACTION, Notung, and TreeFix performing very similarly and ProfileNJ and ecceTERA having somewhat higher error.



■ **Figure 3** Comparison of methods on ILS+HGT datasets with respect to average RF error rate (max is 1.0), as a function of GTEE (gene tree estimation error of the RAxML gene trees); ecceTERA failed on many datasets (with increasing failure rate as GTEE increases), and so those results are not shown. Results shown here are for only those datasets on which all methods completed.

Experiment 2: Comparison of methods on the HGT+ILS datasets. The HGT+ILS datasets have heterogeneity due to both HGT and ILS, with the degree of HGT varying from moderate (m5) to high (m6). On these data, ecceTERA failed on 1,318 datasets with the failure rates increasing as the gene tree estimation error of the initial RAxML gene tree (GTEE) increased: it failed 0% of the time when GTEE is less than 40%, 0.4% of the time when GTEE is 40-60%, 23.6% of the time when GTEE is 60-80%, and 90.8% of the time when GTEE is at least 80%. Because of the high failure rate, we do not report results for ecceTERA under these conditions. Figure 3 shows the impact of the remaining methods on RAxML gene trees as a function of GTEE. The relative performance between the remaining methods show that TRACTION and Notung are more accurate than profileNJ and TreeFix-DTL, with the gap between the two groups increasing with GTEE. We also see that TRACTION has an advantage over Notung for the low GTEE condition and matches the accuracy on the higher GTEE conditions. Finally, for the lowest GTEE bin, no method improves the RAxML gene tree, some methods make the gene trees much less accurate (e.g., profileNJ), and only TRACTION maintains the accuracy of the RAxML gene tree. Overall, on the HGT+ILS datasets, TRACTION consistently does well and has a clear advantage over the other methods in terms of accuracy.

Running Times. We selected a random sample of the 51-taxon HGT+ILS datasets to evaluate the running time (see Table 2). From fastest to slowest, the average running times were 0.5 seconds for TRACTION, 0.8 seconds for Notung, 1.7 seconds for ProfileNJ, 3.8 seconds for TreeFix-DTL, and 29 seconds for ecceTERA. Furthermore, most of the methods had consistent running times from one gene to another, but ecceTERA had high variability,

depending on the size of the largest polytomy. When the largest polytomy was relatively small, it completed in just a few seconds, but it took close to a minute when the largest polytomy had a size at the limit of 12. Results on other HGT+ILS replicates and model conditions gave very similar results.

■ **Table 2** Total time (in seconds) for each method to correct 50 gene trees with 51 species on one replicate (label 01) of the HGT+ILS dataset with moderate HGT and sequences of length 100bp.

Method	Time (s)
EcceTERA	1470
NOTUNG	43
TRACTION	30
ProfileNJ	87
TreeFix-DTL	188

Overall comments. This study shows that the better methods for gene tree correction (TRACTION, Notung, and TreeFix) reliably produce more accurate gene trees than the initial RAXML gene trees for the ILS-only conditions (except for cases where the initial gene tree is already very accurate), and the improvement can be very large when the initial gene trees are poorly estimated. However, the impact of gene tree correction is reduced for the HGT+ILS scenarios, where improvement over the initial gene tree is only obtained when GTEE is fairly high. As shown in Table 1, the average normalized RF distance between the reference tree (ASTRID) and the true gene trees is never more than 33% for the ILS-only scenarios but very high for the HGT+ILS scenarios (54% for moderate HGT and 68% for high HGT). Since the reference tree is the basis for the correction of the gene trees, it is not surprising that improvements in accuracy are difficult to obtain for the HGT+ILS scenario. On the other hand, given the large distance between the reference tree and the true gene tree, the fact that improvements are obtained for several methods (TRACTION, Notung, and TreeFix-DTL) is encouraging.

5 Conclusions

We presented TRACTION, a method that solves the RF-OTRC problem exactly in $O(n^{1.5} \log n)$ time, where n is the number of species in the species tree; the algorithm itself is very simple, but the proof of optimality is non-trivial. TRACTION performs well, matching or improving on the accuracy of competing methods on the ILS-only datasets and dominating the other methods on the HGT+ILS datasets. Furthermore, although all the methods are reasonably fast on these datasets, TRACTION is the fastest on the 51-taxon gene trees, with Notung a close second.

The observation that TRACTION performs as well (or better) than the competing methods (ecceTERA, ProfileNJ, Notung, TreeFix, and TreeFix-DTL) is encouraging. However, the competing methods are all based on stochastic models of gene evolution that are inherently derived from gene duplication and loss (GDL) scenarios (and in one case also allowing for HGT), and thus it is not surprising that GDL-based methods do not provide the best accuracy on the ILS-only or HGT+ILS model conditions we explore (and to our knowledge, all the current methods for gene tree correction are based on GDL models). Yet, TRACTION has good accuracy under a wide range of scenarios. We conjecture that this generally good performance is the result of its non-parametric criterion which can help it to be robust to model mis-specification (of which gene tree estimation error is one aspect).

This study showed that when the reference tree is very far from the true gene trees (as for our HGT+ILS data), gene tree correction typically fail to improve the initial gene tree (and even here, some methods can make the gene tree worse). This brings into question why the species tree (whether true or estimated) is used as a reference tree. We note that while the GDL-based methods may benefit from the use of a species tree as a reference tree (since the correction is based on GDL scenarios), this type of reference tree may not be optimal for TRACTION, which has no such dependency. Thus, part of our future work will be to explore techniques (such as statistical binning [3, 23]) that might enable the estimation of a better reference tree for TRACTION in the context of a multi-locus phylogenomic analysis.

This study suggests several other directions for future research. The GDL-based methods have variants that may enable them to provide better accuracy (e.g., alternative techniques for rooting the gene trees, selecting duploss parameter values, etc.), and future work should explore these variants. Most gene tree correction methods have been developed specifically to address the case where genes have multiple copies of species as a result of gene duplication events. TRACTION is currently restricted to gene trees with at most one copy of each species. In future work, we will explore extensions of TRACTION to handle multi-copy genes by using a generalization of the RF distance, such as proposed in [5]. In particular, one could construct the extended species tree to use as a reference along with a full differentiation of the gene tree as described in [5]. Recent work has shown how Notung could be extended to address HGT [19]; a comparison between TRACTION and a new version of Notung that addresses HGT will need to be made when Notung is modified to handle HGT (that capability is not yet available). Finally, the effect of gene tree correction on downstream analyses should be evaluated carefully.

References

- 1 M.S. Bansal. Linear-Time Algorithms for Some Phylogenetic Tree Completion Problems Under Robinson-Foulds Distance. In M. Blanchette and A. Ouangraoua, editors, *Comparative Genomics*. RECOMB-CG 2018. Lecture Notes in Computer Science, vol 11183. Springer, 2018.
- 2 M.S. Bansal, Y.-C. Wu, E.J. Alm, and M. Kellis. Improved gene tree error correction in the presence of horizontal gene transfer. *Bioinformatics*, 31(8):1211–1218, 2015.
- 3 Md Shamsuzzoha Bayzid, Siavash Mirarab, Bastien Boussau, and Tandy Warnow. Weighted statistical binning: enabling statistically consistent genome-scale phylogenetic analyses. *PLoS One*, 10(6):e0129183, 2015.
- 4 R. Chaudhary, J.G. Burleigh, and O. Eulenstein. Efficient error correction algorithms for gene tree reconciliation based on duplication, duplication and loss, and deep coalescence. *BMC Bioinformatics*, 13(10):S11, 2012.
- 5 Ruchi Chaudhary, John Gordon Burleigh, and David Fernández-Baca. Inferring species trees from incongruent multi-copy gene trees using the Robinson-Foulds distance. *Algorithms for Molecular Biology*, 8(1):28, 2013.
- 6 K. Chen, D. Durand, and M. Farach-Colton. NOTUNG: a program for dating gene duplications and optimizing gene family trees. *Journal of Computational Biology*, 7(3-4):429–447, 2000.
- 7 S. Christensen, E.K. Molloy, P. Vachaspati, and T. Warnow. OCTAL: optimal completion of gene trees in polynomial time. *Algorithms for Molecular Biology*, 13(1):6, March 2018.
- 8 R. Davidson, P. Vachaspati, S. Mirarab, and T. Warnow. Phylogenomic species tree estimation in the presence of incomplete lineage sorting and horizontal gene transfer. *BMC Genomics*, 16:S1, 2015.
- 9 D. Durand, B.V. Halldórsson, and B. Vernot. A hybrid micro-macroevolutionary approach to gene tree reconstruction. *Journal of Computational Biology*, 13(2):320–335, 2006.
- 10 S.V. Edwards. Is a new and general theory of molecular systematics emerging? *Evolution*, 63(1):1–19, 2009.
- 11 George F Estabrook, CS Johnson Jr, and Fred R Mc Morris. An idealized concept of the true cladistic character. *Mathematical Biosciences*, 23(3-4):263–272, 1975.

- 12 George F Estabrook, CS Johnson Jr, and FR McMorris. A mathematical foundation for the analysis of cladistic character compatibility. *Mathematical Biosciences*, 29(1-2):181–187, 1976.
- 13 W. Fletcher and Z. Yang. INDELible: A Flexible Simulator of Biological Sequence Evolution. *Molecular Biology and Evolution*, 26(8):1879–1888, 2009. 10.1093/molbev/msp098.
- 14 P. Gawrychowski, G.M. Landau, W.-K. Sung, and O. Weimann. A Faster Construction of Phylogenetic Consensus Trees. *arXiv preprint*, 2017. [arXiv:1705.10548](https://arxiv.org/abs/1705.10548).
- 15 E. Jacox, C. Chauve, G.J. Szöllősi, Y. Ponty, and C. Scornavacca. ecceTERA: comprehensive gene tree-species tree reconciliation using parsimony. *Bioinf.*, 32(13):2056–2058, 2016.
- 16 E. Jacox, M. Weller, E. Tannier, and C. Scornavacca. Resolution and reconciliation of non-binary gene trees with transfers, duplications and losses. *Bioinf.*, 33(7):980–987, 2017.
- 17 E.D. Jarvis, S. Mirarab, A.J. Aberer, B. Li, P. Houde, C. Li, S. Ho, B.C. Faircloth, B. Nabholz, J.T. Howard, et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, 346(6215):1320–1331, 2014.
- 18 M. Lafond, C. Chauve, N. El-Mabrouk, and A. Ouangraoua. Gene tree construction and correction using supertree and reconciliation. *IEEE/ACM Trans Comp Biol Bioinf (TCBB)*, 15(5):1560–1570, 2018.
- 19 H. Lai, M. Stolzer, and D. Durand. Fast Heuristics for Resolving Weakly Supported Branches Using Duplication, Transfers, and Losses. In J. Meidanis and L. Nakhleh, editors, *Comparative Genomics*, pages 298–320, Cham, 2017. Springer International Publishing.
- 20 Vincent Lefort, Richard Desper, and Olivier Gascuel. FastME 2.0: A Comprehensive, Accurate, and Fast Distance-Based Phylogeny Inference Program. *Molecular Biology and Evolution*, 32(10):2798–2800, 2015.
- 21 W. Maddison. Gene Trees in Species Trees. *Systematic Biology*, 46(3):523–536, 1997.
- 22 D. Mallo, L. Martins, and D. Posada. SimPhy: phylogenomic simulation of gene, locus, and species trees. *Systematic Biology*, 65(2):334–344, 2016.
- 23 S. Mirarab, M.S. Bayzid, B. Boussau, and T. Warnow. Statistical binning enables an accurate coalescent-based estimation of the avian tree. *Science*, 346(6215), 2014. [doi:10.1126/science.1250463](https://doi.org/10.1126/science.1250463).
- 24 S. Mirarab and T. Warnow. ASTRAL-II: Coalescent-based Species Tree Estimation with Many Hundreds of Taxa and Thousands of Genes. *Bioinformatics*, 31(12):i44, 2015.
- 25 E. Molloy and T. Warnow. To include or not to include: The impact of gene filtering on species tree estimation methods. *Systematic Biology*, 67(2):285–303, 2018.
- 26 T.H. Nguyen, V. Ranwez, S. Pointet, A.-M. Chifolleau, J.-P. Doyon, and V. Berry. Reconciliation and local gene tree rearrangement can be of mutual profit. *Algorithms for Molecular Biology*, 8(1):1, 2013.
- 27 E. Noutahi, M. Semeria, M. Lafond, J. Seguin, B. Boussau, L. Guéguen, N. El-Mabrouk, and E. Tannier. Efficient gene tree correction guided by genome evolution. *PLoS One*, 11(8):e0159559, 2016.
- 28 D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical biosciences*, 53(1-2):131–147, 1981.
- 29 A. Stamatakis. RAxML Version 8: A tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies. *Bioinformatics*, 30(9), 2014. 10.1093/bioinformatics/btu033.
- 30 J. Sukumaran and M.T. Holder. DendroPy: a Python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571, 2010. 10.1093/bioinformatics/btq228.
- 31 G.J. Szöllősi, W. Rosikiewicz, B. Boussau, E. Tannier, and V. Daubin. Efficient exploration of the space of reconciled gene trees. *Systematic Biology*, 62(6):901–912, 2013.
- 32 P. Vachaspati and T. Warnow. ASTRID: Accurate Species Trees from Internode Distances. *BMC Genomics*, 16(10):S3, 2015. 10.1186/1471-2164-16-S10-S3.
- 33 Y.-C. Wu, M.D. Rasmussen, M.S. Bansal, and M. Kellis. TreeFix: statistically informed gene tree error correction using species trees. *Systematic Biology*, 62(1):110–120, 2012.
- 34 Y. Zheng and L. Zhang. Reconciliation With Nonbinary Gene Trees Revisited. *Journal of the ACM (JACM)*, 64(4):24, 2017.

A Details about the Experimental Design

The datasets we used are from prior publications (which should be consulted for full details). The information for Steps 1-2 provided here describe the high-level process used to generate these datasets for those studies, Step 3 describes the process used to estimate gene trees, and Steps 4-6 describe the high-level process used to correct gene trees.

- **Step 1:** A model species tree and model gene trees (evolved within the model species tree) were generated using SimPhy [22]. This produced a set of gene trees that could differ from the species tree due to ILS alone or due to both ILS and HGT. Importantly, SimPhy modifies gene tree branch lengths to deviate from a strict molecular clock.
- **Step 2:** Molecular sequences were generated using INDELible [13] by evolving sequences down each true gene tree under the GTR+GAMMA model of sequence evolution without insertions or deletions. GTR+GAMMA model parameters and sequence lengths were drawn from distributions as described in [24]. Because the sequence length parameter was drawn from a distribution, sequences had different lengths. In some experiments, sequence lengths were truncated to 100, 250, or 500 sites before estimating gene trees in order to vary the degree of gene tree estimation error.
- **Step 3:** Binary gene trees were estimated on each gene sequence alignment using RAxML, a maximum likelihood (ML) heuristic, under the GTR+GAMMA model, with all numeric parameters estimated directly from the data. Branch support for each internal branch in the best ML tree was computed using the RAxML rapid bootstrapping procedure [29] with 100 bootstrap replicates for the ILS-only datasets and 50 bootstrap replicates for the ILS+HGT datasets.
- **Step 4:** Species trees were estimated on each multi-gene dataset using ASTRID [32] on the estimated (best ML) gene trees from Step 3.
- **Step 5:** For each estimated (best ML) gene tree, all edges with branch support below 75% were collapsed to produce a set of “collapsed gene trees”.
- **Step 6:** Estimated gene trees were corrected using the ASTRID tree from Step 4 as the reference tree. The input reference tree was rooted at the outgroup for all gene tree correction methods except for TRACTION. TRACTION and Notung were given the collapsed gene trees as input, whereas TreeFix, TreeFix-DTL, and ecceTERA were given the best ML gene trees (without any edges collapsed) as input. To run ecceTERA, we specified the threshold value (i.e., minimum required bootstrap support value), with the default setting of 75%; ecceTERA then collapses all branches that have support less than that value and exhaustively evaluates all refinements. However, when the collapsed gene trees have polytomies of degree greater than 12, then ecceTERA lowers the threshold until all polytomies have degree at most 12. Finally, Notung, ProfileNJ, and ecceTERA required that the input gene trees be rooted, so we rooted these input gene trees at the outgroup.

B Commands

In the following commands, “resolved gene trees” refers to the gene trees estimated using RAxML, “unresolved gene trees” refers to these estimated gene trees with branches having bootstrap support less than 75% collapsed, and “reference species tree” refers to the species tree estimated using ASTRID. Rooted means the input tree was rooted at the outgroup.

RAxML v8.2.11 was run as

```
raxml -f a -m GTRGAMMA -p 12345 -x 12345 -N <# bootstrap replicates> \
-s <alignment file> -n <output name>
```

ASTRID v1.4 was run as

```
ASTRID -i <resolved gene trees> -o <output>
```

Notung v2.9 was run as

```
java -jar Notung-2.9.jar --resolve -s <rooted reference species tree> \
  -g <rooted unresolved gene tree> --speciestag postfix \
  --treeoutput newick --nolosses
```

TRACTION v1.0 was run as

```
traction.py --refine -r -s 12345 -b <unrooted reference species tree> \
  -u <unrooted resolved gene trees> -i <unrooted unresolved gene trees> \
  -o <output>
```

ecceTERA v1.2.4 was run as

```
eccetera resolve.trees=0 \
  collapse.mode=1 \
  collapse.threshold=75 \
  dated=0 print.newick=true \
  species.file=<rooted reference species tree> \
  gene.file=<rooted resolved gene tree>
```

Since ecceTera enters an infinite loop on some gene trees, the “timeout” command was used to kill ecceTera if it took more than five minutes on a single gene tree.

ProfileNJ requires a distance matrix; to compute distance matrices (with K2P-corrected distances) for ProfileNJ, FastME v2.1.6.1 [20] was run as

```
fastme -i <input gene alignment> -O <output distance matrix> -dK
```

ProfileNJ was run as

```
profileNJ \
  -g <rooted unresolved gene tree> \
  -s <rooted reference species tree> \
  -d <distance matrix> \
  -o <output> \
  -S <name map> \
  -r none \
  -c nj \
  --slimit 1 \
  --plimit 1 \
  --firstbest \
  --cost 1 0.99999
```

TreeFix v1.1.10 was run on the ILS-only datasets as

```
treefix -s <rooted reference species tree> \
  -S <name map> \
  -A <alignment file extension> \
  -o <old tree file extension> \
  -n <new tree file extension> \
  <resolved gene tree>
```

TreeFix-DTL v1.0.2 was run on the HGT+ILS datasets as

```
treefixDTL -s <rooted reference species tree> \
  -S <map file> \
  -A <alignment file extension> \
  -o <old gene tree file extension> \
  -n <new gene tree file extension> \
  <resolved gene tree>
```

Normalized Robinson-Foulds distances were computed using Dendropy v4.2.0 [30] as

```
n1 = len(t1.internal_edges(exclude_seed_edge=True))
n2 = len(t2.internal_edges(exclude_seed_edge=True))
[fp, fn] = false_positives_and_negatives(t1, t2)
rf = float(fp + fn) / (n1 + n2)
```

B.1 Details about failures

No method other than ecceTERA and profileNJ failed on any datasets.

B.1.1 ecceTERA failures

In our analyses for the ILS-only datasets, ecceTERA failed on 10/4000 genes (moderate ILS) and 57/4000 genes (high ILS). In our analyses for the ILS+HGT datasets, ecceTERA failed on 744/7500 genes (moderate HGT) and 574/7500 genes (high HGT). Notably, the number of datasets that ecceTERA failed on increased with gene trees estimation error; for example, for datasets with ILS and HGT, ecceTERA completed on 100% of datasets with GTEE in (0.0, 0.4], 99.6% of datasets with GTEE in (0.4, 0.6], 76.4% of datasets with GTEE in (0.6, 0.8], and 9.2% of datasets with GTEE in (0.8, 1.0].

B.1.2 profileNJ failures

ProfileNJ computes distances to construct the corrected gene tree; when used with FastME, it failed on 2/4000 genes for the ILS-only condition (moderate ILS).

Better Practical Algorithms for rSPR Distance and Hybridization Number

Kohei Yamada

Division of Information System Design, Tokyo Denki University, Japan
19rmd38@ms.dendai.ac.jp

Zhi-Zhong Chen

Division of Information System Design, Tokyo Denki University, Japan
zzchen@mail.dendai.ac.jp

Lusheng Wang

Department of Computer Science, City University of Hong Kong, China
cswangl@cityu.edu.hk

Abstract

The problem of computing the rSPR distance of two phylogenetic trees (denoted by RDC) is NP-hard and so is the problem of computing the hybridization number of two phylogenetic trees (denoted by HNC). Since they are important problems in phylogenetics, they have been studied extensively in the literature. Indeed, quite a number of exact or approximation algorithms have been designed and implemented for them. In this paper, we design and implement one exact algorithm for HNC and several approximation algorithms for RDC and HNC. Our experimental results show that the resulting exact program is much faster (namely, more than **80 times faster** for the easiest dataset used in the experiments) than the previous best and its superiority in speed becomes even more significant for more difficult instances. Moreover, the resulting approximation programs output much better results than the previous bests; indeed, the outputs are always nearly optimal and often optimal. Of particular interest is the usage of the Monte Carlo tree search (MCTS) method in the design of our approximation algorithms. Our experimental results show that with MCTS, we can often solve HNC exactly within short time.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases phylogenetic tree, fixed-parameter algorithms, approximation algorithms, Monte Carlo tree search

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.5

Supplement Material Our programs are available at <http://rnc.r.dendai.ac.jp/rsprHN.html>.

1 Introduction

Constructing the evolutionary history of a set of species is an important problem in the study of biological evolution. Phylogenetic trees are used in biology to represent the ancestral history of a collection of existing species. This is appropriate for many groups of species. However, due to reticulation events such as hybridization, recombination, and lateral gene transfer, there are certain groups for which the ancestral history cannot be represented by a tree. For this kind of groups of species, it is more appropriate to represent their ancestral history by rooted acyclic digraphs, where vertices of in-degree at least two represent reticulation events.

More specifically, by looking at two different segments of sequences or two different sets of genes of a set of extant species, we may obtain two different phylogenetic trees T_1 and T_2 of the same extant species with high confidence. Given T_1 and T_2 , we want to construct a reticulate network N with the smallest number of reticulation events needed to explain the



© Kohei Yamada, Zhi-Zhong Chen, and Lusheng Wang;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 5; pp. 5:1–5:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

evolution of the species under consideration [13]. Roughly speaking, N is the smallest rooted acyclic digraph such that each of T_1 and T_2 is homeomorphic to a subgraph of N . The number of vertices of in-degree larger than 1 in N is called the *hybridization number* of T_1 and T_2 . The problem of computing the hybridization number of two given phylogenetic trees, denoted by HNC, is NP-hard [12, 4]. For this reason, quite a number of approximation algorithms and fixed-parameter algorithms have been designed and implemented for HNC [1, 9, 11, 14, 16, 20]. To the best of our knowledge, the software in [10] for solving HNC exactly achieves the previously best speed in practice, while the software in [16] for solving HNC approximately achieves the previously best approximation-ratio in practice.

A problem closely related to HNC is the problem of computing the rSPR distance of two given phylogenetic trees T_1 and T_2 of the same extant species. The *rSPR distance* between T_1 and T_2 can be defined as the minimum number of edges that should be deleted from each of T_1 and T_2 in order to transform them into homeomorphic rooted forests F_1 and F_2 . The problem of computing the rSPR distance of two trees, denoted by RDC, is NP-hard [4, 12]. This has motivated researchers to design and implement either exact or approximation algorithms for RDC [4, 6, 7, 8, 10, 12, 15, 17, 20, 19]. To the best of our knowledge, the software in [7] for solving RDC exactly (respectively, approximately) achieves the previously best speed (respectively, approximation-ratio) in practice (<http://rnc.r.dendai.ac.jp/rspr.html>).

In this paper, we first improve Chen *et al.*'s exact algorithm [10] for HNC. Since rSPR distance is a lower bound on hybridization number, the main idea is to use the lower bound on rSPR distance outputted by Chen *et al.*'s algorithm [7] to cut unnecessary branches of the search tree. Another main idea is to arrange the child recursive calls of each recursive call carefully. Our experimental results show that the resulting algorithm can be implemented into a program that runs more than **80 times faster** than Chen *et al.*'s UltraNet [10] for the easiest dataset used in the experiments. Moreover, its superiority in speed becomes even more significant for more difficult instances.

We then present a new approximation algorithm for RDC. Although this algorithm does not necessarily always output a better result than the algorithm in [7], we can obtain a new algorithm which calls the two algorithms and outputs the better result returned by them. Our experimental results show that the resulting algorithm can be implemented into a program that often outputs better results than Chen *et al.*'s program [7]. We further propose to use the so-called Monte Carlo tree search (MCTS) method [5] to improve *any* approximation algorithm A for RDC. In our application of MCTS, instead of performing a number of random play-outs¹ in the simulation phase of each round, we make a single call of A and then in the backpropagation phase, use its returned result to update information in the sequence of nodes selected for this round. Our experimental results show that the MCTS-based algorithm (denoted by MCTS_ A) can be implemented into a program that outputs much better and indeed always nearly optimal results. It is worth mentioning that even if A has a theoretical performance guarantee (say, a 2-approximation ratio), we are unable to prove any theoretical performance guarantee for MCTS_ A . Nevertheless, our experimental results show that MCTS_ A almost always outputs better results than A . Of course, if we want MCTS_ A to have the same theoretical performance guarantee as A , then we can modify MCTS_ A so that it calls A and uses A 's output instead if its own is worse.

¹ Roughly speaking, a random play-out here means computing an approximate solution by always making a random choice whenever a decision has to be made among a set of multiple choices.

We further combine our MCTS-based approximation algorithm for RDC with the integer-linear programming (ILP) approach in [16] to obtain a new approximation algorithm for HNC. Our experimental results show that the new algorithm can be implemented into a program that outputs much better (indeed always nearly optimal and often optimal) results than the previous best in [16].

Our programs are available at <http://rnc.r.dendai.ac.jp/rsprHN.html>.

2 Preliminaries

Throughout this paper, a *rooted forest* always means a directed acyclic graph in which every vertex has in-degree at most 1 and out-degree at most 2.

Let F be a rooted forest. The *roots* (respectively, *leaves*) of F are those vertices whose in-degrees (respectively, out-degrees) are 0. The *size* of F , denoted by $|F|$, is the number of roots in F minus 1. A vertex v of F is *unifurcate* if it has only one child in F . If a root v of F is unifurcate, then *contracting v in F* is the operation that modifies F by deleting v . If a non-root vertex v of F is unifurcate, then *contracting v in F* is the operation that modifies F by first adding an edge from the parent of v to the child of v and then deleting v .

For a vertex v of F , the *subtree of F rooted at v* , denoted by F^v , is the subgraph of F whose vertices are the descendants of v in F and whose edges are those edges connecting two descendants of v in F . If v is a root of F , then F^v is a *component tree* of F ; otherwise, it is a *pendant subtree* of F . For convenience, we view each vertex u of F as an ancestor and descendant of u itself. A vertex u is *lower than* another vertex $v \neq u$ in F if u is a descendant of v in F . The *lowest common ancestor* (LCA) of a set U of vertices in F , denoted by $\ell_F(U)$, is the lowest vertex v in F such that for every vertex $u \in U$, v is an ancestor of u in F . Note that if no component tree of F contains all vertices of U , then $\ell_F(U)$ does not exist. Two vertices u and v of F are *incomparable* if neither of them is an ancestor of the other in F . For two incomparable vertices u and v appearing in the same component tree of F , $D_F(u, v)$ denotes the set of all vertices w such that w is not a vertex of the (undirected) path $P_{u,v}$ between u and v in F but is the child of some inner vertex of $P_{u,v}$. For each pendant subtree T of F that has at least two leaves, the leaf-label set of T is a *cluster* of F .

A *rooted binary forest* is a rooted forest in which the out-degree of every non-leaf vertex is 2. Let F be a rooted binary forest. F is a *rooted binary tree* if it has only one root. If v is a non-root vertex of F with parent p , then *detaching F^v* is the operation that modifies F by first deleting the edge (p, v) and then contracting p . A *detaching operation* on F is the operation of detaching a pendant subtree of F .

2.1 Phylogenetic Trees and Forests

Let X be a set of existing species. A *phylogenetic tree* on X is a rooted binary tree whose leaf set is X . A *phylogenetic forest* is the graph obtained by applying a sequence of zero or more detaching operations on a phylogenetic tree. In other words, a phylogenetic forest is a graph whose connected components are phylogenetic trees on different sets of species.

An *FF pair* is a pair (F_1, F_2) , where F_1 and F_2 are two phylogenetic forests on the same set X of species. A *TT pair* is an FF pair (F_1, F_2) such that both F_1 and F_2 are trees.

For an FF pair (F_1, F_2) , the labeled leaves of F_1 naturally one-to-one correspond to those of F_2 (i.e., each pair of corresponding leaves have the same label). We extend the correspondence between the labeled leaves of F_1 and F_2 to (some of) their ancestors recursively as follows. Suppose that v_1 is a non-leaf vertex of F_1 , v_2 is a non-leaf vertex of F_2 , and the children of v_1 in F_1 one-to-one correspond to those of v_2 in F_2 . Then, v_1 corresponds to v_2 .

An FF pair (F_1, F_2) is *proper* if every root of F_1 , except at most one, corresponds to a root in F_2 . Obviously, a TT pair is also a proper FF pair. *Simplifying* a proper FF pair (F_1, F_2) is to repeatedly perform the following operation on F_1 and F_2 until it is not applicable:

- If some non-root vertex v of F_1 corresponds to a root of F_2 , then modify F_1 by detaching F_1^v .

Obviously, if (F_1, F_2) is proper, then it remains proper after being simplified.

Throughout the remainder of this paper, *an FF pair always means a proper FF pair*. A *sub-FF pair* of a TT pair (T_1, T_2) is an FF pair (F_1, F_2) such that for each $i \in \{1, 2\}$, F_i is obtained from T_i by performing zero or more detaching operations.

For an FF-pair (F_1, F_2) , if a vertex v_1 of F_1 and a vertex v_2 of F_2 correspond to each other, then both v_1 and v_2 are *matched* and they are the *mates* of each other. For brevity, if v is a matched vertex of F_i for some $i \in \{1, 2\}$, then we will also use v to denote its mate in F_{3-i} .

2.2 Agreement Forests

Let (F_1, F_2) be a sub-FF pair of a TT pair (T_1, T_2) . If we can apply a sequence of detaching operations on each of F_1 and F_2 so that they become the same forest F , then we refer to F as an *agreement forest* (AF) of (F_1, F_2) . A *maximum agreement forest* (MAF) of (F_1, F_2) is an AF of (F_1, F_2) whose size is minimized over all AFs of (F_1, F_2) . The size of an MAF of (F_1, F_2) minus $|F_2|$ is called the *rSPR distance* of (F_1, F_2) , and is denoted by $d(F_1, F_2)$. Obviously, an AF F of (F_1, F_2) is also an AF of (T_1, T_2) . The following lemma is shown in [7].

► **Lemma 1** ([7]). *Given an FF-pair (F_1, F_2) , we can compute a lower bound b_ℓ and an upper bound b_u on the rSPR distance of (F_1, F_2) in cubic time such that $b_u \leq 2b_\ell$.*

Suppose that F is an AF of (T_1, T_2) . For each $i \in \{1, 2\}$, we can define an injective mapping f_i from the vertex set of F to that of T_i as follows. For each leaf u of F , $f_i(u)$ is the leaf of T_i with the same label. For each non-leaf vertex u of F , $f_i(u)$ is $\ell_{T_i}(f_i(v_1), \dots, f_i(v_q))$, where v_1, \dots, v_q are the leaf descendants of u in F . For convenience, we hereafter also use each vertex u of F to denote $f_i(u)$ in T_i . We can now use F , T_1 , and T_2 to construct a directed graph G_F as follows:

- The vertices of G_F are the roots of F .
- For every two roots r_1 and r_2 of F , there is an edge from r_1 to r_2 in G_F if and only if r_1 is an ancestor of r_2 in T_1 or T_2 .

We refer to G_F as the *decision graph associated with F* . If G_F is acyclic, then F is an *acyclic agreement forest* (AAF) of (T_1, T_2) ; otherwise, F is a *cyclic agreement forest* (CAF) of (T_1, T_2) . If F is an AAF of (T_1, T_2) and its size is minimized over all AAFs of (T_1, T_2) , then F is a *maximum acyclic agreement forest* (MAAF) of (T_1, T_2) . The *hybridization number* of (T_1, T_2) is the size of an MAAF of (T_1, T_2) , and is denoted by $h(T_1, T_2)$.

We are now ready to define the problems studied in this paper:

Hybridization Number Computation (HNC):

Input: A TT-pair (T_1, T_2) .

Output: The hybridization number of (T_1, T_2) .

rSPR Distance Computation (RDC):

Input: A TT-pair (T_1, T_2) .

Output: The rSPR distance of (T_1, T_2) .

2.3 Transforming a CAF to an AAF

Suppose that F is a CAF of a TT-pair (T_1, T_2) . We construct a directed graph D as follows. For every non-leaf vertex of F , we create a vertex in D . There is an edge in D from a vertex u to a vertex v precisely if in either F_1 or F_2 (or in both), there is a directed path from u to v . A *minimum directed feedback vertex set* (MDFVS) of D is a minimum-sized set U of vertices in D such that modifying D by removing the vertices in U yields a directed acyclic graph.

► **Lemma 2** ([14]). *Let U be an MDFVS of D . Then, to transform F to an AAF of (F_1, F_2) by performing a minimum number of detaching operations on F , it suffices to modify F by removing the vertices corresponding to those in U and further contracting unifurcate vertices.*

Let V be the set of vertices in D . By Lemma 2, to compute an MDFVS U of D , it suffices to solve the following integer linear programming (ILP) model [16]:

$$\text{Minimize : } \sum_{v \in V} x_v \quad (1)$$

$$\text{s.t. } \quad 0 \leq \ell_v \leq |V| - 1 \quad \text{for all } v \in V \quad (2)$$

$$\ell_v \geq \ell_u + 1 - |V|x_u - |V|x_v \quad \text{for all } e = (u, v) \in E \quad (3)$$

$$\ell_v \in \mathbb{Z} \quad \text{for all } v \in V \quad (4)$$

$$x_v \in \{0, 1\} \quad \text{for all } v \in V \quad (5)$$

Fortunately, in our application, we will have an integer k and only want to know whether the optimal value of the objective function is bounded by k from above. So, we modify the model by replacing the objective function with any constant (say, 0) and adding the new constraint $\sum_{v \in V} x_v \leq k$. We refer to this modified model as *the ILP model associated with (T_1, T_2, F, k)* .

3 Solving HNC Exactly

Our algorithm for solving HNC exactly will use a subroutine for the following parameterized version of HNC.

Parameterized HNC (PHNC):

Input: (T_1, T_2, F_1, F_2, k) , where (T_1, T_2) is a TT pair, (F_1, F_2) is a sub-FF pair of (T_1, T_2) , and k is an integer.

Output: “Yes” if performing k more detaching operations on F_2 leads to an AAF of (T_1, T_2) ; “no” otherwise.

Several definitions are in order. Let (F_1, F_2) be an FF-pair, and $i \in \{1, 2\}$. A vertex v of F_i is *active* if v is a matched non-root vertex of F_i and its parent in F_i is not matched. Since (F_1, F_2) is an FF-pair, all active vertices of F_1 fall into the same component tree of F_1 . An *active sibling-pair* of F_i is a pair (u, v) of active vertices in F_i such that u and v are siblings in F_i .

3.1 Key Ideas

Basically, our algorithm is a significantly refined version of the algorithm for HNC implemented in Chen and Wang’s UltraNet [10]. In this subsection, we list the key new ideas behind our new algorithm for HNC.

First, the new algorithm builds on a recent 2-approximation algorithm for RDC [7]. When we compute the hybridization number, we use the lower bound outputted by the approximation algorithm to bound the search of the hybridization number. Since the lower bound is often nearly optimal, this bounding idea makes it possible for our algorithm to find the hybridization number in short time. Since the exact algorithm for RDC in [7] is also fast, we can use it to bound the search of the hybridization number instead of using the 2-approximation algorithm for RDC.

Secondly, the new algorithm is recursive and we make child recursive-calls in a careful order. More precisely, child recursive-calls that appear to finish in shorter time are made earlier than those that look likely to finish in longer time.

Thirdly, when we make a recursive call, we may know certain vertices v such that the subtree rooted at v should not be detached, and so we lock these vertices so that the subtrees rooted at them will never be detached in subsequent recursive calls. Moreover, the locked vertices help us make fewer child recursive-calls.

Finally, when our algorithm needs to transform a CAF F of a TT-pair (T_1, T_2) to an AAF of (T_1, T_2) , we use the ILP-based method outlined in Section 2.3. However, we modify the ILP model in Section 2.3 as follows.

- Let D be the digraph constructed from F and (T_1, T_2) as in Section 2.3. Since F is a CAF, D has a cycle and we need to remove at least one vertex from D to make D acyclic. Once D becomes acyclic, its number of vertices has decreased by at least 1. So, it is safe to modify the ILP model by changing the upper bound on the value of ℓ_v from $|V| - 1$ to $|V| - 2$.
- Some vertices of F may have been locked. So, for each locked vertex v of F , we can modify the model by fixing $x_v = 0$.
- By Lemma 4 in [9], we know that for each edge (p, c) of F , if removing a set U of vertices from D with $\{p, c\} \subseteq U$ makes D acyclic, then removing the vertices of $U \setminus \{c\}$ also makes D acyclic. Thus, for each edge (p, c) of F , we can add the constraint $x_c \leq x_p$ to the model.

3.2 The Algorithm

Throughout this subsection, fix an instance (T_1, T_2) of HNC.

Our algorithm for computing $h(T_1, T_2)$ exactly first repeatedly performs a *cluster reduction* on T_1 and T_2 until no such reduction is applicable. For the detail of *cluster reductions*, the reader is referred to [2]. As the result of zero or more cluster reductions on T_1 and T_2 , we obtain a sequence $(T_{1,1}, T_{2,1}), \dots, (T_{1,q}, T_{2,q})$ of instances of HNC such that $q \geq 1$ and $h(T_1, T_2) = \sum_{i=1}^q h(T_{1,i}, T_{2,i})$. Hence, it suffices to compute $h(T_{1,i}, T_{2,i})$ for each $i \in \{1, \dots, q\}$. Therefore, for simplicity, we hereafter assume that $q = 1$ and in turn $(T_1, T_2) = (T_{1,1}, T_{2,1})$.

Our algorithm then uses the program in [7] for RDC to compute $d(T_1, T_2)$. The program can also output an AF F of (T_1, T_2) with size $d(T_1, T_2)$. So, our algorithm checks whether F is indeed an AAF of (T_1, T_2) (by constructing the decision graph G_F associated with F and testing if G_F is acyclic or not). If it is, then $d(T_1, T_2)$ is also $h(T_1, T_2)$ and so the algorithm outputs $d(T_1, T_2)$ and stops. Thus, we hereafter assume that F is not an AAF of (T_1, T_2) .

To compute $h(T_1, T_2)$, it suffices to solve PHNC on input (T_1, T_2, T_1, T_2, k) for $k = d(T_1, T_2), d(T_1, T_2) + 1, \dots$ (in this order) until a “yes” is returned. So, it remains to detail our algorithm for PHNC. During its execution, our algorithm will lock certain non-root vertices v of F_2 at certain time points so that F_2^v will never be detached thereafter; it will always maintain the following invariant:

Invariant 1: Whenever a non-root vertex is locked by the algorithm, it knows that it will return “yes” with the locking if and only if it will return “yes” without the locking.

Our algorithm for PHNC is recursive and proceeds as follows. It starts by checking whether $k \geq 0$. If $k < 0$, then this is **Base Case 1** and it returns “no”. So, we hereafter assume $k \geq 0$. Then, it simplifies (F_1, F_2) and further checks the following base case:

Base Case 2: *All roots of F_1 are matched.* In this case, F_1 and F_2 are the same forest and hence F_2 is an AF of (T_1, T_2) . To test if F_2 is an AAF, our algorithm constructs the decision graph G_{F_2} associated with F_2 and tests if it is acyclic or not. If G_{F_2} is acyclic, then it returns “yes”. Otherwise, it checks if $k \geq 1$ or not. If $k \leq 0$, then it returns “no”. On the other hand, if $k \geq 1$, then it constructs the ILP model associated with (T_1, T_2, F_2, k) and solves the ILP model by an ILP solver (say, CPLEX or GUROBI); it returns “yes” if and only if the model is feasible.

We hereafter assume that one or more roots of F_1 are still not matched. Our algorithm then uses the program in [7] to compute a lower bound b_ℓ and an upper bound b_u on $d(F_1, F_2)$. The program will also return an AF F of (F_1, F_2) with size b_u as a witness for b_u . If $k < b_\ell$, then this is **Base Case 3**, and the algorithm returns “no”. Otherwise, the algorithm checks if the ILP model associated with (T_1, T_2, F, k) is feasible or not. If it is feasible, then this is **Base Case 4**, and the algorithm returns “yes”.

We hereafter assume that $k \geq b_\ell$ and the ILP model associated with (T_1, T_2, F, k) is infeasible. Clearly, both F_1 and F_2 must have at least one active sibling-pair. Our algorithm now distinguishes several cases *in the following order*.

Case 1: There is an active sibling-pair (u, v) in F_1 such that $|D_{F_2}(u, v)| = 1$. In this case, we clearly know that to transform F_2 into an AF of (F_1, F_2) , we need to select at least one $x \in \{u, v, w\}$ and detach F_2^x , where $D_{F_2}(u, v) = \{w\}$. So, if all vertices of $\{u, v, w\}$ are locked, then this is **Base Case 5**, and the algorithm returns “no”. Thus, we may assume that at least one vertex of $\{u, v, w\}$ is not locked. As observed in [18], selecting $x = u$ is the same as selecting $x = v$ (which means that the former selection leads to a “yes”-output if and only if so does the latter). Hence, if u or v is not locked, then our algorithm chooses an arbitrary unlocked $x \in \{u, v\}$ and makes a recursive call on input $(T_1, T_2, F_1, F_2', k - 1)$, where F_2' is obtained from F_2 by detaching F_2^x . In addition, if w is also not locked, then our algorithm makes a recursive call on input $(T_1, T_2, F_1, F_2'', k - 1)$, where F_2'' is obtained from F_2 by detaching F_2^w and further locking x in case the recursive call on input $(T_1, T_2, F_1, F_2', k - 1)$ has been made. So, we make one or two recursive calls. If at least one call returns “yes”, the algorithm returns “yes”; otherwise, it returns “no”.

Case 2: There is an active sibling-pair (u, v) in F_2 such that $|D_{F_1}(u, v)| = 1$ and the unique vertex w in $D_{F_1}(u, v)$ is active. This case is symmetric to Case 1; so, the algorithm proceeds as in Case 1 except that each of u, v , and w is replaced by its mate.

Case 3: Neither Case 1 nor 2 occurs. In this case, our algorithm searches F_1 for an active sibling-pair (u, v) *in the following order*:

Type 1: Both u and v are locked in F_2 .

Type 2: u and v belong to different connected components of F_2 .

Type 3: Either u or v is locked in F_2 .

Type 4: The sibling s of the parent of u and v in F_1 satisfies that either s is active or both children of s in F_1 are active.

Type 5: (u, v) is of none of the above types.

We emphasize that the smaller type of an active sibling-pair in F_1 is, the more our algorithm prioritizes it. Intuitively speaking, choosing an active sibling-pair of a smaller type in F_1 will likely lead to fewer recursive calls.

Suppose that our algorithm has selected an active sibling-pair (u, v) in F_1 as above. Our algorithm constructs a family \mathcal{F} of sets as follows. Initially, \mathcal{F} is empty. For each $y \in \{u, v\}$ such that y is not locked in F_2 , we add the set $\{y\}$ to \mathcal{F} . Moreover, if no vertex in $D_{F_2}(u, v)$ is locked in F_2 , then we add $D_{F_2}(u, v)$ to \mathcal{F} . Since Case 1 does not occur, $|D_{F_2}(u, v)| \geq 2$. Clearly, to transform F_2 into an AF of (F_1, F_2) , we need to select a set $S \in \mathcal{F}$ and detach F_2^w for all $w \in S$. Thus, if \mathcal{F} is empty, then our algorithm returns “no”. Otherwise, it sorts the sets in \mathcal{F} so that larger sets precede smaller sets. Let S_1, \dots, S_t be the sets in \mathcal{F} . For each $i \in \{1, \dots, t\}$, let $F_{2,i}$ be the phylogenetic forest obtained from F_2 by first detaching F_2^y for all $y \in S_i$ and further distinguishing two cases as follows:

1. If $|S_i| \geq 2$, then lock both u and v in F_2 .
2. If $i \geq 2$ and $|S_{i-1}| = |S_i| = 1$, then lock the vertex of S_{i-1} in F_2 .

Now, our algorithm makes t recursive calls on input $(T_1, T_2, F_1, F_{2,1}, k - |S_1|), \dots, (T_1, T_2, F_1, F_{2,t}, k - |S_t|)$. If at least one call returns “yes”, the algorithm returns “yes”; otherwise, it returns “no”.

4 Solving RDC Approximately

Basically, we want an approximate algorithm that outputs better results than the algorithm in [7]. Although the algorithm in [8] has a worse theoretical-guarantee than the algorithm in [7], it does not necessarily mean that the former always outputs worse results. So, we obtain a new approximation algorithm for RDC which simply runs the algorithms in [7, 8] and outputs the better result returned by them.

Our new idea is to use MCTS to improve the performance of *any* approximation algorithm for RDC. MCTS has a number of variants, but we here use the basic one (namely, the UCT algorithm) for its simplicity.

4.1 Outline of the Algorithm

In the remainder of this section, fix an FF-pair (F_1, F_2) . Our algorithm for computing $d(F_1, F_2)$ approximately is recursive and starts by simplifying (F_1, F_2) and further checking whether F_2 is already an AF of (F_1, F_2) . If it is, then this is **Base Case 1** and it returns 0. So, assume that F_2 is not an AF of (F_1, F_2) . Then, F_1 has a unique non-matched root r . If r has at most 6 leaf descendants in F_1 , then this is **Base Case 2** and our algorithm computes $d(F_1, F_2)$ in $O(1)$ time by brute force. Thus, we further assume that r has more than 6 leaf descendants in F_1 . Now, our algorithm finds a *promising* vertex z in F_2 , next detaches F_2^z , further makes a recursive call on the modified (F_1, F_2) , and finally returns $c + 1$, where c is the value returned by the recursive call.

It remains to consider how to find a promising z . In the following two cases, we know an optimal choice of z , i.e., we know that the choice of z will lead to an optimal solution [6]:

- **Optimal Case 1:** (u, v) is an active sibling-pair in F_1 with $|D_{F_2}(u, v)| = 1$. In this case, z is the unique vertex in $D_{F_2}(u, v)$.
- **Optimal Case 2:** (u, v) is an active sibling-pair in F_2 with $|D_{F_1}(u, v)| = 1$ and the unique vertex in $D_{F_1}(u, v)$ is a leaf. In this case, z is the mate of the unique vertex in $D_{F_1}(u, v)$.

We hereafter assume that none of the above optimal cases occurs. Next, we outline how to find a promising z with MCTS. The idea behind MCTS is to build a small-sized search tree Γ . We will always use ρ to denote the root of Γ . In our case, each node α of Γ holds the following information:

- $f(\alpha)$: A sub-FF pair of (F_1, F_2) . (*Comment*: We use $f(\alpha)_1$ and $f(\alpha)_2$ to denote the first and the second forest in $f(\alpha)$, respectively.)
- $t(\alpha)$: The *number of times* α has been visited so far.
- $s(\alpha)$: The *score* of α .
- $Q(\alpha)$: the *reward* α has received so far.

When creating a node α , we are always given a sub-FF pair (\hat{F}_1, \hat{F}_2) of (F_1, F_2) and initialize $f(\alpha) = (\hat{F}_1, \hat{F}_2)$, $t(\alpha) = 0$, $s(\alpha) = 0$, and $Q(\alpha) = 0$. To evaluate a child α of a node β of Γ , we use the *UCT value* of α , which is computed as follows:

$$\frac{Q(\alpha)}{t(\alpha)} + C \cdot \sqrt{\frac{2 \ln t(\beta)}{t(\alpha)}},$$

where C is a constant (called the *balance constant* and fixed to be 0.2 in our experiments). The *best child* of a node β in Γ is the child of β in Γ whose UCT value is maximized over all children of β in Γ .

Initially, Γ has a unique node, namely, the root ρ created with (F_1, F_2) . We then grow Γ by repeatedly performing the following steps (in this order) for a predetermined number (fixed to be 60 in our experiments) of repetitions:

1. Select a leaf-node α in Γ by starting at ρ and repeatedly descending to the best child of the current node until reaching a leaf. (*Comment*: Ties are broken arbitrarily.)
2. Expand α . (*Comment*: See Section 4.2.)
3. Perform a simulation for α by calling an approximation algorithm (say, the algorithm in [7]) on input $f(\alpha)$, and then update $s(\alpha)$ to $App(f(\alpha)) + |f(\alpha)_2| - |f(\rho)_2|$, where $App(f(\alpha))$ means the approximate rSPR distance of $f(\alpha)$ returned by the approximation algorithm. (*Comment*: We refer to this step as the *simulation step*.)
4. Compute the reward $Q(\alpha) = \begin{cases} 1 & \text{if } s(\alpha) \leq \text{the average score of the nodes in } \Gamma \\ 0 & \text{otherwise} \end{cases}$.
5. Backpropagate the reward $Q(\alpha)$ from α all the way to the root ρ by performing the following step for all ancestors β of α in Γ :
 - Increase $t(\beta)$ by 1 and increase $Q(\beta)$ by $Q(\alpha)$,

Once finishing growing Γ as above, we select the best child γ of ρ . As will be detailed in Section 4.2, $f(\gamma)_2$ is obtained from $f(\rho)_2$ by detaching the subtrees rooted at the vertices of a set S . Finally, we set z to be an arbitrary vertex in S .

4.2 Expanding a Node α

Suppose that we have selected a leaf node α to expand. We first simplify $f(\alpha)$ and then search $f(\alpha)_1$ and $f(\alpha)_2$ for an active sibling-pair (u, v) in the following order:

Type 1: (u, v) is an active sibling-pair in $f(\alpha)_1$ with $|D_{f(\alpha)_2}(u, v)| = 1$.

Type 2: (u, v) is an active sibling-pair in $f(\alpha)_2$ such that $|D_{f(\alpha)_1}(u, v)| = 1$ and the unique vertex in $D_{f(\alpha)_1}(u, v)$ is a leaf

Type 3: (u, v) is an active sibling-pair in $f(\alpha)_1$ such that u and v belong to different connected components of $f(\alpha)_2$.

Type 4: (u, v) is an active sibling-pair in $f(\alpha)_1$ such that u and v belong to the same connected component of $f(\alpha)_2$ and $\ell_{f(\alpha)_2}(u, v)$ is a root of $f(\alpha)_2$.

Type 5: (u, v) is of none of the above types.

We emphasize that the smaller type of an active sibling-pair is, the more our algorithm prioritizes it.

If (u, v) is not found, we know that $f(\alpha)_2$ is an AF of $f(\alpha)$ and hence we have nothing to do with expanding α . Thus, we hereafter assume that (u, v) has been found. Then, we construct a family \mathcal{F} of sets as follows.

- If (u, v) is of Type 1 (respectively, 2), then \mathcal{F} consists of only $D_{f(\alpha)_2}(u, v)$ (respectively, $D_{f(\alpha)_1}(u, v)$).
- If (u, v) is of Type 3 or 4, then \mathcal{F} consists of $\{u\}$ and $\{v\}$.
- If (u, v) is of Type 5, then \mathcal{F} consists of $\{u\}$, $\{v\}$, and $D_{f(\alpha)_2}(u, v)$.

We now use \mathcal{F} to create the children of α as follows. For each set $S \in \mathcal{F}$, we create a child β_S , where $f(\beta_S)_1 = f(\alpha)_1$ and $f(\beta_S)_2$ is obtained from $f(\alpha)_2$ by detaching the subtrees rooted at the vertices in S .

5 Solving HNC Approximately

We say that an approximation algorithm A for RDC is *useful* if given a TT-pair (T_1, T_2) , A can not only output an approximate value d' of $d(T_1, T_2)$ but also output an AF F of (T_1, T_2) with $|F| = d'$. Our approximation algorithm given in Section 4 is useful and so are all known approximation algorithms for RDC. Using a useful approximation A for RDC, we can design an approximation algorithm for HNC, denoted by A_{hn} , as follows. Given a TT-pair (T_1, T_2) , A_{hn} calls A to obtain an approximate value d' of $d(T_1, T_2)$ and an AF F of (T_1, T_2) with $|F| = d'$. If F is an AAF of (T_1, T_2) , then d' is also an approximate value of $h(T_1, T_2)$ and hence A_{hn} returns d' . So, assume that F is a CAF of (T_1, T_2) . Then, as in Section 2.3, we can transform F into an AAF of (T_1, T_2) by solving an ILP model. Thus, d' plus the optimal value of the objective function of the model gives us an approximate value of $h(T_1, T_2)$ and so A_{hn} returns it.

6 Experimental Results

To compare our new algorithms against the previous bests, we have implemented them in Java. In this section, we compare the real performance of our programs against that of the previous bests. In our experiments, we use a Linux (x64) desktop PC with Intel i7-4790 CPU (4.00GHz, 8 threads) and 32GB RAM. As the ILP solver, we use the IBM CPLEX which is freely available for academic research.

We define the *average approximation ratio* (AAR) of an approximation algorithm A (for RDC or HNC) as follows. For a given instance I , we use $A(I)$ (respectively, $B(I)$) to denote the value outputted by A (respectively, an exact algorithm) on input I ; the *approximation ratio* of A for I , denoted by $r_A(I)$, is $\frac{A(I)}{B(I)}$. The AAR of A for a set \mathcal{I} of instances is $\frac{\sum_{I \in \mathcal{I}} r_A(I)}{|\mathcal{I}|}$.

As in previous studies [1, 3, 7, 10, 16, 17], we here generate simulated datasets randomly. More specifically, for a given pair (n, m) of parameters, we use the program of [3] to generate a dataset consisting of 120 TT-pairs, where each TT-pair is generated by first generating a random phylogenetic tree T_1 with n leaves and then obtaining another phylogenetic tree T_2 by applying m random rSPR operations on T_1 . So, the rSPR distance of each pair (T_1, T_2) in the dataset is at most m , but the hybridization number of (T_1, T_2) may be larger than m . In our experiments stated below, we choose (n, m) from $\{(100, 50), (200, 80), (200, 100)\}$ and generate a dataset $\mathcal{I}(n, m)$ for each (n, m) in this set.

■ **Table 1** Comparing the AARs of Approximation Algorithms for RDC.

Svv	CMW	CHN	CombApp	MCTS_CMW	MCTS_CHN	CombMCTS
1.41	1.133	1.135	1.104	1.03	1.03	1.019
1.391	1.141	1.127	1.108	1.048	1.044	1.031

The first and the second rows show the results for $\mathcal{I}(100, 50)$ and $\mathcal{I}(200, 100)$, respectively; Svv, CMW, and CHN mean the algorithm in [15], [8], and [7], respectively; MCTS_CMW and MCTS_CHN mean our MCTS algorithm with CMW and CHN used in the simulation step, respectively; CombApp (respectively, CombMCTS) means the algorithm which runs CMW and CHN (respectively, MCTS_CMW and MCTS_CHN) and outputs the better solution returned by them.

■ **Table 2** Comparing the AARs of Approximation Algorithms for HNC.

Svv _{hn}	CMW _{hn}	CHN _{hn}	CombApp _{hn}	MCTS_CMW _{hn}	MCTS_CHN _{hn}	CombMCTS _{hn}
1.397	1.146	1.134	1.1	1.032	1.031	1.02
1.419	1.087	1.083	1.062	1.021	1.02	1.015

The first and the second rows show the results for $\mathcal{I}(100, 50)$ and $\mathcal{I}(200, 80)$, respectively.

6.1 Results on Approximating RDC

Since all programs used in our experiments for approximating RDC are fast, it is meaningless to compare them in terms of running time. So, we compare them in terms of their AARs. We use $\mathcal{I}(100, 50)$ and $\mathcal{I}(200, 100)$ in the experiment. Our experimental results are summarized in Table 1. From the table, we can see that MCTS is very helpful in improving the performance of approximation algorithms for RDC. In particular, our best algorithm (namely, CombMCTS) achieves a significantly better AAR than the previous best (namely, CHN). We did not test MCTS_Svv because the source code of Svv has not been made public.

6.2 Results on Approximating HNC

Since we want the exact hybridization number to be known, we use the two easiest datasets (namely, $\mathcal{I}(100, 50)$ and $\mathcal{I}(200, 80)$) in this experiment to compare the AARs of our approximation algorithms for HNC against the previous bests. Our experimental results are summarized in Table 1. From the table, we can see that MCTS is very helpful in improving the performance of approximation algorithms for HNC as well. In particular, our best algorithm (namely, CombMCTS_{hn}) achieves a much better AAR than the previous best (namely, Svv_{hn}). Indeed, our experimental results show that for about half the tested instances, CombMCTS_{hn} found optimal solutions.

6.3 Results on Computing HNC Exactly

To compare the speed of our new exact algorithm for HNC against the previous best (namely, UltraNet in [10]), we use $\mathcal{I}(100, 50)$ and $\mathcal{I}(200, 80)$ again. For each tested instance, we set a 1-hour time limit on the running time of each program. As the result, UltraNet fails to solve **1** (respectively, **16**) instances of $\mathcal{I}(100, 50)$ (respectively, $\mathcal{I}(200, 80)$), while our new program fails to solve none. With the failed instances excluded, the average running time of UltraNet is **54.46** (respectively, **323.86**) seconds for the first (respectively, second) dataset, while that

of our new program is only **0.66** (respectively, **0.86**) seconds. So, our new program is more than **82 times faster** than UltraNet and its superiority in speed over UltraNet becomes more significant for larger instances.

References

- 1 B. Albrecht, C. Scornavacca, A. Cenci, and D.H. Huson. Fast computation of minimum hybridization networks. *Bioinformatics*, 28(2):191–197, 2012.
- 2 M. Baroni, C. Semple, and M. Steel. Hybrids in real time. *Systematic Biology*, 55(1):46–56, 2006.
- 3 R.G. Beiko and N. Hamilton. Phylogenetic identification of lateral genetic transfer events. *BMC Evolutionary Biology*, 6(15):159–169, 2006.
- 4 M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8(4):409–423, 2005.
- 5 C. Browne, E. Powley, D. Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–49, 2012.
- 6 Z.-Z. Chen, Y. Fan, and L. Wang. Faster exact computation of rSPR distance. *Journal of Combinatorial Optimization*, 29(3):605–635, 2015.
- 7 Z.-Z. Chen, Y. Harada, Y. Nakamura, and L. Wang. Faster exact computation of rSPR distance via better approximation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, to appear.
- 8 Z.-Z. Chen, E. Machida, and L. Wang. An Approximation Algorithm for rSPR Distance. In *22nd International Computing and Combinatorics Conference, Ho Chi Minh City, Vietnam, August 2-4, 2016*, pages 468–479, 2016.
- 9 Z.-Z. Chen and L. Wang. Algorithms for reticulate networks of multiple phylogenetic trees. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 9(2):372–384, 2012.
- 10 Z.-Z. Chen and L. Wang. An ultrafast tool for minimum reticulate networks. *Journal of Computational Biology*, 20(1):38–41, 2013.
- 11 L. Collins, S. Linz, and C. Semple. Quantifying hybridization in realistic time. *J. of Comput. Biol.*, 18(10):1305–1318, 2011.
- 12 J. Hein, T. Jing, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Disc. Appl. Math.*, 71(1-3):153–169, 1996.
- 13 D.H. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, 2010.
- 14 S. Kelk, L. van Iersel, N. Lekic, S. Linz, C. Scornavacca, and L. Stougie. Cycle killer...qu’est-ce que c’est? On the comparative approximability of hybridization number and directed feedback vertex set. *SIAM J. Discrete Math.*, 26(4):1635–1656, 2012.
- 15 F. Schalekamp, A. van Zuylen, and S. van der Ster. A Duality Based 2-Approximation Algorithm for Maximum Agreement Forest. In *43rd International Colloquium on Automata, Languages and Programming, Rome, Italy, July 11-15, 2016*, pages 70:1–70:14, 2016.
- 16 L. van Iersel, S. Kelk, N. Lekic, and C. Scornavacca. A practical approximation algorithm for solving massive instances of hybridization number for binary and nonbinary trees. *BMC Bioinformatics*, 15(127), 2014.
- 17 C. Whidden, R.G. Beiko, and N. Zeh. Fast FPT algorithms for computing rooted agreement forest: theory and experiments. In *International Symposium on Experimental Algorithms, Naples, Italy, May 20-22, 2010*, pages 141–153, 2010.
- 18 C. Whidden, R.G. Beiko, and N. Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM J. Comput.*, 42(4):1431–1466, 2013.
- 19 C. Whidden and N. Zeh. A unifying view on approximation and FPT of agreement forests. In *9th International Workshop on Algorithms in Bioinformatics, Philadelphia, PA, USA, September 12-13, 2009*, pages 390–401, 2009.
- 20 Y. Wu. A practical method for exact computation of subtree prune and regraft distance. *Bioinformatics*, 25(2):190–196, 2009.

pClay: A Precise Parallel Algorithm for Comparing Molecular Surfaces

Georgi D. Georgiev*

Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA

Kevin F. Dodd*

Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA

Brian Y. Chen¹

Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
chen@cse.lehigh.edu

Abstract

Comparing binding sites as geometric solids can reveal conserved features of protein structure that bind similar molecular fragments and varying features that select different partners. Due to the subtlety of these features, algorithmic efficiency and geometric precision are essential for comparison accuracy. For these reasons, this paper presents pClay, the first structure comparison algorithm to employ fine-grained parallelism to enhance both throughput and efficiency. We evaluated the parallel performance of pClay on both multicore workstation CPUs and a 61-core Xeon Phi, observing scaleable speedup in many thread configurations. Parallelism unlocked levels of precision that were not practical with existing methods. This precision has important applications, which we demonstrate: A statistical model of steric variations in binding cavities, trained with data at the level of precision typical of existing work, can overlook 46% of authentic steric influences on specificity ($p \leq .02$). The same model, trained with more precise data from pClay, overlooked 0% using the same standard of statistical significance. These results demonstrate how enhanced efficiency and precision can advance the detection of binding mechanisms that influence specificity.

2012 ACM Subject Classification Applied computing → Molecular structural biology; Computing methodologies → Volumetric models; Computing methodologies → Parallel algorithms

Keywords and phrases Specificity Annotation, Structure Comparison, Cavity Analysis

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.6

1 Introduction

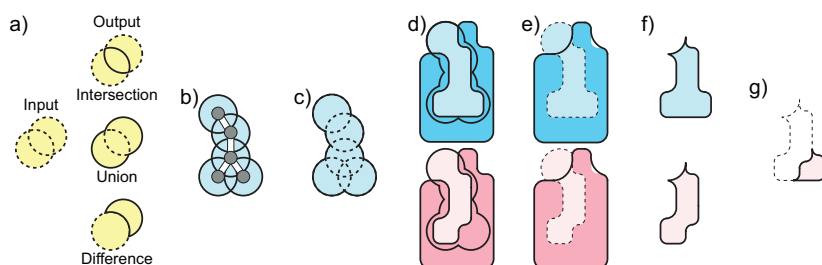
Molecular shape and electric fields have a strong influence on binding specificity. At binding interfaces, complementary molecular shapes can accommodate some ligands and hinder those that fit poorly. Electric fields attract molecules with complementing charges and repel others. This connection, between molecular recognition and the geometric complementarity of surfaces and fields, is evidence by which human investigators infer the roles of individual mechanisms in function. Comparison software can detect this kind of evidence and use it to make similar inferences. Some methods detect proteins with geometrically conserved binding sites, supporting the inference that they bind similar partners [12, 7, 4, 27, 33, 11, 15, 18]. Other methods find variations in the electric fields near binding sites, suggesting that they accommodate differently charged ligands [19, 5, 29, 34]. These techniques, and their potential for large scale and accurate applications, depend on rapid and precise digital representations of molecular shape, which are the focus of this paper.

Rapid and precise algorithms can integrate many observations to support inferences that are impossible with single comparisons. For example, a single comparison does not provide a

¹ Corresponding Author * Equal contribution



frame of reference that would be needed to assess whether or not two binding sites are different enough that they have different binding preferences. After all, conformational variations and single mutations can occur in many ways that change nothing about binding. This kind of inference is traditionally reserved for experts with a wealth of biochemical experience. However, statistical models can be trained on the steric differences between closely related ligand binding sites that prefer the same ligands. In such cases, structures of close homologs or even single mutants could provide the primary data, but the subtle variations needed to train the model would have to be found with many individual comparisons. Once trained, the statistical model provides a frame of reference that reveals steric variations that are too large to be typical of binding sites with the same binding preferences. The large variations found would therefore be indicators of binding sites that have different binding preferences [6]. To support and advance statistical models like these, this paper presents pClay, the first structure comparison algorithm that maximizes precision and computational throughput using arbitrary precision representations and parallel algorithms.



■ **Figure 1** CSG operations on Molecular Surfaces. a) Basic CSG operations. Input solids are yellow with dotted outlines. Outputs have solid outlines. b) Ligand with grey atoms and white bonds, with spheres centered on each atom (light blue). c) The union of atom-centered spheres. d) Two molecular surfaces (blue, red) in complex with two ligands shown as sphere unions (black lines). e) CSG difference of the sphere unions minus molecular surfaces (dotted lines), shown with molecular surfaces (blue and red, no outline) and envelope surfaces (black outline). f) Intersection of differences with envelope surfaces (light blue and red). g) The CSG difference between binding cavities reveals a variation in steric hindrance that causes differences in binding preferences.

pClay performs geometric comparisons using Constructive Solid Geometry (CSG) operations (Fig. 1a) on analytically represented three dimensional solids. These operations, which include unions, intersections and differences, can be combined like arithmetic operators to sculpt a geometric solid. This sculptural nature of CSG inspires both the name pClay, a portmanteau of “protein” and “clay,” and also the solid geometric approach to the analysis of protein shape that pClay makes possible. For example, the union of large spheres centered at ligand atoms can represent the neighborhood of a ligand (Fig. 1b,c). The difference between the spheres and the molecular surface of a receptor can describe the solvent-accessible binding cavity in the receptor (Fig. 1d,e). The CSG difference between one binding cavity and another is the cavity region that is solvent accessible in one protein and inaccessible in the other (Fig. 1g). This difference, the variation between the two cavities, could be small, when binding preferences are similar, or large, when steric hindrance creates differences in specificity.

The utility of these computations can be seen in multiple applications: When applying this approach to the S1 subsites of trypsins and elastases, we observed that it could identify threonine 226 which, in elastases, sterically hinders the longer substrates preferred by trypsins that might otherwise bind [8]. To illustrate the importance of precision, that region of hindrance is only 50 percent larger than a carbon atom (31 \AA^3). A similar approach identified

“gatekeeper” residue 338 in the tyrosine kinases [14], which creates steric clash with larger drugs [22]. We have also observed that a CSG-based comparison of electrostatic isopotentials can reveal single amino acids crucial for selecting ligands in the in the cysteine proteases [5] and for stabilizing the three interfaces of the SMAD trimer [29]. Experimental validation has demonstrated the correctness of our prediction that arginine 235 forms critical electrostatic interactions for the activity of the ricin toxin [34]. By making CSG analysis possible on geometric solids that are exact, up to machine precision, pClay ensures that subtle but influential details cannot be overlooked.

The precision that pClay achieves derives from solids that have analytical representations, like spheres and tetrahedra. pClay can assemble these primitives into solvent excluded regions, which we call *molecular solids*. The boundary of a molecular solid is the classic molecular surface, also known as the solvent excluded surface or Connolly surface, which was originally developed by Richardson and others [21, 9]. While we can construct molecular solids with CSG operations on many individual primitives, pClay exploits molecular properties to sidestep those operations and achieve greater efficiency. The resulting molecular solids avoid the “photocopier effect”, where multiple CSG operations can accumulate geometric errors. They can also be exported as triangle meshes, generated at an arbitrary degree of precision, for compatibility with other software.

pClay enhances computational efficiency through parallelism. We achieve parallelism in pClay in a number of ways, most notably by recasting Marching Cubes, a traditional method for implementing CSG operations [23, 17], into a series of parallel breadth first searches (BFS). In pClay, we use BFS to traverse cubic lattices and identify contiguous regions of cubes within defined boundary regions. These breadth first traversals can be distributed evenly across arbitrary numbers of threads. By dividing the computation in this way, parallelism can make comparisons faster and also enable more detail to be considered. This advancement stands in qualitative contrast with existing efforts to parallelize structure comparisons (e.g. [7]), where throughput was increased without enhancing precision. To demonstrate the parallel scalability of our method, pClay was tested on both modern multicore processors as well as on a Xeon Phi, a manycore processor with 61 cores.

Relative to existing methods, pClay is the first algorithm to use arbitrarily precise representations of molecular surfaces for protein structure comparison. It is also the first structure comparison method to use fine grained parallelization, enhancing both precision and computational throughput. Several methods do employ arbitrarily precise representations of the molecular surface, using NURBs [2], alpha shapes [32] or spherical coordinates [26, 28], but they are used for visualization and have not been integrated into comparison algorithms. Other methods parallelize structure comparison to refine representations of binding sites [7], to accelerate database searches [20], or create cloud-based search services [16], but use parallelism to enhance throughput and not also precision. To our knowledge, pClay is the first integration of arbitrary precision and parallelism into a structure comparison method.

2 Methods

As input, pClay accepts a collection of geometric solids and an expression of CSG operations. We convert the CSG expression into a binary tree, a CSG tree, where the nodes of the tree are geometric solids. The input solids, which include spheres, spindles, tetrahedra or molecular surfaces, are leaves on the CSG tree, while the result of CSG operations are the nonleaf nodes. The final result of all operations, the root node, is the output. pClay can also generate a closed triangular mesh at user-defined resolutions to approximate the boundary

of the output.

To perform CSG operations, pClay implements a parallel version of Marching Cubes [23] (Section 2.1), which we summarize below. Our method requires three basic functions to be performed by every node in the CSG tree. These functions are *containsPoint()*, *intersectSegment()*, and *findSurfaceCubes()*. Given any point p in three dimensions, *containsPoint(p)* determines exactly if p is inside or outside the solid. A point exactly on the surface is said to be inside the solid. Second, given a line segment s , *intersectSegment(s)* determines all points of intersection between the surface of the operand and s , as well as the interior or exterior state of each interval on the segment. Finally, given a cubic lattice l that surrounds the primitive, *findStartingCubes(l)* finds a few cubes of the lattice that are *surface cubes*, having at least one corner inside and one corner outside the solid. These cubes are used to initiate a parallel breadth first search for all surface cubes, called *findAllSurfaceCubes()*, which is implemented once for all primitives (Section 2.2). To implement leaf nodes it is thus sufficient to describe how these basic functions are implemented for that solid. Nonleaf nodes implement the basic functions as logical operations, as we will explain in Section 2.3.

Below, we first describe how the output approximations are generated using a parallelization of Marching Cubes and how we find all surface cubes beginning the output from the starting cubes generated by the basic function. We next explain how the three basic functions are implemented for every primitive. Finally, we detail how the basic functions are implemented in nonleaf nodes.

2.1 Parallel Marching Cubes

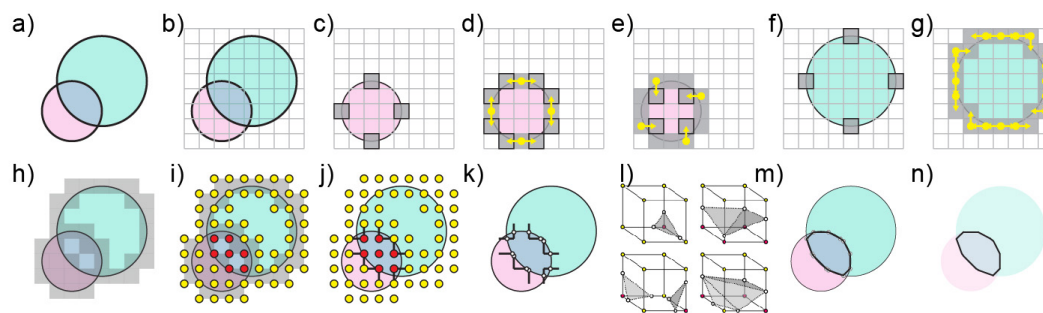
As input, Marching Cubes accepts a set of geometric solids (Fig. 2a), which we will refer to as operands, and a CSG expression tree to be performed on the operands. It also accepts a resolution parameter in angstrom units that specifies the degree to which the result of the CSG expression should be approximated in the output.

We begin by defining an axis aligned cubic lattice surrounding the input operands, where each cube has sides equal to the user-specified resolution parameter (Fig. 2b). This step is performed by examining the sizes of all operands and the related CSG operations.

Once the lattice is defined, we invoke *findStartingCubes(l)* on each input solid (Fig. 2c,f). The surface cubes identified are provided as input to *findAllSurfaceCubes()*, which identifies all remaining surface cubes of all inputs solids in parallel (Fig. 2h). The process of identifying surface cubes for all input solids also necessarily determines the interior/exterior state of the points on these cubes in relation to specific solids. We then compute the interior/exterior state of these points in relation to all other solids in an embarrassingly parallel manner. Once this assessment is made for any point, we can access whether that point is inside or outside the output region (Fig. 2i). In this way, we find the subset of cubes that contain a corner inside and a corner outside the output region.

Next, on each cube of the output surface, we identify edges that connect one corner that is inside the output region to one that is outside (Fig. 2j). Since these edges must pass through the output surface, we call *segIntersect()* on the root node to find the point of intersection between the edge and the output surface (Fig. 2k). This process is parallelized across the list of edges, ensuring that the calculation is never duplicated when dealing with adjacent cubes.

Finally, once intersections for every edge on every surface cube are determined, triangles are generated in each cube following a lookup table (Fig. 2l). The collection of all resulting triangles form a closed triangular mesh that approximates the output region (Fig. 2m,n).



■ **Figure 2** a) Input operands (red, green). b) Cubic lattice around operands (gray). c, f) surface cubes (gray boxes). d, e, g) several steps of floodfill propagation (starting at yellow circle, following yellow arrow). i) Corner points of each surface cube with exterior (yellow) or interior (red) state. j) Segments that cross the boundary of the output surfaces (Black lines). k) Intersection points (white circles) segments intersect the output surface. l) Lookup table of 3D surface constructions with different edge intersection patterns. m, n) Triangles (black lines) approximating output region (gray).

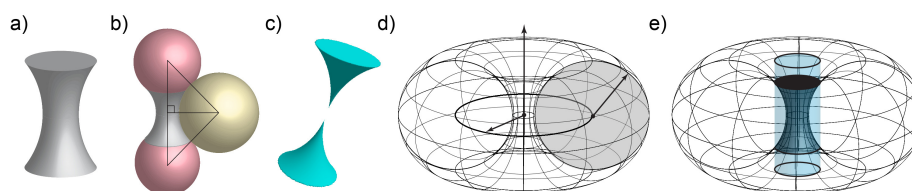
2.2 Finding All Surface Cubes

`findAllSurfaceCubes()` accepts a cubic lattice (Fig. 2b), a list of starting cubes (e.g. Fig. 2c, f), and a CSG tree node for which to find all remaining surface cubes. We perform a parallel floodfill algorithm to find all remaining surface cubes: Each available thread is assigned a cube from the queue. Each thread tests cubes adjacent to the assigned cube to find any that are also on the surface of the input solid (Fig. 2d). This test is performed by calling `containsPoint()` on the corners of the adjacent cube. If at least one corner is inside the input solid and another corner is outside, the adjacent cube is stored on a queue of upcoming cubes. Once all cubes adjacent to the initial surface cubes have been either added to the queue or discarded, all threads are then directed to find cubes adjacent to those still on the queue (e.g. Fig. 2e), and so on, until the queue is empty, and all cubes on the surface of the input solid have been identified. Duplicate entries onto the queue are avoided by recording previously-examined cubes on a parallel hash table.

2.3 Nodes of the CSG Tree

pClay supports several kinds of simple and complex solids for CSG operations. These are spheres, tetrahedra, spindles, and molecular surfaces. Our implementation of each type supports three basic functions: `containsPoint()`, `intersectSegment()`, and `findSurfaceCubes()`. To describe the implementation of these solids, we describe how each method is implemented for the solid. Spheres and tetrahedra are excluded because their implementations are trivial.

Spindles. Spindles (Fig. 3a) define the solvent excluded region between two atoms that are too close to permit a sphere representing a solvent molecule to pass between them (Fig. 3b). “Broken” spindles (Fig. 3c) can occur when the edge of the solvent sphere can pass beyond the centerline of the two atoms. Conceptually, spindles are the volume within a cylinder minus the volume within a coaxial torus. We define spindles by center point, perpendicular vector, major radius, and minor radius taken from the torus (Fig. 3d), and end cap positions along the perpendicular vector (Fig. 3e). The center point is the perpendicular projection of the center of the solvent sphere onto the segment between atom centers (Fig. 3b). The perpendicular vector points from the center point towards the center of one atom. The major radius is the radius of the circle defined by the center of the solvent sphere as it rotates



■ **Figure 3** a) Spindle. b) Formation of a spindle (gray) from two atoms (red) and a solvent sphere (yellow). c) “Broken” spindle. d) Torus defining the characteristics of a spindle, including center point (black dot), perpendicular vector (vertical arrow), major radius (arrow from center point to ellipse), minor radius (arrow from ellipse to torus surface). e) Cylinder (light blue).

about the two atoms. The minor radius is the radius of the solvent sphere. The endcaps are circles perpendicular to the perpendicular vector that are defined by the point of tangency between the solvent sphere and the atoms, as the solvent sphere rotates about the atoms. The boundary surface of a spindle is defined by the end caps and elsewhere by the interior curve of the torus (Fig. 3d).

To implement `containsPoint(p)`, note that the spindle is rotationally symmetric about the perpendicular vector. Thus, a plane K can be defined coplanar to p and the perpendicular vector of the torus. In K , p is inside the spindle only if it is inside the rectangle that defines the rotational cross section of the cylinder and also outside the circle that defines the rotational cross section of the torus.

`intersectSegment(s)` is computed by first setting up the calculation by translating the center of the spindle to the origin and rotating its axis to align it with the x axis. s is translated and rotated with it. We can describe the torus aligned to the x axis as

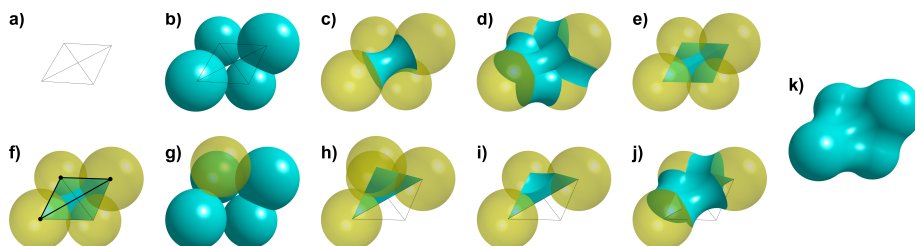
$$(x^2 + y^2 + z^2 + R^2 - r^2)^2 - 4R^2(y^2 + z^2) = 0$$

where R is the major radius, and r is the minor radius of the torus. In the torus equation, we substitute x , y and z with the line expressions $x_0 + td_x$, $y_0 + td_y$, and $z_0 + td_z$, where x_0, y_0, z_0 are segment starting points, and t parameterizes the line containing the line segment. The result of this substitution is a quartic equation on t , and roots of the equation will be parameters on the segment at points of intersection between the segment and the torus. We converted this equation into a monic quartic using Maxima, a computer algebra system [25].

To find the roots of this equation, we produce the Frobenius companion matrix of this quartic polynomial. The roots are the eigenvalues of this matrix. Here, complex eigenvalues will correspond to nonexistent points of intersection between the segment and the torus while real eigenvalues correspond to intersection points on the torus. We find these intersection points and eliminate any intersections that are outside of the cylinder. Separately, we also find intersections with the end caps of the spindle, treating them first as infinite planes and then determining if the intersection point is within the circle on the plane. Intersections between the segment and the endcaps or between the segment and the torus are returned as intervals where the segment is inside the spindle.

`findStartingCubes()` is implemented by first generating the segment between the endcap centers. The lattice cube containing one center is identified, and if it is not a surface cube, the adjoining cube, through whose face which the segment passes, is identified as the next cube to examine. This process is repeated along the segment until the segment ends at the other endcap center or a surface cube has been found. In the case where the spindle is broken (Fig. 3d), two segments are generated, with the same process performed on each segment starting at the center of each endcap.

Molecular Solids. pClay generates molecular solids from unions and differences of solids, such as spheres, spindles, and tetrahedra, that are positioned with the power diagram [1, 10]. This approach follows the classic methods for generating molecular surfaces, such as CASTp [32], MSMS [31], GRASP2 [30], which also use power diagrams or similar constructs like alpha shapes. For this reason, we paraphrase our approach here, expanding on points that differ from classic methods. As in the earlier methods, our approach represents water molecules as solvent spheres, which can be of any given radius. By calling basic functions from simpler primitives, pClay achieves an efficient implementation of the basic functions for the entire molecular solid without describing it as a CSG operation of many individual primitives.



■ **Figure 4** Molecular Surface Construction. a) Dual graph of a power diagram on four atoms (black lines, points). b) Sphere primitives from atoms (teal). c) Atoms (yellow) with one spindle (teal). d) Atoms with all spindles from edges of the dual graph. e) Tetrahedron primitive (teal). f) One triangle of the dual graph (black lines, dots). g) Solvent sphere (yellow) tangent to three atoms. h) New tetrahedron (teal) with corners in the center of the three atoms of the triangle and the solvent sphere. i) Cup region inside the new tetrahedron (teal). j) Cup, shown with three adjacent spindles (teal) and three atoms of the triangle (yellow). k) Finished molecular solid.

We begin with an input file from the Protein Data Bank (PDB). Using atomic coordinates and Van der Waals radii for each atom, we first compute a power diagram with REGTET [3]. The power diagram divides three dimensional space into cells corresponding to each atom of the input. The size of a cell relates to the Van der Waals radius of the atom, through the power function. Using the power diagram, we construct a topologically dual geometric graph (Fig. 4a), which has a vertex at the center of each atom and an edge between any vertices that correspond to adjacent cells. This *dual graph* defines the location of the primitives that will comprise the molecular solid. In sequential stages, we generate all primitives of the same type in parallel, starting with sphere primitives, then spindles, tetrahedra, and so on.

At every vertex of the dual graph, we create sphere primitives with the appropriate Van der Waals radius of each atom (Fig. 4b). Next, we examine every edge on the dual graph and generate a spindle between the atoms on at the endpoint of each edge, except for overlong edges that are longer than the sum of Van set Waals radius of the endpoint atoms and the diameter of the solvent sphere (Fig. 4c,d). Once all spindles are completed, we identify all tetrahedra in the dual graph that lack an overlong edge and we generate a tetrahedron primitive for each one (Fig. 4e).

Next, we identify triangles on the dual graph that are not between two tetrahedra (Fig. 4f). These triangles define triplets of atoms that may be on the molecular surface. To determine whether the atoms are on the surface, we place a solvent sphere tangent to all three atoms (Fig. 4g). If the solvent sphere does not collide with any other atoms, we create a *negsphere*: a sphere primitive in the tangent location in the same size as the solvent that describes a region of the solvent outside the molecular surface. We also generate a tetrahedron with corners on the triangle and at the center of the negsphere (Fig. 4h). The region inside this tetrahedron and outside the negsphere is both inside the solvent excluded

region and not occupied by spindles or atoms or other tetrahedra. We call this concave subset of a tetrahedron a *cup* (Fig. 4i), and describe cups as a negsphere-tetrahedron pair. The concave surface of the cup is continuous with the three adjacent spindles and atoms (Fig. 4j). Once all triangles that are not between two tetrahedra have been examined for the presence of a cup, the combination of spheres, spindles, tetrahedra and negspheres form a molecular solid (Fig. 4k).

To support the three basic functions, we store all of these primitives in a data structure for rapid range-based lookup. First, we generate a bounding box for each primitive. Next, we generate a lattice of cubes, where each cube is 2 angstroms on a side. Finally, we associate each primitive with all lattice cubes that intersect its bounding box. These associations act as a hashing function that enables us to rapidly identify any primitives nearby a given cube in the lattice. Since real molecules have finite atomic density, and since primitives are constructed from atoms and between atoms, the number of primitives associated with any cube is finite. As a result, a hashing function based on the lattice achieves algorithmically constant time lookup of nearby primitives.

For `containsPoint(p)`, given a point p , if p is outside the coarse lattice, then we immediately return false, because p must be outside the molecular surface. If not, we determine which cube c of the coarse lattice contains p . Next, we identify all primitives associated with c . We use the `containsPoint()` function of each associated primitive to determine if p is inside the primitive. If p is inside a `negSphere`, then p is outside the molecular surface. if p is inside any other primitives, then p is considered inside the molecular surface. If p is not inside any primitives, it is outside.

For `intersectSegment(s)`, given a segment s , we generate a list of cubes C that contain some interval of s . Next, we generate a list of primitives P associated with the cubes in C . We then query each primitive p in the list P for an interval of intersection between p and s using the `intersectSegment()` method of each primitive. The output intervals generated are the union of the intervals in tetrahedra, spindles and spheres minus the union of intervals inside negspheres.

For `findStartingCubes(l)`, during the construction of the molecular solid, we record the points of tangency between all negspheres and atom spheres. For each of these points, we identify the lattice cubes of l that contain them. We also generate starting cubes from all spindles and isolated spheres in the protein structure, calling `findStartingCubes()` on each of these primitives. From these cubes, we return only cubes that exhibit at least one corner inside and at least one corner outside the molecular solid.

CSG Operations. CSG operation nodes are non-leaf nodes that represent the outcome of a CSG operation on its operand nodes. They fulfill the three basic functions by calling on its operand nodes, which we refer to as A and B in the text below.

Given a point p , the CSG union returns true only when `containsPoint(p)` returns true on at least one operand. The CSG intersection returns true only when `containsPoint(p)` returns true on both operands. The CSG difference between A and B returns true only when A .`containsPoint(p)` is true and B .`containsPoint(p)` is false.

For a given segment s , `intersectSegment(s)` on any CSG operation calls `intersectSegment(s)` on operands A and B , generating intervals a and b . When run on a CSG Union, Intersection or Difference, the output is, respectively, the union, intersection, or difference of a and b .

Given a cubic lattice l , calling `getSurfaceCubes(l)` on a CSG union, intersection, or difference returns the setwise union of cubes returned by calling A .`getSurfaceCubes()` and B .`getSurfaceCubes()`. We always return a union of cubes because examining the union

of cubes can avoid circumstances where a disconnected region in the final solid is lost. Performance profiling revealed that considering the union of all cubes is a minor cost in overall performance, except in artificially constructed cases that create many irrelevant cubes.

Implementation Details. pClay is implemented in C and C++. Interprocess communication was built with Intel's Threading Building Blocks library. A C wrapper connects REGTET [3] to pClay. Benchmarks were run on a dual Xeon E5-2609 system with 8 cores at 2.5 Ghz and 32GB of ram and on a Xeon Phi 7120P with 61 cores at 1.2 Ghz and 16GB ram.

Datasets Used. *Dataset A* is 100 nonredundant pdb structures from VAST [24], with BLAST p-value cutoff $10e-7$. *Dataset B* is 30 spheres, spindles and tetrahedra randomly generated in a cube with 10\AA sides. *Dataset C* is 14 binding cavities from a nonredundant subset of the trypsins (1a0j,1aks,1ane,1aq7,1bzx,1fn8,1hrw,1trn,2eek,2f91), chymotrypsins (1eq9,8gch) and elastases (1b0e,1elt). More info: <http://www.cse.lehigh.edu/~chen/papers/WABI2019/appendix.pdf>

3 Experimental Results

3.1 Accuracy of molecular solid generation

We compared the surfaces of molecular solids from pClay to surfaces made with the trollbase library, an established tool for molecular surface generation used in GRASP2 [30], VASP [8], VASP-E [5], and MarkUs [13]. pClay surfaces were created at 0.25\AA resolution to yield a similar number of triangles and thus a fairer comparison. Using proteins from dataset A, surfaces generated by pClay had an average of 197,718.54 points. From these points, we measured the distance to the closest point on the surface generated by trollbase for the same protein. That distance averaged 0.00383\AA (sd 0.0004\AA , max $.22024\text{\AA}$). This tiny deviation, on surfaces built from thousands of atoms, demonstrates the accuracy of pClay surfaces.

3.2 Performance Comparison and Parallel Scaling

To our knowledge, VASP [5] is the only existing algorithm for comparing molecular solids using CSG. It lacks exact primitives or parallelism, but it can identify steric elements of protein structure that control specificity [8, 6, 14]. We compared the performance of pClay and VASP on the same CSG operations using Xeon (CPU) and Xeon Phi (PHI) processors.

Random Primitives. First, we compared CSG performance on the union of primitives in dataset B, generating mesh outputs at resolutions 1.0\AA , $.5\text{\AA}$, $.25\text{\AA}$ and $.125\text{\AA}$. Since VASP does not use primitives, it was provided triangle meshes of identical primitives. All CSG trees were balanced, but imbalanced trees had nearly identical runtimes (not shown for brevity).

On one CPU core, pClay required .113 seconds to compute the union at 1.0\AA resolution. 9.492 seconds were required to compute the same union at $.125\text{\AA}$ resolution (Fig. 5a). Increasing to 8 threads, runtime dropped to .03 seconds for unions at 1.0\AA resolution, and 1.465 seconds to at $.125\text{\AA}$. In contrast, single-threaded VASP required 3 seconds to compute the same union at 1.0\AA , and 64 at $.125\text{\AA}$. pClay far outperformed VASP on one thread.

On 8, 16, 32, and 60 PHI cores, which are slower than CPU cores, runtimes exhibited sublinear improvement (Fig. 5b). Runtimes on coarser resolutions improved less than for finer resolutions. The difference in parallel speedup (Fig. 5c) arises from small problem

sizes at coarse resolutions, where communications and setup outweigh the advantages of parallelism.

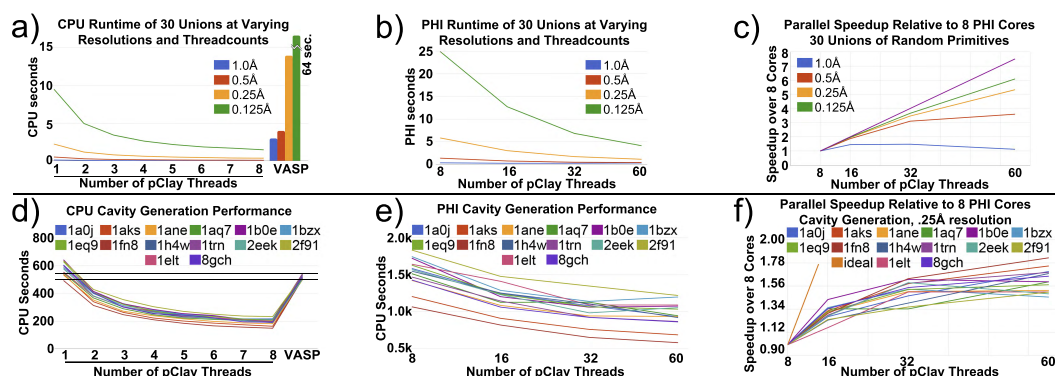


Figure 5 a) Time to compute the union of 30 random primitives at varying resolutions and CPU cores. VASP performance (single threaded) is shown in vertical bars. b) Time spent to compute the unions on PHI cores. c) Parallel speedup on PHI cores. d) Time spent for pClay to produce several binding cavities on CPU cores, compared to single-core VASP. e) Time to produce the same cavities on PHI cores. f) Parallel speedup of pClay in cavity production on varying PHI cores.

Binding Cavities. We also tested pClay by generating binding cavities, using the method from Fig. 1, on dataset C. While pClay is capable of much finer resolutions, all cavities were generated at $.25\text{\AA}$, the practical resolution limit for VASP. Figure 5d plots cavity generation times for these cavities. pClay required between 493 and 643 seconds on one CPU core, and between 149 and 233 seconds on 8 cores. Single threaded VASP required between 499 and 538 seconds to perform the same work. Single threaded, pClay was slightly slower than VASP, but much faster when adding a second core, and faster still when adding more.

Cavity generation was also run on 8, 16, 32, and 60 Xeon Phi cores. Runtimes on PHI cores are slower than on CPU cores because PHI cores have slower clock speeds. Runtimes fell slowly as threads increased (Fig. 5e). Substantial increases in the number of PHI cores resulted in only modest improvements in runtimes (Fig. 5f). This result contrasted from those performed on CPU cores, where performance improved substantially with increases in parallelism. These results point to bottlenecks in the PHI architecture affected by cavity generation, which is more data intensive than unions of random primitives.

3.3 Evaluating pClay on Existing Applications

The added precision of pClay creates several new applications. We demonstrate one such application by producing training data for VASP-S, a statistical model for detecting differences in ligand binding specificity with steric causes [6]. VASP-S is trained on the volumes of individual CSG differences computed from cavities with the same binding preference. This training enables VASP-S to estimate the probability (the p -value) that two given cavities have similar binding preferences. If p is lower than a threshold α , VASP-S rejects the hypothesis that two cavities have similar binding preferences, and predicts that they are different.

We hypothesized that training the VASP-S model with data generated at finer resolutions will produce more accurate predictions than a VASP-S model trained with coarser data. To test this hypothesis, we used cavities from the trypsins, which prefer to bind positively charged amino acids. These cavities contrast from those of the chymotrypsins and the elastases, which prefer large aromatics or small hydrophobics, respectively. Three training

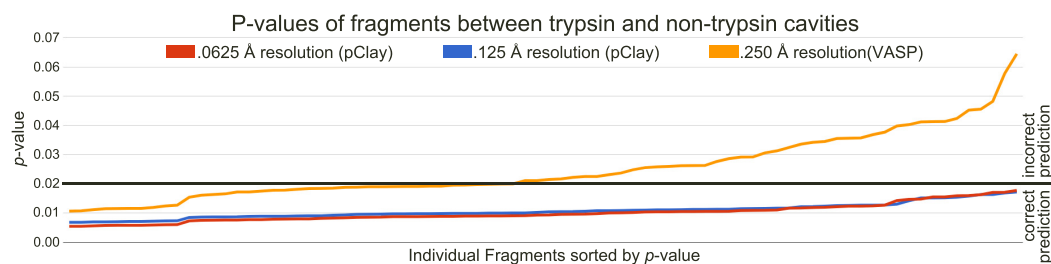


Figure 6 The p -value of the largest fragment between every trypsin-elastase and trypsin-chymotrypsin pair in Dataset C, estimated with training data generated at several resolutions (red, blue, orange lines). Fragments are sorted in ascending p -values along the horizontal axis. The black line indicates the α threshold of 0.02, below which we predict that the fragment is representative of proteins with different binding preferences. The finer-resolution training data, made possible with pClay, yielded more accurate predictions.

sets were constructed from these cavities by generating all possible CSG differences between all pairs of tryptins. VASP was used to produce a copy of the training set at 0.25 Å resolution and pClay was used to produce the same set at resolutions of 0.125 Å and 0.0625 Å. We then computed CSG differences between every trypsin and every nontrypsin at these resolutions. Finally, we estimate the p -value of the largest CSG difference between every trypsin and every non-trypsin in Dataset C, at all three resolutions (Fig. 6). We expect VASP-S to produce a low p -value on these CSG differences.

Using the conservative α threshold of 2%, when trained at 0.25 Å resolution, VASP-S predicts that 43 of the 81 CSG differences between trypsin and non-trypsin cavities indicate binding preferences. This discrepancy indicates 38 false negative predictions, where VASP-S incorrectly overlooked cavities with different binding preferences. However, when trained at 0.125 Å or 0.0625 Å resolution, VASP-S correctly predicts that all CSG differences were from cavities with different binding preferences, a 0% false negative rate. These results demonstrate that pClay can provide superior precision, ensuring that existing aggregate methods do not lose accuracy by overlooking useful predictions.

4 Discussion

We have presented pClay, the first parallel algorithm for performing CSG analysis of protein structures at arbitrarily high resolutions, up to machine precision. It leverages mathematically exact primitives that can be assembled into molecular solids and parallel depth first search to compute CSG operations with multiple threads.

Molecular solids from pClay are nearly identical to molecular surfaces generated by existing, widely used software. At hundreds of thousands of positions, pClay surfaces differed from surfaces generated by existing methods by only thousandths of an angstrom. While existing surface methods have generally been validated by visual examination, this exhaustive comparison sets a new standard for validation. Surface generation stresses the algorithms that underpin CSG operations, illustrating that pClay is making accurate comparisons.

We showed that pClay performs CSG operations efficiently on both artificial and realistic data. Evaluating the method on both Xeon CPU and Xeon Phi architectures, pClay exhibited scalable multithreaded performance on all tests, though scaling was modest on the Xeon Phi for cavity generation. These results show that parallelism can drive both efficiency and precision for the comparison of protein structures.

Finally, we showed how the precision of pClay can advance existing methods by training a statistical classifier to distinguish elements of protein structures that have a steric influence on binding specificity. pClay provided training data to the classifier that was more precise than what could have been provided by existing methods, enabling more accurate estimates of statistical significance, and ultimately a total elimination of false negative predictions.

These capabilities enable applications in the detection and explanation of structural features that influence binding preferences. For example, the statistical model tested here finds elements of protein structures that could have a steric influence on specificity, thereby generating an explanation based on a steric mechanism that typically requires human expertise. As high throughput technologies increasingly reveal the ways in which disease proteins can vary, pClay is a glimpse into a new space of techniques that can use protein variants to supplement human experience in deciphering the structural mechanisms of molecular recognition.

References

- 1 F Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.
- 2 CL Bajaj, V Pascucci, A Shamir, RJ Holt, and AN Netravali. Dynamic maintenance and visualization of molecular surfaces. *Discrete Applied Mathematics*, 127(1):23–51, 2003.
- 3 J Bernal. REGTET: A program for computing regular tetrahedralizations. In *International Conference on Computational Science*, pages 629–632. Springer, 2001.
- 4 M Brylinski and J Skolnick. A threading-based method (FINDSITE) for ligand-binding site prediction and functional annotation. *P Natl Acad Sci USA*, 105(1):129–134, 2008.
- 5 BY Chen. VASP-E: Specificity annotation with a volumetric analysis of electrostatic isopotentials. *PLoS computational biology*, 10(8):e1003792, 2014.
- 6 BY Chen and S Bandyopadhyay. VASP-S: A volumetric analysis and statistical model for predicting steric influences on protein-ligand binding specificity. In *IEEE Int Conf Bioinformatics Biomed 2011*, pages 22–29. IEEE, 2011.
- 7 BY Chen, VY Fofanov, DH Bryant, BD Dodson, DM Kristensen, AM Lisewski, M Kimmel, O Lichtarge, and LE Kavvaki. The MASH pipeline for protein function prediction and an algorithm for the geometric refinement of 3D motifs. *J Comput Biol*, 14(6):791–816, 2007.
- 8 BY Chen and B Honig. VASP: a volumetric analysis of surface properties yields insights into protein-ligand binding specificity. *PLoS computational biology*, 6(8):e1000881, 2010.
- 9 ML Connolly. Analytical molecular surface calculation. *Journal of applied crystallography*, 16(5):548–558, 1983.
- 10 Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994.
- 11 L Ellingson and J Zhang. Protein surface matching by combining local and global geometric information. *PLoS one*, 7(7):e40540, 2012.
- 12 F Ferre, G Ausiello, A Zanzoni, and M Helmer-Citterich. Functional annotation by identification of local surface similarities: a novel tool for structural genomics. *BMC Bioinf*, 6(1):194, 2005.
- 13 M Fischer, QC Zhang, F Dey, BY Chen, B Honig, and D Petrey. MarkUs: a server to navigate sequence–structure–function space. *Nucleic acids research*, 39(suppl_2):W357–W361, 2011.
- 14 BG Godshall and BY Chen. Improving accuracy in binding site comparison with homology modeling. In *IEEE Int Conf Bioinformatics Biomed 2012*, pages 662–669. IEEE, 2012.
- 15 L He, F Vandin, G Pandurangan, and C Bailey-Kellogg. Ballast: a ball-based algorithm for structural motifs. *Journal of Computational Biology*, 20(2):137–151, 2013.
- 16 C-L Hung and Y-L Lin. Implementation of a parallel protein structure alignment service on cloud. *International journal of genomics*, 2013, 2013.
- 17 T Ju, F Losasso, S Schaefer, and J Warren. Dual contouring of hermite data. In *ACM transactions on graphics (TOG)*, volume 21 (3), pages 339–346. ACM, 2002.

- 18 F Kaiser, A Eisold, and D Labudde. A novel algorithm for enhanced structural motif matching in proteins. *Journal of Computational Biology*, 22(7):698–713, 2015.
- 19 K Kinoshita, Y Murakami, and H Nakamura. eF-seek: prediction of the functional sites of proteins by searching for similar electrostatic potential and molecular surface shape. *Nucleic acids research*, 35(suppl_2):W398–W402, 2007.
- 20 J Konc, M Depolli, R Trobec, K Rozman, and D Janežič. Parallel-ProBiS: Fast parallel algorithm for local structural comparison of protein structures and binding sites. *Journal of computational chemistry*, 33(27):2199–2203, 2012.
- 21 B Lee and FM Richards. The interpretation of protein structures: estimation of static accessibility. *Journal of molecular biology*, 55(3):379–IN4, 1971.
- 22 Y Liu, K Shah, F Yang, L Witucki, and KM Shokat. A molecular gate which controls unnatural ATP analogue recognition by the tyrosine kinase v-Src. *Bioorg Med Chem*, 6(8):1219–1226, 1998.
- 23 WE Lorensen and HE Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *ACM siggraph computer graphics*, volume 21 (4), pages 163–169. ACM, 1987.
- 24 Thomas Madej, Christopher J Lanczycki, Dachuan Zhang, Paul A Thiessen, RC Geer, A Marchler-Bauer, and SH Bryant. MMDB and VAST+: tracking structural similarities between macromolecular complexes. *Nucleic acids research*, 42(D1):D297–D303, 2013.
- 25 WA Martin and RJ Fateman. The MACSYMA system. In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 59–75. ACM, 1971.
- 26 NL Max and ED Getzoff. Spherical harmonic molecular surfaces. *IEEE Computer Graphics and Applications*, 8(4):42–50, 1988.
- 27 EC Meng, BJ Polacco, and PC Babbitt. 3D Motifs. In *From Protein Structure to Function with Bioinformatics*, pages 187–216. Springer, 2009.
- 28 RJ Morris, RJ Najmanovich, A Kahraman, and JM Thornton. Real spherical harmonic expansion coefficients as 3D shape descriptors for protein binding pocket and ligand comparisons. *Bioinformatics*, 21(10):2347–2355, 2005.
- 29 BE Nolan, E Levenson, and BY Chen. Influential Mutations in the SMAD4 Trimer Complex Can Be Detected from Disruptions of Electrostatic Complementarity. *Journal of Computational Biology*, 24(1):68–78, 2017.
- 30 D Petrey and B Honig. GRASP2: visualization, surface properties, and electrostatics of macromolecular structures and sequences. *Methods in enzymology*, 374:492–509, 2003.
- 31 MF Sanner, AJ Olson, and J-C Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.
- 32 W Tian, C Chen, and J Liang. CASTp 3.0: Computed Atlas of Surface Topography of Proteins and Beyond. *Biophysical Journal*, 114(3):50a, 2018.
- 33 J Venkateswaran, B Song, T Kahveci, and C Jermaine. TRIAL: A Tool for Finding Distant Structural Similarities. *IEEE/ACM Trans Comput Biol Bioinform*, 8(3):819–831, 2011.
- 34 Y Zhou, X-P Li, BY Chen, and NE Tumer. Ricin uses arginine 235 as an anchor residue to bind to P-proteins of the ribosomal stalk. *Scientific reports*, 7:42912, 2017.

Read Mapping on Genome Variation Graphs

Kavya Vaddadi

TCS Research, Hyderabad, India
kavya.vaddadi@tcs.com

Rajgopal Srinivasan

TCS Research, Hyderabad, India
rajgopal.srinivasan@tcs.com

Naveen Sivadasan

TCS Research, Hyderabad, India
naveen.sivadasan@tcs.com

Abstract

Genome variation graphs are natural candidates to represent a pangenome collection. In such graphs, common subsequences are encoded as vertices and the genomic variations are captured by introducing additional labeled vertices and directed edges. Unlike a linear reference, a reference graph allows a rich representation of the genomic diversities and avoids reference bias. We address the fundamental problem of mapping reads to genome variation graphs. We give a novel mapping algorithm V-MAP for efficient identification of small subgraph of the genome graph for optimal gapped alignment of the read. V-MAP creates space efficient index using locality sensitive minimizer signatures computed using a novel graph winnowing and graph embedding onto metric space for fast and accurate mapping. Experiments involving graph constructed from the 1000 Genomes data and using both real and simulated reads show that V-MAP is fast, memory efficient and can map short reads, as well as PacBio/Nanopore long reads with high accuracy. V-MAP performance was significantly better than the state-of-the-art, especially for long reads.

2012 ACM Subject Classification Computing methodologies → Combinatorial algorithms; Applied computing → Computational genomics

Keywords and phrases read mapping, pangenome, genome variation graphs, locality sensitive hashing

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.7

Acknowledgements Authors would like to thank the anonymous reviewers for their valuable comments. Authors would also like to acknowledge Kshitij Tayal for the initial implementation of the algorithm.

1 Introduction

Conventional genome analysis relies primarily on a linear reference. A pangenome reference collection, on the other hand, is a richer representation of the genomic diversity and suffer less from reference bias [26]. Genome variation graphs are natural candidates for representing the genomic variations in a pangenome collection [26, 28]. In genome variation graphs, common subsequences are encoded as vertices and the genomic variations are captured by additional labeled vertices and directed edges. Such pangenome based representations and analysis are becoming increasingly popular also due to several ongoing large scale population-wide sequencing projects worldwide. As a result, there is an increasing need for developing efficient genome analysis pipelines that can work with graph genomes in place of linear genomes [26, 28]. The non-linear graph structure poses additional challenges to even the fundamental problems of sequence alignment and read mapping [28].



© Kavya Vaddadi, Rajgopal Srinivasan, and Naveen Sivadasan;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 7; pp. 7:1–7:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work, we consider the problem of efficiently mapping reads to genome variation graphs. Performing gapped alignment of reads to the whole graph is prohibitively expensive due to the large graph size and its complex structure. To achieve high throughput, the mapping algorithms have to restrict the expensive gapped alignment to only small regions of the reference graph where the read can align optimally. Several high throughput mapping tools are available for linear references [20, 21, 22, 24, 4]. Alignment based mappers use techniques such as Burrows-Wheeler Transforms [3, 9, 11, 23, 20, 17] to construct a search index of the reference along with backtracking to compute target alignment. Seed-and-extend approaches [2, 8, 5] use seeds or short read fragments to identify the target region and extend this to compute the final gapped alignment. Hashing based approaches [14, 21, 22] create hash signatures of the reference for fast identification of the target region. In particular, using locality sensitive hash functions (LSH) such as minhash have gained popularity due to their robustness to small read errors and gaps [14]. For instance, Mashmap [14] combines LSH with sequence winnowing technique [30] to build space efficient index with probabilistic guarantees on the identified target regions for long reads. Some of the tools that are optimized for short reads are not well suited for long reads due to higher error rates and longer read length. BWA-MEM [20], GraphMap [34], MashMap [14], Minimap2 [22] are a few of the mappers that provide long read support.

Linear reference mappers can be extended to a pangenome by treating the pangenome as a collection of linear sequences [25, 37, 12, 36]. One drawback of this approach is the increasing space/time requirements with increasing number of sequences. A unified representation, on the other hand, is significantly more compact and generalizes well to the space of possible variations and recombinations. GenomeMapper[31], MuGI [7], RCSI [37], BWBBLE [12] are some of the recent examples of mapping short reads to genome collection. They work based on the seed-and-extend paradigm with restrictions on the read length and gaps.

Similar to the linear reference setting, for genome graphs, the mapping algorithm has to restrict the costly alignment to small regions of the graph where the read is likely to align optimally to achieve high throughput. The Variation Graph (VG) project provides a state-of-the-art read mapping tool VG Mapper (VG) [10] on genome variation graphs. VG can handle directed acyclic graphs as well as variation graphs with cycles. VG considers the de Bruijn graph associated with the input graph and constructs an exact search index of the de Bruijn graph using Burrows-Wheeler extensions (GCSA2) for graphs [33, 32]. The index is used to identify hits on the graph based on the input read. These hits are then clustered and the final alignment is performed on the subgraph corresponding to the largest cluster. The BWT indexing of the graph is highly compute-intensive and requires an enormous amount of RAM and disk space. Further, de Bruijn graph can introduce false positives as it can encode paths that are not present in the graph [32]. Similarly, isolated hits in the graph produced by the read hits can affect the overall performance. The adverse effect on mapping quality and efficiency can be pronounced in the case of long reads.

1.1 Our Results

We present an algorithm V-MAP for efficient identification of subgraph of the genome variation graph for optimal read alignment. In this work, we consider variation graphs that are directed acyclic. The read can be aligned with affine gaps to the identified subgraph using any of the existing graph-based gapped aligners [10, 16, 15, 18, 29]. V-MAP creates a space efficient index of the graph G by computing locality sensitive minimizer signatures of G using a novel graph winnowing technique and combining it with a graph embedding onto a metric space. Graph winnowing creates a compact index which reduces the signature look-ups and also isolated false positive hits. Additionally, graph embedding allows fast subgraph identification based on signature hits.

We performed mapping experiments using real and simulated short/long read datasets on the graph constructed from the 1000 genomes variation data [1]. The variation data contained about 85 million variations and the resulting variation graph consisted of about 3×10^8 vertices and 4×10^8 edges. The 1000 genomes variation graph is a directed acyclic graph. We compared the performance of V-MAP with the state-of-the-art graph-based read mapper VG [10] and the popular linear reference mappers BWA-MEM [20] and Minimap2 [22].

V-MAP index has ~ 20 GB size and is constructed in just 2 hours using 16GB RAM, which is significantly smaller than VG in terms of the time/resource requirements. In terms of the final alignment score, V-MAP achieved higher alignment scores for a significantly large fraction of long reads. The performance was also significantly better than the linear mappers. Also, V-MAP accuracy in identifying the target region were 96.6% and 99% for short and long reads respectively. V-MAP identifies the target subgraph in milliseconds, even for long reads. In contrast to the whole graph size, the subgraph sizes were significantly smaller and were proportional to the read lengths. This lead to a significant reduction in time for gapped alignment time of reads to target subgraphs. We also provide analytical bounds for the V-MAP index size and for the V-MAP path sampling approach while indexing dense graph regions.

2 Preliminaries

2.1 Notations

Let $G = (V, E, \ell)$ be a connected directed acyclic graph with vertex set V , edge set E and vertex labels given by $\ell(v)$. Edges in E are ordered pairs from $V \times V$. Let σ^+ denote the set of all sequences of one or more elements from an alphabet σ . Each member in σ^+ is called a σ -sequence. When σ is the set of nucleotides, the σ -sequences are the nucleotide sequences. For a vertex $v \in V$, its label $\ell(v) \in \sigma^+$ is a σ -sequence. For an ordered sequence $x = (x_1, x_2, \dots, x_m)$, $|x| = m$ denotes its length. The i th element of x is denoted by $x[i]$. For the sake of brevity, we also denote $\ell(v)$ simply as v and use $v[i]$ to denote $\ell(v)[i]$ when there is no ambiguity.

A directed path p of length r in G is denoted by the ordered sequence (u_1, \dots, u_r) of r vertices, where $u_i \in V$ and $(u_i, u_{i+1}) \in E$. We say that the path p starts at u_1 and ends at u_r . The σ -sequence corresponding to p is obtained by concatenating the labels $\ell(u_1), \dots, \ell(u_r)$ in the same order. For each vertex u in G , we associate a σ -path given by an ordered sequence of pairs of the form $(\langle u, 0 \rangle, \dots, \langle u, |\ell(u)| - 1 \rangle)$, where $\langle u, i \rangle$ denotes the pair of vertex u and the offset i to its label $\ell(u)$. We call each such vertex offset pair a *graph coordinate* of G or *coordinate* in short. A σ -path q in G is given by a sequence $(\langle u_1, i \rangle, \dots, \langle u_1, |\ell(u_1)| - 1 \rangle, \langle u_2, 0 \rangle, \dots, \langle u_2, |\ell(u_2)| - 1 \rangle, \dots, \langle u_{r-1}, 0 \rangle, \dots, \langle u_{r-1}, |\ell(u_{r-1})| - 1 \rangle, \langle u_r, 0 \rangle, \dots, \langle u_r, j \rangle)$, where u_1, \dots, u_r is a path in G and i and j are offsets to $\ell(u_1)$ and $\ell(u_r)$ respectively. Here, the σ -path q is composed of the suffix of u_1 starting at offset i followed by the complete σ -paths of vertices u_2, \dots, u_{r-1} and finally by a prefix of u_r ending at offset j . Clearly, the σ -path q has a corresponding σ -sequence say q' of the same length obtained by concatenating the corresponding label symbols along the σ -path in the same order. We say that G contains a σ -sequence if it corresponds to some σ -path in G . We also say that G contains a σ -path q (or a σ -sequence q') starting at the graph coordinate $\langle u_1, i \rangle$.

2.2 Sequence Winnowing

Given a σ -sequence s , let K_s denote the set of distinct k -mers in s . We assume random mapping of k -mers to distinct non-negative integers uniformly at random. For simplicity, the integer associated with a k -mer x is also denoted by x . Let $z(s)$ denote the *minimizer* k -mer, which is the minimum valued k -mer, in K_s . We denote by $h(s)$, the *minimizer offset* in s . That is, $h(s)$ is the offset of $z(s)$ in s . If the minimizer occurs at multiple offsets in s then $h(s)$ is defined as the rightmost offset. The function z is locality sensitive in the sense that two sequences r and s have the same minimizer with probability $|K_r \cap K_s|/|K_r \cup K_s|$, which is the Jaccard similarity of the underlying sets K_r and K_s [6].

The standard winnowing of a linear sequence s [30] extends $h(s)$ to a function $h_w(s)$ that maps s to a subset of distinct minimizer offsets from $\{0, \dots, |s| - 1\}$ by considering a moving window of some fixed size $w \leq |s|$ over s starting at offsets $0, \dots, |s| - w$. Let $s_0, s_1, \dots, s_{|s|-w}$ denote the corresponding subsequences. More precisely, $h_w(s)$ maps s to the set of distinct offsets from the set

$$\{i + h(s_i) \mid 0 \leq i \leq |s| - w\}$$

The function $h_w(s)$ creates a fingerprint of s as a sequence of minimizers with a bounded positional gap between consecutive minimizers. The minimizer sequence can be used for sequence similarity. The expected size of the minimizer set is upper bound by $2|s|/|w|$ [30]. In [14], it was shown empirically that the minimizers computed by h_w can be used to well approximate the Jaccard similarity between sequences and this allows efficient alignment of reads to linear reference databases with higher precision under certain read error models.

3 Indexing

In this section, we describe our approach for indexing G that allows efficient mapping of reads to G . For this, we extend the notion of sequence winnowing to graph winnowing. We combine the graph winnowing with a graph embedding to generate a space efficient index of G that allows fast read mapping. We also discuss approaches to improve indexing performance.

3.1 Graph Winnowing

We extend the above winnowing of linear sequences to graph G by considering σ -sequences in G . Let p be a σ -path in G where $|p| \geq w$ and let p' be its corresponding σ -sequence. We define

$$h(p) = p[h(p')] \quad \text{and} \quad h_w(p) = \{p[i] \mid i \in h_w(p')\}$$

That is, for a σ -path p , $h(p)$ maps p to the graph coordinate corresponding to the minimizer of p' and h_w maps p to the set of graph coordinates corresponding to the minimizer set obtained by winnowing p' .

Let S denote the set of all σ -paths in G each of length $\geq w$. Let $H = \cup_{p \in S} h_w(p)$ denote the set of all distinct minimizer coordinates in G obtained by winnowing all paths in S . Similarly, let $H_w = \cup_{p \in S_w} h_w(p)$ where S_w is the set of all σ -paths in G each of length exactly w . It is straightforward to see that $H = H_w$ because the minimizers obtained by winnowing any σ -path in G are already present in H_w . In graph winnowing, we therefore compute H_w .

3.2 Graph Embedding and Indexing Approach

In this section, we describe our indexing approach which combines the graph winnowing and graph embedding to compute the graph index. Let \mathbb{N} denote the set of natural numbers. We consider an embedding $\lambda : V \times \mathbb{N} \rightarrow Y$ of coordinates $\langle v, i \rangle$ in G to a target metric space Y . Let $d_\lambda(x_1, x_2) \in \mathbb{R}^{\geq 0}$ denote the distance between the embedding of coordinates x_1 and x_2 in Y , given by $\lambda(x_1)$ and $\lambda(x_2)$ respectively.

Let S_w denote the set of all σ -paths in G each of length w . We do graph winnowing on G and create the index as follows. Let p be a σ -path in S_w and let p' be its corresponding σ -sequence. Consider the set of minimizer locations $h_w(p')$ in p' obtained by winnowing of p' . For each offset i in $h_w(p')$, let k_i denote the corresponding minimizer k -mer. The graph coordinate of k_i is given by $p[i]$. Let T_G be a key value table (dictionary) where for each minimizer k -mer, the corresponding graph coordinate and its embedding are stored. That is, $T_G[k_i]$ stores the tuple $\langle p[i], \lambda(p[i]) \rangle$. Since k -mer k_i can be the minimizer for multiple paths from S_w , $T_G[k_i]$ stores all such tuples corresponding to k_i . The specific choice of embedding and the final structure of the V-MAP graph index are discussed later.

Creating an index in the above manner that combines graph winnowing along with graph embedding results in a space efficient index which at the same time supports fast querying for read mapping. To identify regions in the graph where the read could optimally align, winnowing is performed on the input read and its minimizers are looked up in T_G . The hits from T_G are then clustered in the embedded space to identify maximum density cluster. Though each read minimizer can occur at multiple graph coordinates, the minimizer coordinates belonging to target subgraph tend to cluster in the embedded space. After identifying this cluster, read minimizer hits in this cluster are back-projected to construct the pruned subgraph where the input read is finally aligned. Different choices of embedding allow fast identification of the cluster while also ensuring that the corresponding subgraph is sufficiently pruned. For instance, using an embedding $\lambda : V \times \mathbb{N} \rightarrow \mathbb{N}^d$, the target cluster can be identified by computing the maximum enclosing d -dimensional axis parallel box of bounded side lengths in \mathbb{N}^d , where the side length is related to the read length. In particular, this can be done efficiently for small d [27]. If t is the total number of minimizer elements for an input query read, then the computation can be done in $O(t \log t)$ time for $d = 1$ by scanning the elements in the sorted order (in the embedded space). Similar bounds are also known for $d = 2$ [27].

V-MAP specific Embedding and Index

V-MAP uses an embedding $\lambda : V \times \mathbb{N} \rightarrow \mathbb{N}$ defined as follows. Consider an auxiliary undirected edge weighted graph G' derived from G as follows. For each vertex v in G , we include two vertices v_1 and v_2 in G' which are connected by edge (v_1, v_2) with weight $|\ell(v)| - 1$. Also, for every edge (u, v) in G , an edge (u_2, v_1) with weight 1 is included in G' . If u is the vertex in G with zero in-degree, we designate u_1 as the *source vertex* in G' . In case of multiple vertices with zero in-degree, a dummy source vertex is included that connects to these vertices with zero weight edges. For a vertex v in G' , let $sp(v)$ denote the shortest path distance from the source vertex to v in G' . The embedding $\lambda(\langle v, i \rangle)$ for a coordinate $\langle v, i \rangle$ in G is defined with respect to the shortest path distances of the corresponding vertices v_1 and v_2 in G' as

$$\lambda(\langle v, i \rangle) = \min\{sp(v_1) + i, \quad sp(v_2) + |\ell(v)| - i - 1\}$$

7:6 Read Mapping on Genome Variation Graphs

Under the above embedding, any two coordinates u and v with $\lambda(v) \geq \lambda(u)$ satisfies that $\lambda(v) - \lambda(u)$ is upper bound by the length of the shortest σ -path that connects u and v in G if there is any such path. Thus, nearby minimizers tend to cluster under this embedding.

In summary, the V-MAP indexing winnows all the w length σ -paths in G and the resulting set of minimizer coordinates are embedded onto the integer line based on their shortest path distances (length of σ -paths) from a designated source vertex coordinate in an undirected version of G . These embeddings are stored against the minimizer values.

The V-MAP index is a key/value table I_G which is a trimmed version of the table T_G discussed earlier. For each minimizer k -mer r , only the embedding of its corresponding coordinates in G are stored in $I_G[r]$, whereas $T_G[r]$ stores the respective coordinates along with their embedding. That is,

$$I_G[r] = \{\lambda(x) \mid T_G[r] \text{ contains the coordinate } x\}$$

The set $I_G[r]$ is a multiset in the sense that for each entry in $T_G[r]$, a separate, not necessarily distinct, element is present in $I_G[r]$.

V-MAP index also contains an adjacency representation of G sorted by the vertex embedding values. In this, the adjacency list for each vertex is also maintained as a sorted skip list or sorted array. This allows efficient extraction of the subgraph identified by V-MAP in order to compute the optimal gapped alignment of the read to the subgraph.

Dynamic Programming for Indexing

Enumerating all w length σ -paths in G for graph winnowing could be computationally expensive. Nevertheless, a dynamic programming heuristic can be used to winnow all w length σ -paths in G to reduce recomputations. The details are given in the Appendix (Section A).

4 Minimizer Density

We provide bounds on the expected number of minimizers created by V-MAP for an input graph G . Specifically, we suppose that the graph G is constructed from a collection $\mathcal{C} = \{s_1, \dots, s_N\}$ of N related random sequences each of length at most n . The sequences in \mathcal{C} are generated by a coupled random process, where, starting with a designated random sequence, random independent mutations with probability p are introduced to generate each of the remaining sequences in \mathcal{C} . The generation model is inspired by the mutation model approximations proposed in [35]. Due to the space limitation, the details are provided in Appendix B. For k -mer length k , we show the following theorem.

► **Theorem 1.** *The expected number of minimizers for G is upper bound by $\frac{2n}{w-k}(1 + p(N - 1))^{w+1}$ for window length w .*

To prove the above theorem, we first show that the expected number of w length σ -paths in G is bounded from above by $n(1 + p(N - 1))^w$. Combining this with a charging argument, which is an extension of [30] for linear random sequences to our graph setting, we obtain the theorem. The proofs are provided in Appendix B.

It follows that similar to the linear reference case, in the worst case, the number of minimizers on expectation, which is $O(\frac{n}{w} \exp(pwN))$, grows as $1/w$ fraction of the expected number of w length windows (σ -paths) in G .

4.1 Random Sampling

Dense regions of the graph can lead to increased time and space requirements for V-MAP indexing as the number of possible w length σ -paths in these regions could be very large. We use random sampling in such dense regions where, instead of considering all w length paths, M such paths are randomly sampled (random walk with replacement) where M is a parameter. This helps in reducing the time and space requirements and at the same time maintain high mapping accuracy. V-MAP indexing switches to random sampling mode for a coordinate when the total number of w length σ -paths starting from that coordinate exceeds M . Clearly, the final minimizer set of any graph under sampling is a subset of the minimizer set without sampling.

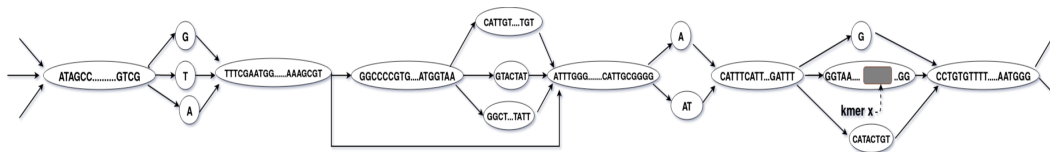
We analyze the effect of random sampling on the minimizer set. In particular, missing out minimizers due to sampling. Consider a k -mer x in a graph G . We analyze the expected number of w length σ -path samples in G that have x as a minimizer. Let $U = \{u_1, \dots, u_r\}$ be the set of coordinates in G such that x is reachable from them through a σ -path of length at most w . Let q be the probability that a random w length σ -path sample from a randomly chosen (uniformly at random) $u \in U$ contains x . We suppose the worst case that the number of w length σ -paths from each $u_i \in U$ exceeds M . In the analysis, we assume that the k -mers in any w length σ -path are distinct.

► **Lemma 2.** *The expected number of path samples each having x as the minimizer $\geq qM$.*

Proof. We observe that $|U| \geq w - k + 1$ as there are at least $w - k + 1$ coordinates lying on some w length σ -path leading to x in G . Consider any coordinate u_i from U . Let α_i denote the probability that a random w length σ -path sample from u_i contains the k -mer x . Then, the probability that a random sampled path contains x as minimizer is at least $\alpha_i / (w - k + 1)$. The total number of path samples (with repetition) from all of u_1, \dots, u_r is rM . In this sampling, let Z denote the the expected number of path samples where each of them have x as a minimizer. We have $Z \geq \frac{M}{(w-k+1)} \sum_{i=1}^r \alpha_i$. Recalling that $r \geq w - k + 1$, we obtain that $Z \geq \frac{M}{r} \sum_{i=1}^r \alpha_i = qM$ ◀

It is straightforward to see that, under no sampling, the expected number of w length σ -paths in G each having x as minimizer is $S / (w - k + 1)$, where S denote the total number of w length σ -paths from all the coordinates in U where each path contains the k -mer x .

From the above lemma, we infer that when a minimizer k -mer x of graph G is present in a dense region with high q value, then with a suitable sampling parameter M , sufficiently many paths under sampling also have x as a minimizer on expectation. For example, Figure 1 shows part of a commonly occurring dense subgraph topology which is a cascade of variations. The number of w length σ -paths from a fixed graph coordinate grows by a multiplicative factor depending on the number of cascade elements it can pass through. For the k -mer x as shown in the subgraph, there are several w length σ -paths from every coordinate in U . However, a random path sample from any coordinate in U will pass through the vertex containing x with probability $1/3$, implying a higher q value.



■ **Figure 1** Dense subgraph region with a cascade of variations.

If there is prior information available about the likelihood of certain paths over others in a graph region, such as the underlying haplotype information, edge weights can be introduced to capture non-uniform transition probabilities between vertices. Graph winnowing is easily generalizable to this setting resulting in biased path sampling for winnowing. Such a haplotype aware sampling can strike a balance between using haplotype prior and allowing newer paths that could arise due to recombinations. Further generalizations that allow a k -step Markov walk for path sampling can tilt the balance further in favour of haplotype prior.

5 Read Mapping

We describe the V-MAP read mapping algorithm in this section. For an input read s , the mapper aims to efficiently find a small subgraph G' of the graph G where s can align optimally. That is, G' has a path where s aligns optimally among all paths in G . The alignment of s to G' with affine gaps could be performed with any of the existing graph-based gapped aligners for sequences [10, 16, 15, 18, 29].

The minimizer set R of the input read s is first computed by applying the winnow hashing h_w on s . For each distinct minimizer $x \in R$, the entries stored at $I[x]$ in the index I are included in the hit set H . We recall that $I[x]$ contains the embedding in \mathbb{N} of all coordinates in G where x is a minimizer. Set $I[x]$ is assumed to be empty if x is not present as a minimizer in G . The final union H is a multiset. We now identify a maximum cardinality enclosing interval in \mathbb{N} of fixed width D that covers the maximum number of elements in H . This can be done easily in $O(|H| \log |H|)$ time by sorting H followed by a linear scan. The interval width D is fixed to be the length ℓ of the read s . Let h_l and h_r be the left end and right end respectively of the final enclosing interval. The final graph region identified by V-MAP is given by the subgraph induced by the vertices whose embedding lies in the interval $[h_l - \ell/2, h_r + \ell/2]$. The vertex set and the adjacency information of each vertex are stored in a sorted fashion based on the embedding of the start and the end coordinates of the vertices. This allows efficient extraction of the subgraph identified by V-MAP. Optimal gapped alignment of the read to the extracted subgraph can be computed using any graph alignment algorithm.

Reverse Complement

Reads from reverse complement strand can be handled using strand prediction in a straightforward way using techniques in [14] with a marginal increase in the index size. While indexing, each k -mer is transformed to its canonical form (either itself or its reverse complement based on their lexicographic ordering). For a minimizer k -mer x stored in the index I_G , we store an additional bit along with each of the coordinate embeddings stored in the set $I_G[x]$. The canonical bit records whether x appeared in the canonical form or not at that coordinate. While mapping, the canonical bit is computed for the read minimizers as well. Strand prediction is done based on the consensus of the canonical bits of the minimizers in the final cluster and bits of the read minimizers.

6 Experimental Results

We performed experiments to measure mapping accuracy and run time performance of V-MAP. We compared V-MAP with the state-of-the-art VG mapper [10] and popular linear reference mappers BWA-MEM [20] and Minimap2 [22].

Genome Variation Graph Construction

Human reference genome GRCh37 and the 1000 Genomes [1] variation data (phase 3) were used to construct a Human genome graph for our experiments. The variation data consisted of about 85 million variations. The VG graph construction tool [10] was used with default parameter setting to construct the *reference graph*. The reference graph consisted of about 3.2×10^8 vertices and 4.1×10^8 edges in total.

Read Set

The Illumina, PacBio and ONT real reads of *AshkenazimTrio HG002_NA24385_son* made available by Genome In A Bottle (GIAB) Consortium [39] were used. Around 10^4 reads of average length 117 formed the Illumina readset. For PacBio, $\sim 10^4$ reads of length $5k$ or above were used. For ONT, about 8000 reads were available in the GIAB dataset with read length 1000 or above.

Simulated reads were also used in the experiments. Reads were generated using read simulation tools from sequences arising from random paths in the reference graph. The ART tool [13] was used to generate Illumina short reads of lengths in the range 100 to 200 with its default error profile. PacBio and Nanopore (ONT) long reads of average lengths of $5k$, $10k$ and $50k$ were generated using ReadSim tool [19]. In total, 144×10^5 short reads and 3.8×10^5 long reads were generated. All reads in the read set are from forward strand. Strand prediction aspect is not considered in this work.

Implementation Details

V-MAP was implemented in C++. The index construction and mapping were performed on 200 GB RAM Linux machine with 48 threads and with 6TB HDD. Choices for the V-MAP parameters w and k were determined by a grid search based on the mapping accuracy on a separate set of random reads. Values $w = 35, k = 20$ were chosen for short reads and $w = 40, k = 20$ were chosen for long reads (PacBio/ONT). The random sampling parameter M for indexing was set to 30,000.

6.1 Mapping Accuracy

The final alignment score achieved by the candidate methods V-MAP, VG, BWA-MEM, and Minimap2 were measured for the real read set. In V-MAP, as in VG, the GSSW graph-based aligner [38, 10] was used to compute the gapped alignment score to the identified subgraph. The default alignment score parameters of BWA-MEM for gap open, gap extension, match, and mismatch were used across all tools. Table 2 gives the percentage of reads where each tool was a top scorer. That is, its score is no less than the score of other tools. The best value among BWA-MEM and Minimap2 is shown under the linear mapper column. The low percentage for linear mapper especially for long reads can be attributed to the fact that unlike the graph-based mappers, linear mappers have to work with only the linear reference and without any variant information. The graph mappers like V-MAP, on the other hand, achieve higher alignment score by working on a unified space of reference and variations encoded as a graph. The purpose of including the linear mappers in the comparison is only to bring out the advantage of using a pangenome reference over a linear reference and not to highlight the performance of any specific linear mapper. In Table 1, we show the average percentage gain in the alignment score achieved respectively by V-MAP and VG for reads where they were top scorers. As seen in Table 1, the percentage gain is pronounced in the case of long reads.

7:10 Read Mapping on Genome Variation Graphs

■ **Table 1** Average percentage gain in the alignment score for V-MAP and VG.

% Reads	Illumina			PacBio			ONT		
	V-MAP	VG	Linear mapper	V-MAP	VG	Linear mapper	V-MAP	VG	Linear mapper
Top scorer	95.28	99.79	87.98	81.15	52.01	3.2	80.73	62.43	3.43

■ **Table 2** Comparison of the candidate methods based on the final alignment scores achieved. The best performance among BWA-MEM and Minimap2 is shown under linear mapper.

Illumina		PacBio		ONT	
V-MAP	VG	V-MAP	VG	V-MAP	VG
0.0004	0.05	5.85	3.29	7.06	6.42

Using simulated reads, we calculated the mapping accuracy measured as the fraction of reads for which the subgraph identified by the mapper contained the path in the graph from which the read was generated. Table 3 shows the mapping accuracy of V-MAP and VG. V-MAP exhibits consistently superior performance for long reads with a much smaller index. In addition to higher accuracy, V-MAP also achieved considerably higher final alignment score. The average score difference was in excess of 680 for reads in the 5k to 10k range and 3590 for 10k to 50k range and 18300 for above 50k length reads. For short reads, the V-MAP alignment score was less than VG by 2 on average. In a related experiment of mapping 24k simulated reads of length 10k each from the graph with no sequencing errors, V-MAP achieved full alignment score for all but 2 reads in comparison to 96.8% of the reads for VG.

■ **Table 3** Mapping accuracies for different read classes.

Read class	100	150	200	Pac 5k	Pac 10k	Pac 50k	ONT 5k	ONT 10k	ONT 50k
V-MAP	95.22	96.21	98.49	96.90	99.04	99.63	99.47	99.57	99.95
VG	99.67	99.80	99.98	71.62	70.48	71.27	94.28	94.13	93.93

6.2 Mapping Performance

Table 4 gives the indexing performance, i.e., index construction time, index size, intermediate storage and RAM requirements for V-MAP and VG.

Table 5 gives the average read mapping time (single thread) for V-MAP and VG for short and long reads. The map time is the subgraph identification time and align time is the time taken for gapped alignment of the read to the identified subgraph using the GSSW graph-based aligner [38, 10]. The difference in alignment time between the two tools can be attributed to the smaller subgraphs identified by V-MAP for the alignment phase as shown in Figure 2.

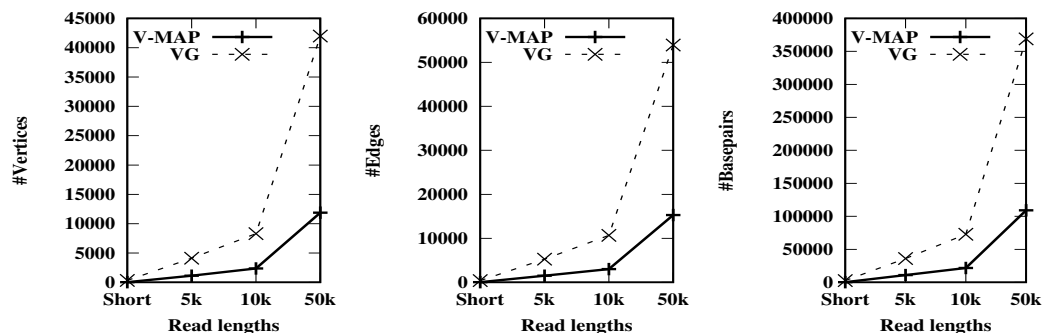
Figure 2 shows the average size statistics of identified subgraphs for short and long reads. The figure shows the number of edges, vertices and the total number of nucleotides in the vertex label sequences of the subgraph. The reduced subgraphs sizes significantly improve the gapped alignment time. Also, improved algorithms and implementations for gapped alignment on graphs can further improve the overall time.

■ **Table 4** Index construction and Mapping performance.

Index Parameters	V-MAP	VG
Construction Time	2 hours	36 hours
Intermediate storage	0	~ 2 TB
Index Size	21 GB (short reads) 18 GB (long reads)	80 GB
RAM	16 GB	200 GB
# indexing threads	4	32
Mapping RAM	27 GB	75 GB

■ **Table 5** Map and align timings. Map time is the subgraph identification time. Align time is the time to perform gapped alignment of the read to subgraph using GSSW.

Read length	Map time (ms)		Align time (ms)		Total time (ms)	
	V-MAP	VG	V-MAP	VG	V-MAP	VG
100	0.45	4.56	1.88	33	2.33	37.56
150	0.67	12.02	2.94	94.85	3.61	106.87
200	1.24	37.5	3.91	153.15	5.15	190.65
5k	5.84	144.16	540.28	4300.22	546.12	4444.39
10k	11.35	292.22	1831.34	9010.63	1842.69	9302.85
50k	63.63	1223.1	39,584.69	48,652.42	39,648.13	49,875.5



■ **Figure 2** Average size statistics of identified subgraphs for short and long reads. The figure shows the number of edges, number of vertices and the total number of nucleotides in the vertex label sequences of the subgraph.

7 Discussion

We present V-MAP for efficient identification of a subgraph of the input genome variation graph for optimal read alignment. Our tool exhibited significantly improved performance in comparison to the state-of-the-art. Improved algorithms and implementations of gapped aligners for graphs can further improve the overall performance significantly. Also, better graph embedding, say in \mathbb{N}^2 or other metric spaces including trees or other simpler graphs, could result in finer graph pruning and hence faster alignment.

References

- 1 1000Genome. 1000 Genome VCF. <ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502>, 2013. [Online; accessed 15-April-2017].
- 2 Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.
- 3 Michael Burrows and David Wheeler. A Block-Sorting Lossless Data Compression Algorithm. In *DIGITAL SRC RESEARCH REPORT*. Citeseer, 1994.
- 4 Stefan Canzar and Steven L Salzberg. Short read mapping: an algorithmic tour. *Proceedings of the IEEE*, 105(3):436–458, 2017.
- 5 Mark J Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC bioinformatics*, 13(1):238, 2012.
- 6 Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.
- 7 Agnieszka Danek, Sebastian Deorowicz, and Szymon Grabowski. Indexes of large genome collections on a PC. *PLoS one*, 9(10):e109384, 2014.
- 8 Arthur L Delcher, Adam Phillippy, Jane Carlton, and Steven L Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic acids research*, 30(11):2478–2483, 2002.
- 9 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.
- 10 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*, 2018.
- 11 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.
- 12 Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):i361–i370, 2013.
- 13 Weichun Huang, Leping Li, Jason R Myers, and Gabor T Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2011.
- 14 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 *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer, 2017.
- 15 Chirag Jain, Haowen Zhang, Yu Gao, and Srinivas Aluru. On the complexity of sequence to graph alignment. In *International Conference on Research in Computational Molecular Biology*, pages 85–100. Springer, 2019.
- 16 Vaddadi Naga Sai Kavya, Kshitij Tayal, Rajgopal Srinivasan, and Naveen Sivadasan. Sequence Alignment on Directed Graphs. *Journal of Computational Biology*, 2018.
- 17 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357, 2012.
- 18 Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.
- 19 Hayan Lee, James Gurtowski, Shinjae Yoo, Shoshana Marcus, W Richard McCombie, and Michael Schatz. Error correction and assembly complexity of single molecule sequencing reads. *BioRxiv*, page 006395, 2014.
- 20 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint*, 2013. [arXiv:1303.3997](https://arxiv.org/abs/1303.3997).

- 21 Heng Li. Minimap and Miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.
- 22 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 1:7, 2018.
- 23 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.
- 24 Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.
- 25 Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de Bruijn graphs. *BMC bioinformatics*, 17(1):237, 2016.
- 26 Tobias Marschall, et al. Computational pan-genomics: status, promises and challenges. *Briefings in bioinformatics*, 19(1):118–135, 2016.
- 27 Subhas C Nandy and Bhargab B Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29(8):45–61, 1995.
- 28 Benedict Paten, Adam M Novak, Jordan M Eizenga, and Erik Garrison. Genome graphs and the evolution of genome inference. *Genome research*, 27(5):665–676, 2017.
- 29 Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in $O(V + mE)$ time. *bioRxiv*, page 216127, 2017.
- 30 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.
- 31 Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome biology*, 10(9):R98, 2009.
- 32 Jouni Sirén. Indexing variation graphs. In *2017 Proceedings of the nineteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 13–27. SIAM, 2017.
- 33 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 (TCBB)*, 11(2):375–388, 2014.
- 34 Ivan Sović, Mile Šikić, Andreas Wilm, Shannon Nicole Fenlon, Swaine Chen, and Niranjan Nagarajan. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature communications*, 7:11307, 2016.
- 35 Matthew Stephens and Peter Donnelly. Inference in molecular population genetics. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(4):605–635, 2000.
- 36 Daniel Valenzuela and Veli Mäkinen. CHIC: a short read aligner for pan-genomic references. *bioRxiv*, page 178129, 2017.
- 37 Sebastian Wandelt, Johannes Starlinger, Marc Bux, and Ulf Leser. RCSI: Scalable similarity search in thousand (s) of genomes. *Proceedings of the VLDB Endowment*, 6(13):1534–1545, 2013.
- 38 Mengyao Zhao, Wan-Ping Lee, Erik P Garrison, and Gabor T Marth. SSW library: an SIMD Smith-Waterman C/C++ library for use in genomic applications. *PloS one*, 8(12), 2013.
- 39 Justin M Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E Mason, Noah Alexander, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific data*, 3:160025, 2016.

A Dynamic Programming for Indexing

Enumerating all w length σ -paths in G for graph winnowing could be computationally expensive. Nevertheless, a dynamic programming approach can be used to winnow all w length σ -paths in G to avoid recomputations. It is based on the observation that the minimizer set H_t of the set of all t length paths, each starting at a given graph coordinate $\langle u, i \rangle$, can be obtained recursively from the minimizer set H_{t-1} of the set of all $t-1$ length paths in G having the property that for each of these paths, its start coordinate, say $\langle v, j \rangle$, occurs immediately after $\langle u, i \rangle$ in some σ -path in G . As a consequence, each of these $t-1$ length σ -path p is extended to a t length σ -path p^+ by appending $\langle u, i \rangle$ to its beginning. Winnowing of p^+ is easily done by finding the minimum of the already computed minimizer of p and the leftmost k -mer of p^+ . The details of the DP formulation are given below.

Let $P_t(v, j, s)$ denote the set of all t length σ -paths in G each starting at the graph coordinate $\langle v, j \rangle$ and having the $k-1$ length sequence s as the prefix of its corresponding σ -sequence. For a σ -path $p \in P_t(v, j, s)$, associate a triplet $\langle u, i, r \rangle$ corresponding to the minimizer of p where $\langle u, i \rangle = h(p)$ denotes the minimizer coordinate and r denotes the minimizer k -mer. Corresponding to $P_t(v, j, s)$, let

$$H_t(v, j, s) = \{\langle u, i, r \rangle \text{ associated with a } p \in P_t(v, j, s)\}$$

We use $H_t(v, j, s)$ to also denote an underlying max heap of its triplets where the ordering is based on the k -mer values. In the following, we define a recurrence on $H_{t+1}(v, j, s)$ where s is a $k-1$ length prefix of some σ -sequence starting at $\langle v, j \rangle$. Clearly $\ell(v)[j] = s[0]$. Let s' be the $k-2$ length suffix of s . Define the set R of successor coordinates of the coordinate $\langle v, j \rangle$ as follows. If $j < |v| - 1$ then $R = \{\langle v, j+1 \rangle\}$. If $j = |v| - 1$ then $R = \{\langle u, 0 \rangle \mid u \in N_o(v)\}$, where $N_o(v)$, called the *out-neighbors* of v , denote the set of all vertices that have directed edges from v .

We now compute $H_{t+1}(v, j, s)$, where $|s| = k-1$, recursively from the H_t sets of the successor coordinates of $\langle v, j \rangle$ in R . For this, we define \mathcal{H}_t which is a set of non-empty sets $H_t(u, i, z)$, where $|z| = k-1$, as

$$\mathcal{H}_t = \{H_t(u, i, z) \mid \langle u, i \rangle \in R \text{ and } s' \text{ is prefix of } z\}$$

For each $H_t(u, i, z) \in \mathcal{H}_t$, let z^+ denote the k -mer obtained by concatenating $s[0]$ followed by z . We modify the max heap $H_t(u, i, z)$ to obtain a modified max heap (and the underlying triplet set) $H'_t(u, i, z)$ by removing all triplets from $H_t(u, i, z)$ whose k -mer values are greater than or equal to z^+ . Also, insert the triplet $\langle v, j, z^+ \rangle$ in $H'_t(u, i, z)$ if at least one triplet was removed from $H_t(u, i, z)$. Finally, the required $H_{t+1}(v, j, s)$ is obtained as

$$H_{t+1}(v, j, s) = \bigcup_{H_t(u, i, z) \in \mathcal{H}_t} H'_t(u, i, z)$$

The final set of triplets L to be indexed is given by the union of all non-empty $H_w(v, j, s)$ sets computed as above. For each triplet $\langle u, i, r \rangle$ in L , $\lambda(\langle u, i \rangle)$ is included in the index set $I_G[r]$.

The above recurrence is defined for $t > k$. For the base case $t = k$, we construct $H_k(v, j, s)$ for each coordinate $\langle v, j \rangle$ in G by including all triplets $\langle v, j, r \rangle$ in $H_k(v, j, s)$ where s is the $k-1$ length prefix of the k -mer r occurring at $\langle v, j \rangle$ in G .

In the above dynamic programming based minimizer computation, after round t of the dynamic programming updation, for each graph coordinate $\langle v, j \rangle$ and for each $k-1$ length string (σ -sequence) s starting from the coordinate $\langle v, j \rangle$, we maintain a separate heap $H_t(v, j, s)$. In the next round $t+1$, the new set $H_{t+1}(u, i, z)$, for a $k-1$ length string z

starting at coordinate $\langle u, i \rangle$, is computed from all sets $H_t(v, j, s)$ such that the $k - 2$ length suffix of s and $k - 2$ prefix of z are identical. Coordinates $\langle u, i \rangle$ and $\langle v, j \rangle$ should additionally satisfy that either (a) $u = v; j = i + 1$ or that (b) $j = 0; i = |\ell(u)| - 1; (u, v)$ is a directed edge in G . In this case, s and z give rise to a new k -mer r which is $z[0]$ followed by the string s . Now, $H_{t+1}(u, i, z)$ is the union of all minimizer entries of such $H_t(v, j, s)$ excluding the minimizers having value greater than r . A new entry for r is also included in $H_{t+1}(u, i, z)$ if at least one minimizer was excluded in the union.

From the complexity perspective, we see that after round t , the number of H_t sets (heaps) that are present is at most the total number of $k - 1$ length σ -paths in G . This is because, a separate $H_t(v, j, s)$ is maintained for each $k - 1$ length string starting from the graph coordinate $\langle v, j \rangle$. For a minimizer coordinate triplet $\langle y, j, x \rangle$, let $\beta(y, j, x)$ denote the total number of t length σ -paths in G each having x at coordinate $\langle y, j \rangle$ as its minimizer. Then, it is straightforward to verify that the minimizer triplet $\langle y, j, x \rangle$ can be present in at most $\beta(y, j, x)$ different H_t sets. Let set B denote the set of all distinct minimizer triplets obtained by winnowing t length σ -paths in G . Then, the sum total of the sizes of all H_t sets is given by $\sum_{r \in B} \beta(r)$. This in the worst case is upper bound as $\sum |H_t| \leq |B| \times d$ where d is the maximum number of t length σ -paths in G such that each give rise to the same minimizer triplet. That is, $d = \max_{r \in B} \beta(r)$. In other words, the blowup in space after round t is by a factor d in the worst case.

From the algorithm description, we can see that, in round $t + 1$, each $H_t(v, j, s)$ with $j > 0$ is scanned exactly once for obtaining a new H_{t+1} set. This is because, only $H_{t+1}(v, j - 1, \cdot)$ requires $H_t(v, j, s)$. However, set $H_t(v, 0, s)$ entries are scanned as many times as the in-degree of vertex v in G . Hence, the total number of triplet scanning performed in round $t + 1$ is upper bound by $\sum |H_t| + (\sum_{v \in V} |H_t(v, 0, \cdot)| \times \text{in-degree}(v))$, which can be crudely upper bound as $O(dm|B|)$, where m is the total number of edges in G .

In the subsequent section, we discuss random path sampling approach for improving the indexing performance in the dense graph regions when the winnowing is performed by scanning w length paths in G . This dynamic programming heuristic does not incorporate the path sampling strategy. The DP approach can be useful for winnowing non-dense graphs.

B Minimzer Density

We provide bounds on the expected number of minimizers created by V-MAP for an input graph G corresponding to a collection of random related sequences. Specifically, we suppose that the graph G is constructed from a collection $\mathcal{C} = \{s_1, \dots, s_N\}$ of N random related sequences each of length at most n . The sequences in \mathcal{C} are generated by a coupled random process described in the following, where, starting with a designated random sequence, random independent mutations with probability p are introduced to generate each of the remaining sequences in \mathcal{C} . This generation model is inspired by the mutation model approximations proposed in [35]. Sequence s_1 , also denoted as s_1^* , is designated as the *basis sequence* which is a random sequence where the nucleotide at each location is chosen independently and uniformly at random. Each remaining sequence s_k for $k \in \{2, \dots, N\}$ is generated independently as gapped variant of the basis sequence s_1 using the following coupled process. The basis sequence s_1^* is scanned sequentially from left to right for n steps. At each step t , the nucleotide present in s_1^* is appended to s_k independently with probability $1 - p$ for some fixed p . With probability p , a gap introduced in s_k either as a deletion from s_1^* (i.e., skip the nucleotide in s_1^*), a random nucleotide substitution in s_k or a nucleotide insertion in s_k . In the case of an insertion, the scan location in s_1^* does not change. It is straightforward to

verify that any given sequence from \mathcal{C} is a random sequence where the nucleotide at each location is chosen independently and uniformly at random. However, any two sequences from \mathcal{C} are not independent of each other. If the mutation and gap probabilities are unequal, let p denote the maximum of them in the following analysis.

We now consider a genome variation graph G constructed from \mathcal{C} . The above generation process gives a natural way to construct G . The complete basis sequence s_1^* is initially represented as a single labeled vertex. Each of the remaining coupled sequence s_k for $k \geq 2$ is incorporated by introducing alternative paths in G at the gap regions with respect to s_1^* . Specifically, a contiguous gap between s_k and s_1^* is handled by introducing a new vertex u that encodes the subsequence of s_k in this gap region. The new vertex u is connected as a bridge vertex parallel to the corresponding gap subsequence of s_1^* in G . For this, a vertex split is done at the begin and end locations of this gap in the respective vertices of the path that encodes the gap subsequence of s_1^* . After the split, there is a path in G that exactly encodes the gap subsequence of s_1^* and the new vertex u is connected as a bridge parallel to this path. If the gap region consists of only deletions along s_k , then the gap subsequence of s_k is empty. In this case, just a bridge edge is introduced across the gap sequence path of s_1^* .

In each step, while generating a new sequence, a gap occurs with probability p and each gap introduces a constant number of additional edges and vertices in G . Hence the expected number of vertices and edges in G is $O(npN)$. In order to bound the minimizer cardinality, first, we bound the expected number of w length σ -paths in G .

► **Lemma 3.** *The expected number of w length σ -paths is upper bound by $n(1 + p(N - 1))^w$.*

Proof. For the ease of exposition, we assume without loss of generality that each vertex of G has only single nucleotide label. If a vertex has a longer label, it can be replaced by a corresponding path of single nucleotide vertices. Let the vertices along the s_1 path in G be $U = \{u_1, \dots, u_n\}$ and let d_i be the out-degree of u_i . Let X denote the set of all w length σ -paths in G . Let X_i for $i \in \{1, \dots, n\}$ denote the subset of X where each σ -path x in it has u_i at location $x[m - 2]$. Let R denote the remaining paths from X not included in any of X_i . Consider the $m - 1$ length prefix $x[0, \dots, m - 2]$ of each σ -path x from an X_i . Let n_i be the number of distinct such prefixes from X_i . Clearly, $|X| \leq \sum n_i d_i + |R|$. We observe that n_i is independent of d_i and that $E(d_i)$ is approximated by $1 + E(\text{Binomial}(N - 1, p))$ because each of s_2, \dots, s_N can introduce an outgoing edge at u_i with probability p . Therefore, we obtain $E(|X|) \leq (1 + p(N - 1))E(\sum n_i + |R|)$. Clearly $\sum n_i + |R|$ is no more than the number of $w - 1$ length σ -paths in G . Here we observe that $w - 1$ length prefix of each path x from R is distinct as the out-degree of a non U vertex (vertex at $x[m - 2]$) is at most 1. Repeating the above argument and using the observation that the expected number of 1 length σ -paths (i.e., total nucleotides) in G is at most $n(1 + p(N - 1))$, we finally obtain $E(X) \leq n(1 + p(N - 1))^w$. ◀

The following Theorem provides a bound on the expected number of minimizers resulting from winnowing of G .

► **Theorem 4.** *The expected number of minimizers for G is upper bound by $\frac{2n}{w-k}(1 + p(N - 1))^{w+1}$ for winnow length w .*

Proof. We extend the charging argument in [30] for linear random sequences to our graph setting. In [30], for a linear sequence s , a w sized window W_i at coordinate i is charged, i.e., contributes a new minimizer k -mer say at $s[j]$, if W_i is the leftmost among the windows that overlap the k -mer at $s[j]$ and the k -mer is a minimizer in them. It was shown in [30] that this

corresponds to the event that in the $w + 1$ sized window W which is the union of W_{i-1} and W_i , the minimizer of W is given by either its first k -mer or its last k -mer. The probability of this to happen in a random sequence is $2/(w + 1)$. The total number of minimizers after winnowing an n length sequence s is then given by the total number of w length windows that are charged, which is $2n/(w + 1)$.

Consider any $w + 1$ length σ -path in G . Let c' denote the w length suffix window of c and let x and x' denote the (minimizer) k -mers at locations $h(c)$ and $h(c')$ respectively. We charge window c' if and only if x is either the first or the last k -mer in c . It is straightforward to verify that each minimizer in G is charged to at least one w length σ -path window in G . In the linear case, a minimizer is charged to exactly one window whereas the same minimizer could be charged to multiple windows in the case of graphs. This can only overestimate the desired count. Recalling the generation process of G , we observe that any given $w + 1$ length σ -path c in G encodes a random σ -sequence. Hence, its w length suffix c' is charged with probability $2/(w - k + 2)$. The graph G has an underlying topology from Ω which denote the space of all graph topologies that can arise from the graph creation process described earlier. Each topology in Ω has a unique encoding where each vertex u has an associated distinct triplet id of the form $\langle i, j, l \rangle$ denoting that u is uniquely associated with the l length subsequence of $s_i \in \mathcal{C}$ starting from offset j . For a given topology, the specific vertex labels are still random. It is clear from the generation process that for any fixed topology T , the σ -sequence for any given σ -path in T is a random sequence. Hence, if n_T denote the number of $w + 1$ length σ -paths for a topology T , a $w + 1$ length σ -path is charged in a graph with topology T with probability $2/(w - k + 2)$. Let D denote the number of charged $k + 1$ length σ -sequence in G . The total number of minimizers is upper bound by D . We have $E(D | T) = 2 \cdot n_T / (w - k + 2)$. Hence, we obtain

$$\begin{aligned} E(D) &= \sum_{T \in \Omega} E(D | T) \Pr(T) \\ &= \frac{2}{w - k + 2} \sum_{T \in \Omega} n_T \Pr(T) \leq \frac{2n}{w - k} (1 + p(N - 1))^{w+1} \end{aligned}$$

where the last inequality follows by applying Lemma 3 as $\sum_{T \in \Omega} n_T \Pr(T)$ is the expected number of $w + 1$ length σ -paths . \blacktriangleleft

Finding All Maximal Perfect Haplotype Blocks in Linear Time

Jarno Alanko 

Department of Computer Science, University of Helsinki, Finland
jarno.alanko@helsinki.fi

Hideo Bannai 

Department of Informatics, Kyushu University, Japan
bannai@inf.kyushu-u.ac.jp

Bastien Cazaux 

Department of Computer Science, University of Helsinki, Finland
bastien.cazaux@helsinki.fi

Pierre Peterlongo 

Univ. Rennes, Inria, CNRS, Irista, France
pierre.peterlongo@inria.fr

Jens Stoye 

Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany
jens.stoye@uni-bielefeld.de

Abstract

Recent large-scale community sequencing efforts allow at an unprecedented level of detail the identification of genomic regions that show signatures of natural selection. Traditional methods for identifying such regions from individuals' haplotype data, however, require excessive computing times and therefore are not applicable to current datasets. In 2019, Cunha *et al.* (Proceedings of BSB 2019) suggested the *maximal perfect haplotype block* as a very simple combinatorial pattern, forming the basis of a new method to perform rapid genome-wide selection scans. The algorithm they presented for identifying these blocks, however, had a worst-case running time quadratic in the genome length. It was posed as an open problem whether an optimal, linear-time algorithm exists. In this paper we give two algorithms that achieve this time bound, one conceptually very simple one using suffix trees and a second one using the positional Burrows-Wheeler Transform, that is very efficient also in practice.

2012 ACM Subject Classification Theory of computation → Pattern matching; Applied computing → Bioinformatics; Mathematics of computing → Combinatorial algorithms; Applied computing → Computational genomics

Keywords and phrases Population genomics, selection coefficient, haplotype block, positional Burrows-Wheeler Transform

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.8

Funding *Hideo Bannai*: JSPS KAKENHI Grant Number JP16H02783

Pierre Peterlongo: ANR Hydrogen ANR-14-CE23-0001

Acknowledgements We thank the organizers of DSB 2019 (dsb2019.gitlab.io) for giving us the opportunity to present earlier work in this area and start a discussion from which the present results originated. We would also like to thank Michel T. Henrichs for providing a script to convert VCF files to haplotype matrices and for assisting with the production of Figure 3.



© Jarno N. Alanko, Hideo Bannai, Bastien Cazaux, Pierre Peterlongo, and Jens Stoye; licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 8; pp. 8:1–8:9

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction and Background

As a result of the technological advances that went hand in hand with the genomics efforts of the last decades, today it is possible to experimentally obtain and study the genomes of large numbers of individuals, or even multiple samples from an individual. For instance, the *National Human Genome Research Institute* and the *European Bioinformatics Institute* have collected more than 3500 genome-wide association study publications in their *GWAS Catalog* [3].

Probably the most prominent example of large-scale sequencing projects is the *1000 Genomes Project* (now *International Genome Sample Resource*, IGSR), initiated with the goal of sequencing the genomes of more than one thousand human individuals to identify 95% of all genomic variants in the population with allele frequency of at least 1% (down toward 0.1% in coding regions). The final publications from phase 3 of the project report about genetic variations from more than 2,500 genomes [1, 12].

Recently, several countries announced large-scale national research programs to capture the diversity of their populations, while some of these efforts started already more than 20 years ago. Since 1996 Iceland's deCODE company is mining Icelanders' genetic and medical data for disease genes. In 2015, deCODE published insights gained from sequencing the whole genomes of 2,636 Icelanders [8]. *Genome of the Netherlands* (GoNL) is a whole genome sequencing project aiming to characterize DNA sequence variation in the Dutch population using a representative sample consisting of 250 trio families from all provinces in the Netherlands. In 2016, GoNL analysed whole genome sequencing data of 769 individuals and published a haplotype-resolved map of 1.9 million genome variants [10]. Similar projects have been established in larger scale in the UK: Following the *UK10K* project for identifying rare genetic variants in health and disease (2010–2013), *Genomics England* was set up in late 2012 to deliver the 100,000 Genomes Project [13]. This flagship project has by now sequenced 100,000 whole genomes from patients and their families, focusing on rare diseases, some common types of cancer, and infectious diseases. The scale of these projects is culminating in the US federal *Precision Medicine Initiative*, where the NIH is funding the *All of Us* research program¹ to analyze genetic information from more than 1 million American volunteers. Even more extreme suggestions go as far as to propose “to sequence the DNA of all life on Earth”².

The main motivation for the collection of these large and comprehensive data sets is the hope for a better understanding of genomic variation and how variants relate to health and disease, but basic research in evolution, population genetics, functional genomics and studies on demographic history can also profit enormously.

One important approach connecting evolution and functional genomics is the search for genomic regions under natural selection based on population data. The *selection coefficient* [7] is an established parameter quantifying the relative fitness of two genetic variants. Unfortunately, haplotype-based methods for estimating selection coefficients have not been designed with the massive genome data sets available today in mind, and may therefore take prohibitively long when applied to large-scale population data. In view of the large population sequencing efforts described above, methods are needed that – at similar sensitivity – scale to much higher dimensions.

¹ allofus.nih.gov

² Biologists propose to sequence the DNA of all life on Earth, by Elizabeth Pennisi. Science News, Feb. 24, 2017. <https://doi.org/10.1126/science.aa10824>.

Only recently a method for the fast computation of a genome-wide selection scan has been proposed that can be computed quickly even for large datasets [4]. The method is based on a very simple combinatorial string pattern, *maximal perfect haplotype blocks*. Although considerably faster than previous methods, the running time of the algorithm presented in that paper is not optimal, as it takes $O(kn^2)$ time in order to find all maximal perfect haplotype blocks in k genomes of length n each. This is sufficient to analyse individual human chromosomes on a laptop computer, for datasets of the size of the 1000 Genomes Project (1,000s of genomes and 1,000,000s of variations). However, with the larger datasets currently underway and with higher resolution it will not scale favourably. More efficient methods are therefore necessary and it was phrased as an open question whether there exists a linear-time algorithm to find all maximal perfect haplotype blocks.

In this paper we settle this open problem affirmatively. More specifically, after some basic definitions in Section 2 we present in Sections 3 and 4 two new algorithms for finding all maximal perfect haplotype blocks in optimal time. The latter of these two algorithms is then experimentally compared to the one from [4] in Section 5, proving its superiority in running time by a factor of about 5 and memory usage by up to two orders of magnitude for larger data sets. Section 6 concludes the paper.

2 Basic Definitions

The typical input to genome-wide selection studies is a set of haplotype-resolved genomes, or *haplotypes* for short. Clearly, for a given set of haplotypes only those sites are of interest where there is variation in the genomes. Therefore, formally, we consider as input to our methods a $k \times n$ *haplotype matrix* where each of the k rows corresponds to one haplotype and each of the n columns corresponds to one variable genetic site.

Most methods distinguish only between ancestral and derived allele, a consequence of the popular *infinite sites assumption* [11]. Therefore the entries in a haplotype matrix are often considered binary where the ancestral allele is encoded by 0 and the derived allele is encoded by 1. However, the computational problem and its solutions considered in this paper do not depend on this restriction and instead are applicable to any type of sequence over a constant-size alphabet Σ .

The concept of a maximal perfect haplotype block as defined in [4] is the following, where $S|_K$ denotes the elements of an ordered set S restricted to index set K :

► **Definition 1.** *Given k sequences $S = (s_1, \dots, s_k)$ of same length n (representing the rows of a haplotype matrix), a maximal perfect haplotype block is a triple (K, i, j) with $K \subseteq \{1, \dots, k\}$, $|K| \geq 2$ and $1 \leq i \leq j \leq n$ such that*

1. $s[i, j] = t[i, j]$ for all $s, t \in S|_K$ (equality),
2. $i = 1$ or $s[i - 1] \neq t[i - 1]$ for some $s, t \in S|_K$ (left-maximality),
3. $j = n$ or $s[j + 1] \neq t[j + 1]$ for some $s, t \in S|_K$ (right-maximality), and
4. $\nexists K' \subseteq \{1, \dots, k\}$ with $K \subset K'$ such that $s[i, j] = t[i, j]$ for all $s, t \in S|_{K'}$ (row-maximality).

Definition 1 is illustrated in Figure 1.

In [4] it was shown that the number of maximal perfect haplotype blocks is in $O(kn)$, while the algorithm presented there takes $O(kn^2)$ time to find all blocks. It is based on the observation that branching vertices in the trie T_p of the suffixes of the input sequences starting at position p correspond to right-maximal and row-maximal blocks, while left-maximality can be tested by comparing T_p and T_{p-1} . In the next two sections we show how this running time can be improved.

0	1	0	1	0	1	0	0
1	0	1	1	1	1	0	1
0	1	0	1	1	1	0	0

Figure 1 Illustration of Definition 1: a binary 3×8 haplotype matrix with three maximal perfect haplotype blocks $(\{1, 3\}, 1, 4)$, $(\{2, 3\}, 4, 7)$ and $(\{1, 2, 3\}, 6, 7)$ highlighted. (The example contains additional maximal perfect haplotype blocks that are not shown.)

3 Linear-Time Method I: Based on Suffix Trees

In this section, we present our first algorithm to find all maximal perfect haplotype blocks in linear time. This solution is purely theoretical, it would likely require large amounts of memory while being slow in practice. However, it demonstrates the connection to the concept of maximal repeats in strings. We recall from [9, Section 7.12] that a *maximal repeat* is a substring occurring at least twice in a string or a set of strings and such that it cannot be extended to the left or to the right without losing occurrences.

Let $\mathbb{S} = s_1\$1s_2\$2 \dots s_k\$k$, with the $\$i$ being k different characters absent from the original alphabet Σ . The key point is that any maximal perfect haplotype block in S is a maximal repeat in \mathbb{S} . The opposite is not true: In a maximal perfect haplotype block, all occurrences of the repeat are located at the same position of each sequence of S (equality condition in Definition 1), while this constraint does not exist for maximal repeats in \mathbb{S} .

Nevertheless, finding all maximal perfect haplotype blocks in S can be performed by computing all maximal repeats in \mathbb{S} , while keeping only those whose occurrences are located at the same positions over all s_i in which they occur. This can be done by performing the following procedure:

1. “Decorate” each sequence $s_i \in S$ to create $s_i^+ = \alpha_0 s_i[1] \alpha_1 s_i[2] \alpha_2 \dots s_i[n] \alpha_n$, where the *index characters* $\alpha_0, \alpha_1, \dots, \alpha_n$ are $n + 1$ symbols from an alphabet Σ' , disjoint from the original alphabet Σ .
2. Find in $\mathbb{S}^+ = s_1^+ \$1 s_2^+ \$2 \dots s_k^+ \$k$ all maximal repeats.
3. Any maximal repeat $r = \alpha_p r_1 \alpha_{p+1} r_2 \alpha_{p+2} \dots r_\ell \alpha_{p+\ell}$ in \mathbb{S}^+ with $\ell \geq 1$ corresponds to a maximal perfect haplotype block of length ℓ , starting at position $p + 1$ in the input sequences from S .

The key idea here is that the index characters impose that each maximal repeat occurrence starts at the same position in all sequences and, as a consequence, ensure that all occurrences occur in distinct sequences from S .

Hence any maximal repeat $r = \alpha_p r_1 \alpha_{p+1} \dots r_\ell \alpha_{p+\ell}$ defines a unique maximal perfect haplotype block $(K, p + 1, p + \ell)$. The value $|K|$ is the number of occurrences of r . Also the set K can be derived from occurrence positions of r in \mathbb{S}^+ , as any position in r corresponds to a unique position in \mathbb{S} . We prefer to omit useless technical details here.

The maximal repeat occurrences in \mathbb{S}^+ may be found using a suffix tree, constructed in time linear with respect to the size of the input data $O(kn)$, even for large integer alphabets [6], as we have here. The maximal repeat detection is also linear with the size of the input data [9, Section 7.12.1]. Therefore the overall time complexity is $O(kn)$.

4 Linear-Time Method II: Based on the Positional BWT

Here we present our second algorithm to find all maximal perfect haplotype blocks in linear time. It works by scanning the haplotype matrix column by column while maintaining the positional Burrows-Wheeler Transform (pBWT) [5] of the current column. For simplicity of

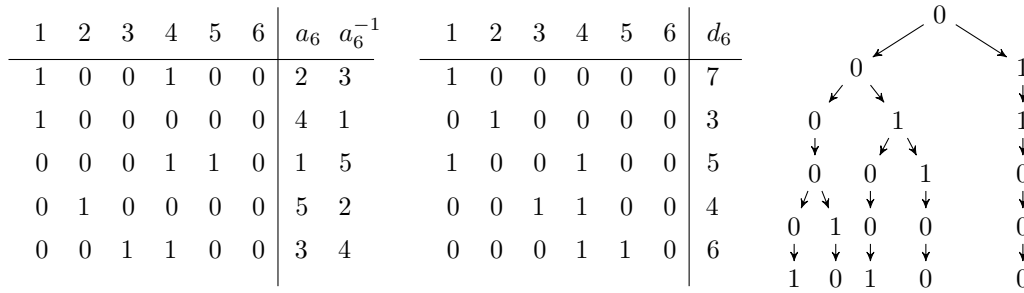


Figure 2 Available blocks. **Left:** an example of a haplotype matrix up to column 6 with the two arrays a_6 and a_6^{-1} on the right. **Center:** the colexicographically sorted rows and the array d_6 listed on the right. **Right:** the trie of the reverses of the rows of the matrix. For example, the block $(\{1, 2, 4, 5\}, 5, 6)$ is available because $a_6^{-1}(1) = 3, a_6^{-1}(2) = 1, a_6^{-1}(4) = 2, a_6^{-1}(5) = 4$ is the consecutive range $[x, y] = [1, 4]$, we have $d_6[r] \leq 5$ for all $r \in [1 + 1, 4]$ with $d_6[3] = 5$, and we have $x = 1$ and $d_6[4 + 1] = 6 > 5$. The repeat in the block is 00, and we see it is a branching node in the trie on the right.

presentation we assume that all rows of the haplotype matrix S are distinct. Recall that the pBWT of S consists of a pair of arrays for each column of S : For each $l, 1 \leq l \leq n$, we have arrays a_l and d_l of length k such that the array a_l is a permutation of $[1, k]$ with $S[a_l[1]][1..l] \leq \dots \leq S[a_l[k]][1..l]$ colexicographically (i.e. right-to-left lexicographically) and the array d_l is such that $d_l[1] = l + 1$ and for all $r, 1 < r \leq k$, we have $d_l[r] = 1 + \max\{j \in [1, l] : S[a_l[r]][j] \neq S[a_l[r-1]][j]\}$. Further let us denote by a_l^{-1} the inverse permutation of a_l . For readers familiar with string processing terminology, the arrays a_l and a_l^{-1} are analogous to the suffix array and the inverse suffix array, respectively, while the arrays d_l are analogous to the LCP array. We note that the term pBWT is somewhat misleading, since there is no array analogous to the BWT.

Conditions 1, 2 and 4 (equality, left-maximality and row-maximality) of Definition 1 can be stated in terms of the arrays a_l and d_l as follows. Suppose we have a block (K, i, j) . If the block is a maximal perfect haplotype block, then the set $\{a_j^{-1}[r] \mid r \in K\}$ must be a contiguous range $[x, y]$ of indices such that the following holds:

- $d_j[r] \leq i$ for all $r \in [x + 1, y]$ (equality),
- there exists at least one $r \in [x + 1, y]$ such that $d_j[r] = i$ (left-maximality), and
- $(x = 1$ or $d_j[x] > i)$ and $(y = k$ or $d_j[y + 1] > i)$ (row-maximality).

We call a block satisfying these conditions an *available* block and $[x, y]$ the *colexicographic range* of the block. Let us consider the set B_l of available blocks ending at column l . We have that $|B_l| \leq k$ because each available block corresponds to a distinct branching node in the trie of the reverses of $\{S[1][1..l], \dots, S[k][1..l]\}$, and the number of branching nodes in the trie is bounded from above by the number of leaves k . The branching nodes of the trie can be enumerated in $O(k)$ time by using a standard algorithm [2] for enumerating LCP intervals of the LCP array of the trie, $LCP_l[r] = l - d_l[r] + 1$. This gives us the colexicographic ranges $[x, y]$ of all available blocks in B_l . An example is shown in Figure 2.

The only thing left is to show how to check the right-maximality property of an available block, i.e., whether $j = n$ or $|\{S[a[r]][j + 1] : r \in [x, y]\}| \geq 2$. To check the condition in constant time for $j \neq n$, we build a bit vector V_j such that $V_j[1] = 1$ and $V_j[r] = 1$ if and only if $S[a_j[r]][j + 1] \neq S[a_j[r - 1]][j + 1]$. Now the block is right-maximal if and only if $V_j[x + 1..y]$ contains at least one 1-bit. We can build a vector of prefix sums of V_j to answer this question in constant time.

Time and space complexity

We assume the *column stream model*, where we can stream the haplotype matrix column by column. We can thus build the arrays d_l , a_l and a_l^{-1} on the fly column by column [5], and also easily build the required prefix sums of arrays V_l from these. The time is $O(nk)$, since each of the n columns takes $O(k)$ time to process. The algorithm needs to keep in memory only the data for two adjacent columns at a time, so in space $O(k)$ we can report the colexicographic ranges of all maximal blocks ending in each column $l \in [1, n]$. If the colexicographic range of a block at column l is $[x, y]$, then the rows in the original haplotype matrix are $a_l[x], a_l[x + 1], \dots, a_l[y]$. There are $O(nk)$ blocks and $O(k)$ rows per block, so the time to report all rows explicitly is $O(nk^2)$. Alternatively, we can store a complete representation of the answer taking $O(nk)$ space by storing all the a_l arrays and the colexicographic ranges of the maximal perfect blocks for each column, from which we can readily report all rows in any maximal perfect block in constant time per row.

5 Empirical Evaluation

Since the algorithm of Section 3 is mostly of theoretical interest, we evaluate only the pBWT-based algorithm presented in Section 4. The source code is available from <https://gitlab.com/bacazaux/haploblocks>. As a baseline for comparison we use the implementation of the trie-based algorithm by Cunha et al. [4], available from the same gitlab site. The experiments were run on a machine with an Intel Xeon E5-2680 v4 2.4GHz CPU, which has a 35 MB Intel SmartCache. The machine has 256 gigabytes of memory at a speed of 2400MT/s. The code was compiled with `g++` using the `-Ofast` optimization flag.

Our test data consists of chromosomes 2, 6 and 22 from phase three of the 1000 Genomes Project [1], which provides whole-genome sequences of 2,504 individuals from multiple populations worldwide. We preprocessed the data by extracting all biallelic SNPs from the provided VCF files³ and converting them to a binary haplotype matrix using our own program `vcf2bm`, also available from <https://gitlab.com/bacazaux/haploblocks>.

Our implementation has a user-defined parameter allowing to adjust the minimum size of a reported maximal perfect haplotype block (K, i, j) , where *size* is defined as the width $(j - i + 1)$ times the number of rows $(|K|)$ in the block. Table 1 shows the running times and memory usage of our implementation on the different chromosomes and for different settings of the minimum block size parameter. The larger the minimum block size, the faster the algorithm is, because there are less blocks to report. In general, it takes only a few minutes to process a complete human chromosome. Locating all 323,163,970 blocks of minimum size 10^6 in all 22 human autosomes (non-sex chromosomes) took in total 4 hours and 26 minutes with a memory peak of 12.8 MB (data not shown).

Table 2 shows a comparison of our implementation to the trie-based implementation from [4]. Our implementation is about 5 times faster on all datasets, and the memory consumption is up to 93 times smaller.

Using the method for estimating a local selection coefficient from the size of maximal perfect haplotype blocks covering a certain genomic region presented in [4], it is now possible to generate chromosome-wide selection scans indicating the loci of maximum selection, as shown in Figure 3 for the complete human chromosome 2 (size parameter 10^6), in less than half an hour.

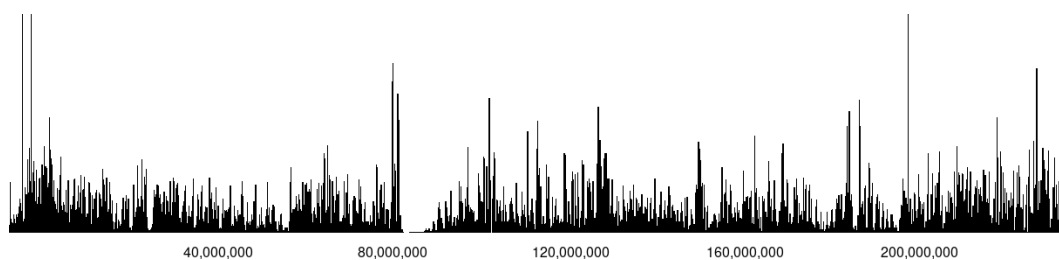
³ <ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/>

■ **Table 1** Running times and memory usage of our pBWT-based implementation. Note that in our streaming implementation the memory usage is dominated by the number of haplotypes times the buffer size, and therefore is essentially constant in this study.

data set	#lines	#columns	min block size	time	memory	#blocks
chr. 22	5,008	1,055,454		4min 54s	12.8MB	148,613,645
chr. 22	5,008	1,055,454	500,000	3min 50s	12.8MB	16,076,453
chr. 22	5,008	1,055,454	1,000,000	3min 40s	12.8MB	2,228,762
chr. 22	5,008	1,055,454	2,000,000	3min 43s	12.8MB	4,779
chr. 6	5,008	4,800,101		19min 42s	12.8MB	624,689,548
chr. 6	5,008	4,800,101	500,000	17min 20s	12.8MB	89,840,467
chr. 6	5,008	4,800,101	1,000,000	16min 30s	12.8MB	11,388,982
chr. 6	5,008	4,800,101	2,000,000	16min 36s	12.8MB	5,585
chr. 2	5,008	6,786,300		31min 57s	12.8MB	946,717,897
chr. 2	5,008	6,786,300	500,000	25min 06s	12.8MB	160,094,115
chr. 2	5,008	6,786,300	1,000,000	23min 24s	12.8MB	25,533,314
chr. 2	5,008	6,786,300	2,000,000	23min 18s	12.8MB	120,243

■ **Table 2** Comparison of the trie-based implementation from [4] and our pBWT-based implementation with minimum block size 10^6 .

data set	trie		pBWT	
	time	memory	time	memory
chr. 22	17min 08s	927.8MB	3min 40s	12.8MB
chr. 6	1h 34min 34s	3.23GB	16min 30s	12.8MB
chr. 2	2h 07min 21s	4.46GB	23min 24s	12.8MB



■ **Figure 3** Selection scan for human chromosome 2. Shown is for each position of the chromosome the largest maximum likelihood estimate derived from any maximal perfect haplotype block overlapping that locus. It is easy to spot potential regions of high selection. The centromere, located around 93 Mbp, shows no signal as sequencing coverage is low here and no SNPs could be called.

6 Conclusion

In this paper we presented two algorithms that are able to find all maximal perfect haplotype blocks in a haplotype matrix of size $k \times n$ in linear time $O(kn)$. In particular the second method, based on the positional Burrows-Wheeler Transform, performs also extremely well in practice, as it allows for a streaming implementation with extremely low memory footprint.

While an initial implementation of the method is available from <https://gitlab.com/bacazaux/haploblocks>, a user-friendly software combining the algorithm presented here with the computation of the selection coefficient suggested in [4] remains to be developed.

References

- 1 1000 Genomes Project Consortium, Adam Auton, Lisa D Brooks, Richard M Durbin, Erik P Garrison, Hyun Min Kang, Jan O Korb, Jonathan L Marchini, Shane McCarthy, Gil A McVean, and Gonçalo R Abecasis. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015. doi:10.1038/nature15393.
- 2 Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004. doi:10.1016/S1570-8667(03)00065-0.
- 3 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 Vrousou, Patricia L Whetzel, Ridwan Amode, Jose A Guillen, Harpreet S Riat, Stephen J Trevanion, Peggy Hall, Heather Junkins, Paul Flicek, Tony Burdett, Lucia A Hindorf, 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, 2018. doi:10.1093/nar/gky1120.
- 4 Luís Cunha, Yoan Diekmann, Luis Antonio Brasil Kowada, and Jens Stoye. Identifying Maximal Perfect Haplotype Blocks. In *Advances in Bioinformatics and Computational Biology - 11th Brazilian Symposium on Bioinformatics, BSB 2018, Niterói, Brazil, October 30 - November 1, 2018, Proceedings*, pages 26–37, 2018. doi:10.1007/978-3-030-01722-4_3.
- 5 Richard Durbin. Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). *Bioinformatics*, 30(9):1266–1272, 2014. doi:10.1093/bioinformatics/btu014.
- 6 Martin Farach. Optimal suffix tree construction with large alphabets. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 137–143. IEEE, 1997.
- 7 John H. Gillespie. *Population Genetics - A Concise Guide*. The Johns Hopkins University Press, Baltimore and London, 1998.
- 8 Daniel F Gudbjartsson, Hannes Helgason, Sigurjon A Gudjonsson, Florian Zink, Asmundur Oddson, Arnaldur Gylfason, Søren Besenbacher, Gisli Magnusson, Bjarni V Halldorsson, Eirikur Hjartarson, Gunnar Th Sigurdsson, Simon N Stacey, Michael L Frigge, Hilma Holm, Jona Saemundsdottir, Hafdis Th Helgadóttir, Hrefna Johannsdóttir, Gunnlaugur Sigfusson, Gudmundur Thorgeirsson, Jon Th Sverrisson, Solveig Gretarsdóttir, G Bragi Walters, Thorunn Rafnar, Bjarni Thjodleifsson, Einar S Bjornsson, Sigurdur Olafsson, Hildur Thorarinsdóttir, Thora Steingrimsdóttir, Thora S Gudmundsdóttir, Asgeir Theodors, Jon G Jonasson, Asgeir Sigurdsson, Gyda Bjornsdóttir, Jon J Jonsson, Olafur Thorarensen, Petur Ludvigsson, Hakon Gudbjartsson, Gudmundur I Eyjolfsson, Olof Sigurdardóttir, Isleifur Olafsson, David O Arnar, Olafur Th Magnusson, Augustine Kong, Gisli Masson, Unnur Thorsteinsdóttir, Agnar Helgason, Patrick Sulem, and Kari Stefansson. Large-scale whole-genome sequencing of the Icelandic population. *Nature Genetics*, 47:435–444, 2015. doi:10.1038/ng.3247.
- 9 Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, Cambridge, 1997.

- 10 Jayne Y Hehir-Kwa, Tobias Marschall, Wigard P Kloosterman, Laurent C Francioli, Jasmijn A Baaijens, Louis J Dijkstra, Abdel Abdellaoui, Vyacheslav Koval, Djie Tjwan Thung, René Wardenaar, Ivo Renkens, Bradley P Coe, Patrick Deelen, Joep de Ligt, Eric-Wubbo Lameijer, Freerk van Dijk, Fereydoun Hormozdiari, The Genome of the Netherlands Consortium, Jasper A Bovenberg, Anton J M de Craen, Marian Beekman, Albert Hofman, Gonneke Willemsen, Bruce Wolffenbuttel, Mathieu Platteel, Yuanping Du, Ruoyan Chen, Hongzhi Cao, Rui Cao, Yushen Sun, Jeremy Sujie Cao, Pieter B T Neerincx, Martijn Dijkstra, George Byelas, Alexandros Kanterakis, Jan Bot, Martijn Vermaat, Jeroen F J Laros, Johan T den Dunnen, Peter de Knijff, Lennart C Karssen, Elisa M van Leeuwen, Najaf Amin, Fernando Rivadeneira, Karol Estrada, Jouke-Jan Hottenga, V Mathijs Kattenberg, David van Enkevort, Hailiang Mei, Mark Santcroos, Barbera D C van Schaik, Robert E Handsaker, Steven A McCarroll, Arthur Ko, Peter Sudmant, Isaac J Nijman, André G Uitterlinden, Cornelia M van Duijn, Evan E Eichler, Paul I W de Bakker, Morris A Swertz, Cisca Wijmenga, Gert-Jan B van Ommen, P Eline Slagboom, Dorret I Boomsma, Alexander Schönhuth, Kai Ye, and Victor Guryev. A high-quality human reference panel reveals the complexity and distribution of genomic structural variants. *Nature Communications*, 7:12989, 2016. doi:10.1038/ncomms12989.
- 11 Motoo Kimura. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics*, 61(4):893, 1969.
- 12 Peter H Sudmant, Tobias Rausch, Eugene J Gardner, Robert E Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, Miriam K Konkel, Ankit Malhotra, Adrian M Stütz, Xinghua Shi, Francesco Paolo Casale, Jieming Chen, Fereydoun Hormozdiari, Gargi Dayama, Ken Chen, Maika Malig, Mark J P Chaisson, Klaudia Walter, Sascha Meiers, Seva Kashin, Erik Garrison, Adam Auton, Hugo Y K Lam, Ximeng Jasmine Mu, Can Alkan, Danny Antaki, Taejeong Bae, Eliza Cerveira, Peter Chines, Zechen Chong, Laura Clarke, Elif Dal, Li Ding, Sarah Emery, Xian Fan, Madhusudan Gujral, Fatma Kahveci, Jeffrey M Kidd, Yu Kong, Eric-Wubbo Lameijer, Shane McCarthy, Paul Flicek, Richard A Gibbs, Gabor Marth, Christopher E Mason, Androniki Menelaou, Donna M Muzny, Bradley J Nelson, Amina Noor, Nicholas F Parrish, Matthew Pendleton, Andrew Quitadamo, Benjamin Raeder, Eric E Schadt, Mallory Romanovitch, Andreas Schlattl, Robert Sebra, Andrey A Shabalina, Andreas Untergasser, Jerilyn A Walker, Min Wang, Fuli Yu, Chengsheng Zhang, Jing Zhang, Xiangqun Zheng-Bradley, Wanding Zhou, Thomas Zichner, Jonathan Sebat, Mark A Batzer, Steven A McCarroll, The 1000 Genomes Project Consortium, Ryan E Mills, Mark B Gerstein, Ali Bashir, Oliver Stegle, Scott E Devine, Charles Lee, Evan E Eichler, and Jan O Korbel. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015. doi:10.1038/nature15394.
- 13 Clare Turnbull, Richard H Scott, Ellen Thomas, Louise Jones, Nirupa Murugaesu, Freya Boardman Pretty, Dina Halai, Emma Baple, Clare Craig, Angela Hamblin, Shirley Henderson, Christine Patch, Amanda O'Neill, Andrew Devereau, Katherine Smith, Antonio Rueda Martin, Alona Sosinsky, Ellen M McDonagh, Razvan Sultana, Michael Mueller, Damian Smedley, Adam Toms, Lisa Dinh, Tom Fowler, Mark Bale, Tim J P Hubbard, Augusto Rendon, Sue Hill, Mark J Caulfield, and 100 000 Genomes Project. The 100 000 Genomes Project: bringing whole genome sequencing to the NHS. *BMJ*, 361:k1687, 2018. doi:10.1136/bmj.k1687.

A New Paradigm for Identifying Reconciliation-Scenario Altering Mutations Conferring Environmental Adaptation

Roni Zoller

Ben Gurion University of the Negev, Israel
ronizo@post.bgu.ac.il

Meirav Zehavi¹

Ben Gurion University of the Negev, Israel
meiravze@bgu.ac.il

Michal Ziv-Ukelson¹

Ben Gurion University of the Negev, Israel
michaluz@cs.bgu.ac.il

Abstract

An important goal in microbial computational genomics is to identify crucial events in the evolution of a gene that severely alter the duplication, loss and mobilization patterns of the gene within the genomes in which it disseminates. In this paper, we formalize this microbiological goal as a new pattern-matching problem in the domain of Gene tree and Species tree reconciliation, denoted “Reconciliation-Scenario Altering Mutation (RSAM) Discovery”. We propose an $O(m \cdot n \cdot k)$ time algorithm to solve this new problem, where m and n are the number of vertices of the input Gene tree and Species tree, respectively, and k is a user-specified parameter that bounds from above the number of optimal solutions of interest. The algorithm first constructs a hypergraph representing the k highest scoring reconciliation scenarios between the given Gene tree and Species tree, and then interrogates this hypergraph for subtrees matching a pre-specified RSAM Pattern. Our algorithm is optimal in the sense that the number of hypernodes in the hypergraph can be lower bounded by $\Omega(m \cdot n \cdot k)$. We implement the new algorithm as a tool, denoted RSAM-finder, and demonstrate its application to the identification of RSAMs in toxins and drug resistance elements across a dataset spanning hundreds of species.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Gene tree, Species tree, Reconciliation

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.9

Supplement Material <https://github.com/ronizoller/RSAM>

Funding Supported by ISF grants no. 1176/18 and no. 939/18. This work was supported by the Lynn and William Frankel Center for Computer Science.

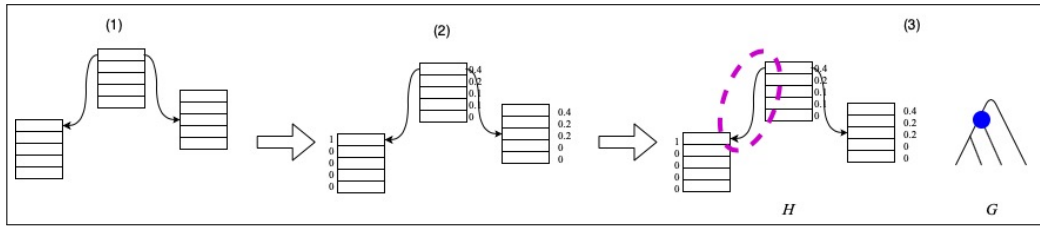
Acknowledgements We thank the anonymous WABI reviewers for very helpful comments.

1 Introduction

Prokaryotes can be found in the most diverse and severe ecological niches of the planet. Adaptation of prokaryotes to new niches requires expanding their repertoire of protein families, via two evolutionary processes: first, by selection of novel gene mutants carrying stable genetic alterations that confer adaptation, and second, by dissemination of an adaptively mutated gene. These two processes are correlated: an adaptation-conferring mutation in a

¹ Corresponding authors.





■ **Figure 1** High-level overview of the RSAM-finder algorithm.

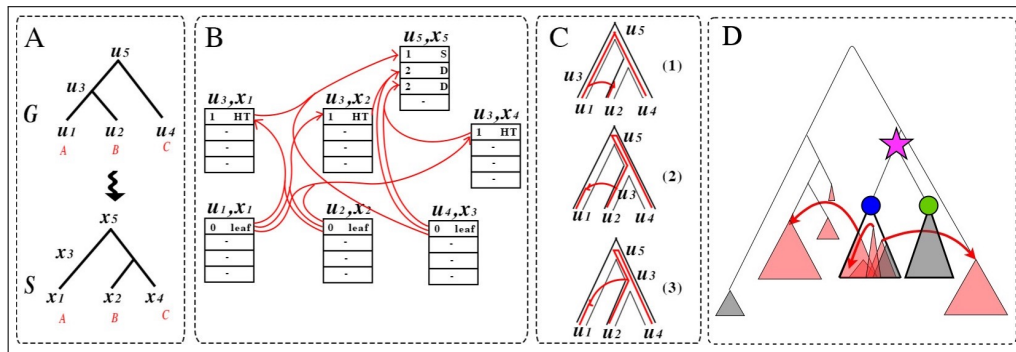
gene could accelerate its mobilization across bacterial lineages populating the corresponding environmental niche [25], and vice-versa, the mobilization of a gene by transposable elements increases its chances to mutate or “pick up” novel genomic context. Thus, an important research goal is to identify gene-level mutations that affect the spreading pattern of the mutated gene within and across the genomes harboring it.

For example, consider mutations conferring adaptation of bacteria to a human-pathogenesis environment. Here, a mutation to a resistance or virulence factor could enhance pathogenic adaptation, thus increasing the horizontal mobilization of the mutated gene within other human pathogens inhabiting this niche [25]. In this case, we say that the mutation has a *causal association* with the observed dissemination pattern of the mutated gene (i.e. the increased mobilization of the gene among pathogenic bacteria). Identifying such mutations could inform infectious disease monitoring and outbreak control, and assist in identifying potential drug targets.

The co-evolution of genes and their host species is classically described by computing the most parsimonious reconciliation scenario between a given Gene tree G and the corresponding Species tree S , that is, a mapping of each vertex $u \in G$ to a vertex $x \in S$. Three major evolutionary processes, traditionally considered by reconciliation approaches, are horizontal gene transfer, gene duplication, and gene loss [36]. Each mapping of a vertex $u \in G$ to a vertex $x \in S$ is associated with one of these evolutionary events, and assigned a cost, accordingly. The optimization problem of computing a least-cost reconciliation between G and S , where the total cost is computed as the sum of the costs assigned to each of the mappings, is denoted *Duplication-Loss-Transfer (DLT) Reconciliation*.

Motivated by examples such as the one given above, we formalize a new pattern-matching problem in the domain of DLT reconciliation. Given are a Gene tree G , a corresponding Species tree S , a mapping σ from the leaves of G to the leaves of S , and (optional) an environmental annotation labeling the leaves of the input trees. Let \mathcal{H} denote some data structure, to be defined later in the paper, that models the space of reconciliations between G and S . A *DLT Reconciliation Scenario Pattern* denotes a mapping between a vertex $u \in G$ to a vertex $x \in S$, which obeys a set of user-defined specifications regarding the corresponding reconciliation event, the labels on the paired vertices, and other features associated with the mapping. Mappings between pairs of vertices ($u \in G, x \in S$) that abide by the requirements specified by P are denoted *instances of P in \mathcal{H}* . Given a pre-specified DLT Reconciliation Scenario pattern P and a data structure \mathcal{H} modeling the space of reconciliations between G and S , a *Reconciliation Scenario Altering Mutation (RSAM) of P in \mathcal{H}* is a vertex $v \in G$ representing a gene mutation with a putative causal association to instances of P in \mathcal{H} . The *RSAM Discovery problem* is to identify RSAMs in G .

In what follows, we propose a three-stage solution to the RSAM Discovery problem defined above (illustrated in Fig. 1). The first stage constructs a hypergraph \mathcal{H} that recursively aggregates all the k -best reconciliations of G and S . Each supernode in \mathcal{H} consists of k



■ **Figure 2** Various aspects of the problem addressed in this paper. (A) The input trees G and S . (B) An example of a hypergraph constructed based on the input trees and parameter k . (C) Three top-scoring solutions. (D) A putative RSAM (blue vertex) with causal association to the mobilization pattern of the gene among species labeled with the red environmental annotation.

hypernodes, where each hypernode represents a partial solution for the DLT-reconciliation problem. Our hypergraph-ensemble approach is based on a model proposed by [24] for network evolution, where here we extend and adapt it to the DLT model. This hypergraph of k -best reconciliations, intended to provide some robustness to the noise typical of this data, will serve as the search-space for the pattern-matching stage. The second stage of our proposed solution consists of assigning a probability to each partial solution, that is, to each hypernode of \mathcal{H} . Finally, in the third stage, instances of the sought RSAM-pattern P are identified within \mathcal{H} , and RSAM-ranking scores are assigned accordingly to the vertices of G . Based on these scores, vertices representing putative RSAMs are identified in G and subjected to biological interpretation.

The construction of \mathcal{H} , in the first stage, is the computational bottleneck of the RSAM-Discovery pipeline mentioned above. Here, we adapt the approach proposed by Bansal et al. [3] for the basic, one-best variant of DLT reconciliation, extending it to an efficient k -best variant. This yields an $O(m \cdot n \cdot k)$ time algorithm for the problem, where m and n are the number of vertices of the input Gene tree and Species tree, respectively, and k is a user-specified parameter that bounds from above the number of optimal solutions of interest. Our algorithm is optimal in the sense that the number of hypernodes in the hypergraph can be lower bounded by $\Omega(m \cdot n \cdot k)$.

Our proposed solution to the problem defined in this paper is implemented as a tool called RSAM-finder, publicly available on GitHub. We assert the performance of RSAM-finder in large scale simulations, and exemplify its application to the identification of RSAMs in toxins and drug resistance elements across a dataset spanning hundreds of species.

Previous Related Works. The DLT Reconciliation problem has been extensively studied. In particular, two main DLT variants have been considered: (1) the *undated DLT-reconciliation* where the species are undated, and (2) the *fully-dated DLT-reconciliation* where either each vertex in the Species (and Gene) tree is associated with an estimated date or the vertices of the Species (and Gene) tree are associated with a total order, and any reconciliation must respect these dates (i.e. an HT event can occur only between co-existing species).

In the acyclic version of these variants, there cannot exist two genes such that one is a descendant of the other, yet the descendant is mapped (in the Species tree S) to an ancestor of the other. Tofigh et al. [36] showed that the acyclic undated version is NP-Hard. However, the acyclic dated version becomes polynomially solvable [19].

Tofigh et al. [36, 35] and David et al. [12] studied a version of the undated (cyclic) problem that ignores losses and proposed an $O(mn^2)$ dynamic programming algorithm for it. They also gave a fixed-parameter tractable algorithm for enumerating all optimal solutions. The time complexity of the algorithm was improved to $O(mn)$ in [35] (under a restricted model that ignores the losses) and in [3] (which does not ignore losses).

It is well-known that the biological data used as input to the DLT Reconciliation problem could be inaccurate, whether due to a sequencing problem, a problem in the reconstruction of G or S [5], or due to some other problem caused by noise. To overcome this problem, previous works try to examine more than one optimal solution, for example [13, 27]. A probabilistic method for exploring the space of optimal solutions was suggested in [4, 14], where the latter was improved in [15]. Additional studies considered a space of candidate co-optimal scenarios within special variants of the DLT problem, some of which employed special constraints to drive the search [30, 34, 22, 10]. Although all of the previous works reviewed in this paragraph compute a space of candidate reconciliation scenarios, none of these works mentioned considered the application of pattern matching on this space, as we do in this work.

DLT Reconciliation algorithm variants, where the reconciliation computation is guided by constraints derived from vertex-coloring information, were proposed in applications studying host-parasite co-evolution, such as [6], where the vertex coloring (in both G and S) represents the geographical area of residence. However, the applied constraints were “hard-wired” to the specific problem addressed in that paper. In contrast, the approach proposed in this paper is more general, supporting a pattern-search that is guided by a user-defined pattern. Our tool RSAM-finder provides the users with a query language able to express more robust patterns, according to the various applications where the pattern-search is to be employed.

2 Preliminaries

For a (binary) rooted tree T , let $L(T)$, $V(T)$, $I(T)$ and $E(T)$ denote the sets of leaves, vertices, internal vertices and edges, respectively, of T . Additionally, let $V(T)^*$ denote the set of finite (ordered) vectors over $V(T)$, i.e. $V(T)^* = \{(v_1, v_2, \dots, v_\ell) \mid v_i \in V(T) \text{ for all } i \in \{1, \dots, \ell\}, \ell \in \mathbb{N}\}$. When T is clear from context, let $V^* = V(T)^*$. Throughout, we treat any (binary) rooted tree T as a directed graph whose edges are directed from root to leaves. Then, if $(u, v) \in E(T)$, we say that v is a *child* of u , and u is the *parent* of v . For $u, v \in V(T)$, the notation $v \leq_T u$ signifies that v is a *descendant* of u (alternatively, u is an *ancestor* of v), i.e. there is a directed path from u to v or $u = v$. We say that v is a proper ancestor (proper ancestor) of u if $v \leq_T u$ ($v \geq_T u$) and $u \neq v$, denoted $<_T$ ($>_T$). When both $u \not\leq_T v$ and $v \not\leq_T u$, we say that u and v are *incomparable*.

For any $u, v \in V(T)$, let $d_T(u, v)$ denote the number of edges in the (unique simple undirected) path between u and v in T . When T is clear from context, we drop it from the notations $v \leq_T u$ and $d_T(u, v)$. For any $u \in V(T)$, let T_u denote the subtree of T rooted in u (then, $V(T_u) = \{v \in V(T) \mid v \leq u\}$).

DLT Scenario. A *DLT scenario* for two binary trees G (the *Gene tree*) and S (the *Species tree*) is a tuple $\langle \sigma, \gamma, \Sigma, \Delta, \Theta, \Xi \rangle$ where $\sigma : L(G) \rightarrow L(S)$ is a mapping of the leaves of G to the leaves of S , $\gamma : V(G) \rightarrow V(S)$ is a mapping of the vertices of G to the vertices of S , and (Σ, Δ, Θ) is a partition of $I(G)$ (the set of internal vertices of G) into three event classes: *Speciation* (Σ), *Duplication* (Δ) and *Horizontal Transfer* (Θ). The subset $\Xi \subseteq E(G)$ specifies which edges are involved in horizontal transfer events. Additionally, the following constraints should be satisfied.

1. **Consistency of σ and γ .** For each leaf $u \in L(G)$, $\gamma(u) = \sigma(u)$. This constraint ensures that γ respects σ – that is, each leaf of G is mapped to the species where it is found.
2. **Consistency of γ and ancestorship relations in S .** For each $u \in I(G)$ with children v and w :
 - a. $\gamma(u) \not\leq_S \gamma(v)$ and $\gamma(u) \not\leq_S \gamma(w)$. This constraint ensures that each of the two children (in G) of the gene u is mapped by γ to a species that is not a proper ancestor (in S) of the species to which the gene u is mapped; thus, it can be either a descendant of u or incomparable to u .
 - b. At least one of $\gamma(v)$ and $\gamma(w)$ is a descendant of $\gamma(u)$. This constraint ensures that at least one of the two children (in G) of the gene u is mapped by γ to a species that is a descendant (in S) of the species to which the gene u is mapped.
3. **Identifying horizontal transfer edges.** For each edge $(u, v) \in E(G)$, it holds that $(u, v) \in \Xi$ if and only if $\gamma(u) \not\leq_S \gamma(v)$ and $\gamma(v) \not\leq_S \gamma(u)$. This constraint identifies which edges are horizontal transfer edges – specifically, a horizontal transfer edge is an edge $(u, v) \in E(G)$ from a gene u to a gene v that are mapped to species $\gamma(u)$ and $\gamma(v)$ that are incomparable.
4. **Associating events with internal vertices.** For each $u \in I(G)$ with children v, w :
 - a. **Speciation.** $u \in \Sigma$ only if both (i) $\gamma(u) = \text{lca}(\gamma(v), \gamma(w))$ and (ii) $\gamma(v)$ and $\gamma(w)$ are incomparable (i.e. $\gamma(v) \not\leq_S \gamma(w)$ and $\gamma(w) \not\leq_S \gamma(v)$).
 - b. **Duplication.** $u \in \Delta$ only if $\gamma(u) \geq_S \text{lca}(\gamma(v), \gamma(w))$.
 - c. **Horizontal transfer.** $u \in \Theta$ if and only if either (i) $(u, v) \in \Xi$ or (ii) $(u, w) \in \Xi$

Fig. S1 demonstrates a DLT scenario.

Costs. We let c_Δ and c_Θ denote the costs of a duplication event and a horizontal transfer event, respectively. Accordingly, the cost of a DLT scenario is defined as $|\Delta| \cdot c_\Delta + |\Theta| \cdot c_\Theta$. When seeking a “best” DLT scenario, the goal is to find one that minimizes this cost. It is straightforward to extend the cost model, and the corresponding algorithm, to take losses into account, yet for lack of space, these details are omitted throughout the paper.

3 Hypergraph of k -Best Scenarios

To represent k -best solutions,² we use a directed hypergraph denoted by \mathcal{H} based on the notation in [17]. The hypergraph is a tuple $\mathcal{H} = \langle V, E \rangle$ where V is a finite set of vertices, and E is a finite set of (directed) hyperedges defined as follows. Each $e \in E$ is a pair $\langle T(e), h(e) \rangle$, where $h(e) \in V$ is the head of e and $T(e) \in V^*$ (i.e. $T(e)$ is a vector of vertices in V) is its tail. In our settings, $|T(e)| = 2$ for every $e \in E$. In what follows, we define the hypernodes and hyperedges of \mathcal{H} with respect to our problem. In Fig. 2.B, we illustrate the hypergraph corresponding to the input G and S given in Fig. 2.A, where $k = 4$. Each hypernode (u, x, i) represents a DLT scenario, annotated with its cost and with the event that occurred between u and x in this scenario (where ‘S’, ‘D’ and ‘HT’ stand for speciation, duplication and horizontal transfer, respectively). For additional details, see the Supplementary Materials.

- **Hypernodes.** For every vertex u in G , a vertex x in S and an integer $i \in \{1, \dots, k\}$, we have a node (u, x, i) in \mathcal{H} . Such a node (u, x, i) is associated with the i^{th} best (where ties are broken arbitrarily) solution mapping the subtree of G rooted at u to the subtree of S

² That is, k DLT scenarios of the highest score(s), where ties (if any exist) can be broken arbitrarily.

rooted in x that is a DLT scenario. In addition, for every integer $i \in \{1, \dots, k\}$ we have a node $(root, i)$ in the hypergraph \mathcal{H} . Such a node $(root, i)$ is associated with the i^{th} best solution of mapping G (entirely) to any subtree of S . Each node (u, x, i) has a score $c(u, x, i)$, and each node $(root, i)$ has a score $c(root, i)$. Moreover, each node (u, x, i) is associated with the event corresponding to the mapping of u and x in the DLT scenario of (u, x, i) (speciation, duplication and horizontal transfer), denoted $\text{event}(u, x, i)$.

- **Supernodes.** For any vertex $u \in V(G)$ and vertex $x \in V(S)$, we define the *supernode* (u, x) as the list $\{(u, x, i) : 1 \leq i \leq k\}$ (i.e. (u, x) is the set of k hypernodes corresponding to the mapping of the subtree of G rooted in u to the subtree of G rooted in x). This notation will simplify our presentation.
- **Hyperedges.** Recall that each hypernode $(u, x, i) \in V(\mathcal{H})$ describes a DLT scenario. Each hypernode has exactly one incoming hyperedge, but it can have multiple outgoing hyperedges. In particular, for each hypernode $(u, x, i) \in V(\mathcal{H})$, the (only) incoming hyperedge $e = \langle T(e), h(e) \rangle = \langle [(v, y, j), (w, z, r)], (u, x, i) \rangle$ describes the mapping of the subtrees of the children of u , namely, v and w , in the scenario of (u, x, i) ; here, the subtree of v is mapped to the subtree of y as in the scenario of (v, y, j) , and the subtree of w is mapped to the subtree of z as in the scenario of (w, z, r) .

4 Framework and Algorithms

In this section, we elaborate on each of the three stages of the workflow in Section 1.

4.1 Stage 1: Hypergraph Construction

The first stage of our framework constructs the hypergraph described in Section 3. To this end, we develop an efficient algorithm that runs in time $O(m \cdot n \cdot k)$. The technical details (including pseudocode) are given in the Supplementary Materials.

The Algorithm. We iterate over all $u \in V(G)$ in postorder, as well as over all $x \in V(S)$ in postorder. (However, as explained immediately, when we consider a vertex $u \in V(G)$, after iterating over all vertices $x \in V(S)$ in postorder, we also iterate over all vertices $x \in V(S)$ in preorder.) In each iteration, corresponding to a pair (u, x) , we construct three lists: p_Σ (speciation), p_Δ (duplication) and p_Θ (horizontal transfer). Specifically, p_Σ should be a list of k -best solutions that are DLT scenarios where the subtree of G rooted in u is mapped to the subtree of S rooted in x under the restriction that the event corresponding to matching u and x is speciation. The meaning of the lists p_Δ and p_Θ is similar, where the restriction of speciation is replaced by duplication or horizontal transfer, respectively. Having these three lists suffices to construct the hypernode (u, x) .

To avoid repetitive computation, we maintain two additional lists: `subtree` and `incomp`. Intuitively, `subtree` (u, x, i) is the i^{th} best cost of a reconciliation of the subtree of G rooted in u with some subtree of S whose root is a vertex y that is a descendant of x , and `incomp` (u, x, i) is the i^{th} best cost of a reconciliation of the subtree of G rooted in u with some subtree of S whose root is a vertex y incomparable to x . We use `subtree` to speed-up the computation of p_Σ , p_Δ and p_Θ , and `incomp` to speed-up the computation of p_Θ . The notations `subtree` (u, x) and `incomp` (u, x) refer to the lists of the k -best scores $\{\text{subtree}(u, x, i)\}_{i=1}^k$ and $\{\text{incomp}(u, x, i)\}_{i=1}^k$, respectively, similarly to our usage of the notation of a supernode.

The efficient computation of p_Σ , p_Δ and p_Θ , along with the maintenance of `subtree` and `incomp` themselves, is highly non-trivial. For space constraints, the technical details (in particular, pseudocode) are delegated to the Supplementary Materials Section 2.1. On a high-level, we first initialize all five lists to contain only costs of ∞ ; then, still in the initialization phase, we add hypernodes that match between leaves of G and S in accordance with σ and update `subtree` consequently. After the initialization, the main computation considers each $u \in V(G)$ in postorder, and performs two steps. In the first step, we consider each $x \in V(S)$ in postorder. Then, for each $i \in \{1, \dots, k\}$, we compute $p_\Sigma(u, x, i)$, $p_\Delta(u, x, i)$ and $p_\Theta(u, x, i)$ based on somewhat involved recursive formulas. Afterwards, we construct the supernode (u, x) , as well as compute the list `subtree`(u, x). In the second step, we consider each $x \in I(S)$ with children y and z in preorder, and compute the lists `incomp`(u, y) and `incomp`(u, z).

Having constructed all hypernodes of the form (u, x, i) along with their ingoing hyperedges, it is trivial to construct the hypernodes of the form $(root, i)$ and their ingoing edges. Thus, we conclude the outline with statements of correctness and running time proved in Supplementary Materials Section 2.1.

► **Lemma 1.** *Given an instance (G, S, σ) of the DLT problem and a positive integer k , the efficient algorithm correctly constructs a hypergraph \mathcal{H} that represents k -best solutions for (G, S, σ) .*

► **Remark 2.** Given an instance (G, S, σ) of the DLT problem and a positive integer k , the efficient algorithm runs in time $O(m \cdot n \cdot k)$.

4.2 Stage 2: Assigning Probabilities

In the second stage, we assign a probability to each hypernode so that a hypernode with best score has the highest probability, and hypernodes with score ∞ (the worst possible score) have probability 0.

Weight Computation. Let $\gamma \in \mathbb{R}^+$ be a user-specified parameter. As γ grows lower, hypernodes with higher (worse) scores are assigned probabilities much lower than hypernodes with lower scores.

Denote $r = root$, and let $m(r)$ be the largest integer $i \in \{1, \dots, k\}$ such that $c(r, i) \neq \infty$ (the notation $(root, i)$ was defined in Section 3). For a node (r, i) where $i \in \{1, \dots, m(r)\}$, define $w'(r, i) = e^{\gamma \frac{c(r,1) - c(r,i)}{c(r,1) - c(r,m(r))}}$. Then, the weight of a node (r, i) , which stands for the (unconditional) probability that the scenario described by (r, i) happens, is defined as follows: if $i \in \{1, \dots, m(r)\}$, then $w(r, i) = \frac{w'(r,i)}{\sum_{j=1}^{m(r)} w'(r,j)}$; otherwise (i.e. if $i \in \{m(r) + 1, m(r) + 2, \dots, k\}$), $w(r, i) = 0$.

We now turn to define the weight of a hypernode (u, x, i) , which should stand for the (unconditional) probability that the scenario described by (u, x, i) happens. The definition is recursive. In the basis, where u is the root of G , we define $w(u, x, i)$ (for any $x \in V(S)$ and $i \in \{1, \dots, k\}$) as follows: if there exists an index $j \in \{1, \dots, k\}$ such that (r, j) is derived from (u, x, i) (here, it means that they represent the same scenario), then $w(u, x, i) = w(r, j)$; otherwise, $w(u, x, i) = 0$.

Now, consider v that is not the root of G . We define $w(v, y, i)$ (for any $y \in V(S)$ and $i \in \{1, \dots, k\}$) as follows. First, let $D(v, y, i)$ denote the collection of nodes (u, x, j) such that $c(u, x, j)$ was derived from $c(v, y, i)$ – in other words, the hypergraph has an hyperedge directed from (v, y, i) (and some other node) to (u, x, j) . In particular, u is the parent of v in G , hence the weight $w(u, x, j)$ is calculated before the weight $w(v, y, i)$. Then, define $w(v, y, i) = \sum_{(u,x,j) \in D(v,y,i)} w(u, x, j)$.

Note that $\sum_{i \in \{1, \dots, k\}} w(r, i) = 1$. As an additional check, we prove the following in Supplementary Materials Section 3.

► **Lemma 3.** *For any two compatible leaves $u \in L(G)$ and $x \in L(S)$, $w(u, x, 1) = 1$.*

Time Complexity. Iterating the hypergraph in $O(|V(\mathcal{H})|) = O(m \cdot n \cdot k)$ time.

4.3 Stage 3: Pattern Discovery

The current version of RSAM-finder allows pattern queries to be specified as follows. A pattern specification consists of a tuple $(\text{EV}, \text{color}, \text{distance})$ where:

1. $\text{EV} \subseteq \{\text{S}, \text{D}, \text{HT}\}$ specifies the evolutionary event of the pattern (S for speciation, D for duplication and HT for horizontal transfer).
2. $\text{color} \in \{\text{red}, \text{black}, \text{None}\}$ specifies the color representing the environmental niche to which the sought RSAM confers adaptation.
3. $\text{distance} \in \{\text{True}, \text{False}\}$ is a boolean indicator specifying whether or not to consider edge lengths (representing evolutionary distances) in the pattern specification.

For a colored query (having the second parameter in the specification set to **red** or **black**), the user is expected to provide, as part of the input, a function $\text{colors} : L(X) \rightarrow \Upsilon$ where X specifies whether the pattern refers to a subtree of S or a subtree of G , and $\Upsilon = \{\text{red}, \text{black}\}$. Here, colors represent a binary environmental annotation of the leaves. Then, a preprocessing step is applied, in which the nodes of S and G are colored based on the colors assigned to the leaves of the subtree they root. We omit the technical details entailing the implementation of this preprocessing step to Supplementary Materials Section 4.1.

In addition to the settings described above, the user can select one of two modes:

1. **Single-pattern mode.** In this mode, the user specifies a single pattern and a threshold, and the sought RSAMs are identified as nodes $u \in I(G)$ such that G_u is enriched in the pattern, and $|V(G_u)|$ is bounded from below by the specified threshold.
2. **Dual-pattern (contrasting) mode.** In this mode, the user specifies two patterns and one threshold, and the sought RSAMs are identified as nodes $u \in I(G)$ with children $v, w \in V(G)$ such that $|V(G_u)|$ is enriched with one pattern while $|V(G_w)|$ is enriched with the other pattern. Here, the threshold that bounds (from below) the subtree-size refers to $|G_v|$ and $|G_w|$.

The following three queries will be exemplified in Section 5,

- $Q_1 : (\{\text{D}\}, \text{None}, \text{False})$. Q_1 identifies vertices $u \in I(G)$, such that G_u is enriched in duplication events.
- $Q_2 : (\{\text{S}, \text{HT}\}, \text{None}, \text{True})$. Q_2 identifies vertices $u \in I(G)$, such that G_u is enriched in speciation and horizontal transfer events (at “the expense” of duplication events).
- $Q_3 : ((\{\text{HT}\}, \text{red}, \text{True}), (\{\text{S}, \text{D}, \text{HT}\}, \text{black}, \text{False}))$. Q_3 identifies vertices $u \in I(G)$ with children $v, w \in V(G)$, such that G_v is enriched in red-to-red horizontal transfer events and G_w is enriched in any black events (illustrated in Fig. 2.D).

The Pattern Identification algorithm proceeds as follows.

1. For each pattern $P = (\text{EV}, \text{color}, \text{distance})$ and for each hypernode $(u, x, i) \in V(\mathcal{H})$, check whether both $\text{event}(u, x, i) \in \text{EV}$ and the colors obey the requirements derived from the **color** field of the pattern specification. (described in more details in Supplementary Materials Section 4.1.1). If so, mark (u, x, i) as interesting.

2. Reflect the interesting nodes identified in \mathcal{H} to G , by assigning corresponding weights to $V(G)$; Each $u \in I(G)$ is assigned a score, which is the cumulative probabilities of instances of the pattern found in G_u , normalized by the number of possible events in G_u . Additional book-keeping details regarding how this score is computed are given in Supplementary Materials Section 4.2.
3. Based on the specified mode of the query (single pattern or dual pattern), identify the t top scoring vertices $u \in I(G)$. In case of a single-pattern mode, the scores are as defined in (2). In case of dual-pattern mode, let P_1 and P_2 be the patterns. To each $u \in V(G)$ with children $v, w \in V(G)$, we assign two scores: the first score of u is the score of v for P_1 (as defined in (2)) plus the score of w for P_2 , and the second is the score of w for P_1 plus the score of v for P_2 .

Time Complexity. Iterating over the hypergraph takes $O(|V(\mathcal{H})|) = O(m \cdot n \cdot k)$ time.

5 Applications

In this section we exemplify preliminary applications of RSAM-finder to genomic analysis. Mobile elements in prokaryotes contribute greatly to the process of gene duplication and dissemination. For example, toxins and antibiotic resistance factors, conferring adaptation to the pathogenesis environment, are often encoded on plasmids, prophages, transposons and other mobile elements in bacteria [25]. Thus, we exemplify two microbiological applications of RSAM-finder by applying query patterns Q_1 , Q_2 , and Q_3 (defined in Section 4.3), to the discovery of RSAMs in toxins and resistance factors.

Supplementary Materials Section 5.2 reports on large scale simulations, where we demonstrate the engine's tolerance to noise, and Supplementary Materials Section 5.3 measures the practical running times of the proposed hypergraph construction algorithm as a function of increasing input size.

Methods and Data Bases. Genes in our experiment are represented by their membership in a Cluster of Orthologous Genes [33]. The STRING database [32] was used to extract the chromosomal protein sequences for the COGs of interest, annotated with their corresponding species names as well as the corresponding NCBI IDs. Protein sequences were subjected to multiple sequence alignment and dendrogram construction via Clustal Omega [28]. The list of NCBI IDs was used as input for the *NCBI Taxonomy Browser* which provided a (non-binary) Species tree. Both Gene and Species trees were converted to binary trees via the Ape R package [26]. Habitat labels for the species were extracted from PATRIC, and missing tags were manually annotated by information from the GOLD database [23] and from literature. MEME motif discovery [2] was employed to identify sequence motifs distinguishing the RSAM-subtree gene sequences from the background. CD Search [20] was employed to seek statistically significant discriminating domain-level mutations (i.e. the gain or loss of a protein functional domain). The simulator and our algorithm were implemented in Python, using NetworkX package, DendroPy [31] and ETE Toolkit [18]. Visualizations of the trees and plots were created using Matplotlib and Seaborn tools.

RSAM Discovery in a Chromosomally Acquired Toxin. Bacterial toxin-antitoxin (TA) systems are diverse and widespread in the prokaryotic kingdom. They are composed of closely linked genes encoding a *stable toxin* that can harm the host cell and its cognate *unstable antitoxin*, which protects the host from the toxin's deleterious effect. TA systems

invade bacterial genomes through horizontal gene transfer. Their role in stabilization and maintenance of plasmids or genomic islands by post-segregational killing has been thoroughly studied [25].

However, much is yet to be learned about how horizontally acquired TA systems are fixed within the population, and about the functions of the chromosomally encoded TA systems. Some TA systems, such as *higBA* [38], have integrated into host regulatory networks, controlling drug tolerance, growth arrest and programmed cell death.

Here we exemplify how RSAM-finder could be harnessed to advance such studies, e.g. by identifying chromosomally acquired variants of the same toxin gene, that exhibit distinct reconciliation patterns: One variant of the toxin follows a “deep and ongoing” invasion pattern (pattern Q_1), i.e. one involving abundant recent duplications within invaded genomes. The other variant follows a “wide and shallow” invasion pattern (pattern Q_2), i.e. the invaded genomes are far-apart in terms of phylogenetic distance, and a very slow rate of duplications within the invaded genomes is observed.

The Plasmid Maintenance System Killer Protein *higB* (represented by COG3549) is the toxin component of the TA module *higBA*. This toxin, which is abundant in Proteobacteria [11], is repressed by the Plasmid Maintenance System Antitoxin *higA* (represented by COG3093). Gene trees for COG3549, and for the chromosomes of Protobacterial species harboring it, were constructed (652 genes versus 493 species), and RSAM-Finder was applied to interrogate this dataset with queries Q_1 and Q_2 . Parameters for Q_1 were set as follows: $k = 50$, and the minimum size required per sought subtree was set to 0.05 of the total number leaves of G . Parameters for Q_2 were set in the same manner, without bounding the size of sought subtrees and $c_\Delta = c_\Theta = 1$. Figures displaying G , where the top-ranking RSAM nodes are marked with a star, are provided in the supplementary materials, jointly with the corresponding sequences. Also provided is a figure displaying S , σ , and the full multiple alignment used for constructing G .

The highest-scoring RSAM identified for Q_2 roots a subtree with 23 leaf nodes (shown in Fig. S2.C), spanning classes α, β, γ and δ of proteobacteria. The sequences of the genes represented by the leaves of this subtree, denoted the “identified gene set”, were subjected to distinguishing sequence motif search analysis [2], using the full set of input genes, denoted “background gene set”, as background. This analysis identifies an insertion sequence represented by the motif logo given in Fig. S2.E (MEME e-value 4.5e-094). This insertion is validated by the multiple sequence alignment, based on which the gene tree was constructed (see Fig. S2.F and the corresponding multiple alignment file provided in the Supplementary Materials).

Most instances of the *higB* toxin (473/652) in the background gene set have the antitoxin *higA* in their immediate downstream position. In contrast, the instances of *higA* found immediately downstream the genes from the identified gene set are enriched in an additional domain: the Zn-dependent peptidase *ImmA*, belonging to the M78 peptidase family (14/23 vs. 14/652, hypergeometric p-value = 3.27e-23). Homologues of *immA* are found in many mobile genetic elements [7], and it is predicted to participate in a conserved mechanism for regulation of horizontal gene transfer.

Thus, RSAM-Finder identifies a *putative three-component variant of higBA*, consisting of a variant of the *higB* toxin with a conserved insertion sequence, coupled with an *ImmA-higA* fusion protein variant of the *higA* antitoxin. Further analysis of this result may reveal possible functional correlation between the insertion sequence identified within this *higB* toxin variant, and the additional *ImmA* domain characterizing the corresponding *higA* instances.

In contrast to the top-ranking result for pattern Q_2 , the top-ranking result for pattern Q_1 identifies a subtree of G whose leaves encode members of a *two component variant of the higBA TA* (shown in Fig. S2.D). Further analysis of this result may help decipher whether the observed duplications of this invading *higB* variant are tolerated by the invaded genomes due to mere decay of its addictive properties, or due to some mutations conferring selective advantage to the host.

RSAM Discovery in a Beta Lactamase. Beta lactamases are versatile enzymes conferring resistance to the Beta lactam antibiotics, found in a diversity of bacterial sources. Their commonality is the ability to hydrolyze chemical compounds containing a Beta lactam ring [9]. The persistent exposure of bacterial strains to a multitude of Beta lactams has induced dynamic and continuous production and mutation of Beta lactamases in these bacteria, expanding their activity even against the newly developed Beta lactam antibiotics [29]. Thus, an important objective is to identify mutations in Beta lactamase genes conferring adaptation to human and animal hosts.

Among the known classes (A-D) of Beta lactamase, class D (represented by COG2602) is considered to be the most diverse [16]. Thus, we selected COG2602 (622 genes in 543 genomes) to exemplify the colored RSAM pattern Q_3 , where colors represent a binary environmental annotation: human and animal host (219 species) were annotated “red”, while species associated with all other habitats (324 species), such as soil, water and plant, were annotated “black”. Parameters were set as follows: $k = 50$, the minimum size required per sought subtree was set to 0.1 of the total number leaves of G , and $c_{\Delta} = c_{\Theta} = 1$. A figure displaying G , where the top-ranking RSAM node is marked with a star, are given in the supplementary materials. Also provided are the corresponding sequences, a figure displaying the corresponding S , and σ .

Within the top-ranking result for this query, we were interested in the subtree matching the first part of pattern Q_3 (i.e. enrichment in red-to-red HT edges). The gene set represented by the leaves of this subtree, denoted “identified gene set”, was found to be enriched in an additional domain, *BlaR*, a signal transducer membrane protein regulating Beta lactamase production (87/119 in the identified gene set versus 118/622 in the background, p-val = 3.94e-52). The only transcriptional regulator currently known for Beta lactamase genes is the repressor protein *BlaI*, previously predicted to operate in a two-component regulatory system together with *BlaR* in Class A Beta lactamase [1]. The positions adjacent to the instances of the identified gene set in the corresponding genomes were found to be enriched in *BlaI* (70/119 of the identified gene set instances versus 90/622 of the background gene set instances, hypergeometric p-value = 1.11e-41).

In contrast to the identified gene set, the genes represented by the subtree that matches the second part of Q_3 (frequent black HT, S and D events) are not enriched in the *BlaR* domain (2/36), nor is there contextual enrichment in *BlaI* (4/36) in positions immediately adjacent to instances of these genes. Applying RSAM-finder to this data with simpler queries that take into account only enrichment in environmental coloring does not yield this result, nor does the application of RSAM-finder to this data with any part of Q_3 on its own.

The identified gene set for this result spans a wide range of Firmicutes, including both pathogenic (e.g. *staphylococcus*) and non-pathogenic species (e.g. various gut microbes from the Clostridiales order). Homology between *BlaR* receptor proteins and the extra-cellular domain of Class D Beta-lactamases was previously observed [21, 8], mainly in *gram-negative bacteria* (with focus on clinical samples).

Thus, RSAM-finder identifies a putative Beta lactamase system in *gram positive bacteria*, consisting of a *COG2602-BlaR* Beta lactamase-receptor protein and its BlaI family repressor, *predicted to confer adaptation to animal and human host environment*. Further comparative sequence-level analysis [37] may reveal the affinity of this Beta lactamase system to specific Beta lactam drugs.

References

- 1 L E Alksne and B A Rasmussen. Expression of the AsbA1, OXA-12, and AsbM1 beta-lactamases in *Aeromonas jandaei* AER 14 is coordinated by a two-component regulon. *Journal of bacteriology*, 179(6):2006–2013, 1997.
- 2 Timothy L Bailey, Nadya Williams, Chris Misleh, and Wilfred W Li. MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic acids research*, 34(suppl_2):W369–W373, 2006.
- 3 M S Bansal, E J Alm, and M Kellis. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28(12):i283–i291, 2012.
- 4 M S Bansal, E J Alm, and M Kellis. Reconciliation revisited: Handling multiple optima when reconciling with duplication, transfer, and loss. *Journal of Computational Biology*, 20(10):738–754, 2013.
- 5 E Bapteste, M A O’Malley, R G Beiko, M Ereshefsky, J P Gogarten, L Franklin-Hall, F-J Lapointe, J Dupré, T Dagan, Y Boucher, et al. Prokaryotic evolution and the tree of life are two different things. *Biology direct*, 4(1):34, 2009.
- 6 V Berry, F Chevenet, J-P Doyon, and E Jousset. A geography-aware reconciliation method to investigate diversification patterns in host/parasite interactions. *Molecular ecology resources*, 18(5):1173–1184, 2018.
- 7 Baundauna Bose, Jennifer M Auchtung, Catherine A Lee, and Alan D Grossman. A conserved anti-repressor controls horizontal gene transfer by proteolysis. *Molecular microbiology*, 70(3):570–582, 2008.
- 8 C Brandt, S D Braun, C Stein, P Slickers, R Ehricht, M W Pletz, and O Makarewicz. In silico serine β -lactamases analysis reveals a huge potential resistome in environmental and pathogenic species. *Scientific reports*, 7:43232, 2017.
- 9 K Bush. Past and present perspectives on β -lactamases. *Antimicrobial agents and chemotherapy*, 62(10):e01076–18, 2018.
- 10 MA Charleston. Jungles: a new solution to the host/parasite phylogeny reconciliation problem. *Mathematical biosciences*, 149(2):191–223, 1998.
- 11 R Kumar Chaudhary, G Singh, R Naraian, and S Ram. Structural and functional in-silicoanalysis of toxin-antitoxin proteins in persister cells of *pseudomonas aeruginosa*. *Plant Archives*, 18(2):1643–1651, 2018.
- 12 L A David and E J Alm. Rapid evolutionary innovation during an Archaeal genetic expansion. *Nature*, 469(7328):93, 2011.
- 13 B Donati, C Baudet, B Sinimeri, P Crescenzi, and MF Sagot. EUCALYPT: efficient tree reconciliation enumerator. *Algorithms for Molecular Biology*, 10(1):3, 2015.
- 14 J-P Doyon, C Chauve, and S Hamel. Space of gene/species trees reconciliations and parsimonious models. *Journal of Computational Biology*, 16(10):1399–1418, 2009.
- 15 J-P Doyon, S Hamel, and C Chauve. An efficient method for exploring the space of gene tree/species tree reconciliations in a probabilistic framework. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(1):26–39, 2011.
- 16 B A Evans and S GB Amyes. OXA β -lactamases. *Clinical microbiology reviews*, 27(2):241–263, 2014.
- 17 L Huang and D Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

- 18 J Huerta-Cepas, F Serra, and P Bork. ETE 3: reconstruction, analysis, and visualization of phylogenomic data. *Molecular biology and evolution*, 33(6):1635–1638, 2016.
- 19 R Libeskind-Hadas and M A Charleston. On the computational complexity of the reticulate cophylogeny reconstruction problem. *Journal of Computational Biology*, 16(1):105–117, 2009.
- 20 A Marchler-Bauer and S H Bryant. CD-Search: protein domain annotations on the fly. *Nucleic acids research*, 32(suppl_2):W327–W331, 2004.
- 21 O Massidda, M P Montanari, M Mingoia, and P E Varaldo. Borderline methicillin-susceptible *Staphylococcus aureus* strains have more in common than reduced susceptibility to penicillinase-resistant penicillins. *Antimicrobial agents and chemotherapy*, 40(12):2769–2774, 1996.
- 22 D Merkle, M Middendorf, and N Wieseke. A parameter-adaptive dynamic programming approach for inferring cophylogenies. *BMC bioinformatics*, 11(1):S60, 2010.
- 23 S Mukherjee, D Stamatis, J Bertsch, G Ovchinnikova, O Verezhemska, M Isbandi, A D Thomas, R Ali, K Sharma, N C Kyripides, et al. Genomes OnLine Database (GOLD) v. 6: data updates and feature enhancements. *Nucleic acids research*, page gkw992, 2016.
- 24 R Patro and C Kingsford. Predicting protein interactions via parsimonious network history inference. *Bioinformatics*, 29(13):i237–i246, 2013.
- 25 L Poirel, A Carrère, J D Pitout, and P Nordmann. Integron mobilization unit as a source of mobility of antibiotic resistance genes. *Antimicrobial agents and chemotherapy*, 53(6):2492–2498, 2009.
- 26 A-A Popescu, K T Huber, and E Paradis. ape 3.0: New tools for distance-based phylogenetics and evolutionary analysis in R. *Bioinformatics*, 28(11):1536–1537, 2012.
- 27 C Scornavacca, W Paprotny, V Berry, and V Ranwez. Representing a set of reconciliations in a compact way. *Journal of bioinformatics and computational biology*, 11(02):1250025, 2013.
- 28 F Sievers and D G Higgins. Clustal Omega for making accurate alignments of many protein sequences. *Protein Science*, 27(1):135–145, 2018.
- 29 P JM Stapleton, M Murphy, N McCallion, M Brennan, R Cunney, and R J Drew. Outbreaks of extended spectrum beta-lactamase-producing Enterobacteriaceae in neonatal intensive care units: a systematic review. *Archives of Disease in Childhood-Fetal and Neonatal Edition*, 101(1):72–78, 2016.
- 30 M Stolzer, H Lai, M Xu, D Sathaye, B Vernot, and D Durand. Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. *Bioinformatics*, 28(18):i409–i415, 2012.
- 31 J Sukumaran and M T Holder. DendroPy: a Python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571, 2010.
- 32 D Szklarczyk, J H Morris, H Cook, M Kuhn, S Wyder, M Simonovic, A Santos, N T Doncheva, A Roth, P Bork, et al. The STRING database in 2017: quality-controlled protein–protein association networks, made broadly accessible. *Nucleic acids research*, page gkw937, 2016.
- 33 R L Tatusov, M Y Galperin, D A Natale, and E V Koonin. The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic acids research*, 28(1):33–36, 2000.
- 34 TH To, E Jacox, V Ranwez, and C Scornavacca. A fast method for calculating reliable event supports in tree reconciliations via Pareto optimality. *BMC bioinformatics*, 16(1):384, 2015.
- 35 A Tofigh. *Using trees to capture reticulate evolution: lateral gene transfers and cancer progression*. PhD thesis, KTH, 2009.
- 36 A Tofigh, M Hallett, and J Lagergren. Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 8(2):517–535, 2011.
- 37 M Toth, N T Antunes, N K Stewart, H Frase, M Bhattacharya, C A Smith, and S B Vakulenko. Class D β -lactamases do exist in Gram-positive bacteria. *Nature chemical biology*, 12(1):9, 2016.
- 38 L Van Melderen and M S De Bast. Bacterial toxin–antitoxin systems: more than selfish entities? *PLoS genetics*, 5(3):e1000437, 2009.

Jointly Embedding Multiple Single-Cell Omics Measurements

Jie Liu

Department of Computational Medicine and Bioinformatics,
University of Michigan, Ann Arbor, MI, USA
drjieliu@umich.edu

Yuanhao Huang

Department of Computational Medicine and Bioinformatics,
University of Michigan, Ann Arbor, MI, USA
yhao@umich.edu

Ritambhara Singh

Department of Genome Sciences, University of Washington, Seattle, WA, USA
rsingh7@uw.edu

Jean-Philippe Vert 

Google Brain, Paris, France
Centre for Computational Biology, MINES ParisTech, PSL University, Paris, France
jpvert@google.com

William Stafford Noble

Department of Genome Sciences, University of Washington, Seattle, WA, USA
Paul G. Allen School of Computer Science and Engineering,
University of Washington, Seattle, WA, USA
william-noble@uw.edu

Abstract

Many single-cell sequencing technologies are now available, but it is still difficult to apply multiple sequencing technologies to the same single cell. In this paper, we propose an unsupervised manifold alignment algorithm, MMD-MA, for integrating multiple measurements carried out on disjoint aliquots of a given population of cells. Effectively, MMD-MA performs an *in silico* co-assay by embedding cells measured in different ways into a learned latent space. In the MMD-MA algorithm, single-cell data points from multiple domains are aligned by optimizing an objective function with three components: (1) a maximum mean discrepancy (MMD) term to encourage the differently measured points to have similar distributions in the latent space, (2) a distortion term to preserve the structure of the data between the input space and the latent space, and (3) a penalty term to avoid collapse to a trivial solution. Notably, MMD-MA does not require any correspondence information across data modalities, either between the cells or between the features. Furthermore, MMD-MA's weak distributional requirements for the domains to be aligned allow the algorithm to integrate heterogeneous types of single cell measures, such as gene expression, DNA accessibility, chromatin organization, methylation, and imaging data. We demonstrate the utility of MMD-MA in simulation experiments and using a real data set involving single-cell gene expression and methylation data.

2012 ACM Subject Classification Applied computing → Computational biology; Computing methodologies → Dimensionality reduction and manifold learning; Computing methodologies → Unsupervised learning; Computing methodologies → Machine learning algorithms

Keywords and phrases Manifold alignment, single-cell sequencing

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.10

Funding This work was supported by National Institutes of Health award U54 DK107979.



© Jie Liu, Yuanhao Huang, Ritambhara Singh, Jean-Philippe Vert, and William Stafford Noble; licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 10; pp. 10:1–10:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Next-generation sequencing has enabled high-throughput interrogation of many different physical properties of the genome, including the primary DNA sequence but also the expression of messenger RNAs, localized binding of specific factors, histone modifications, nucleosome occupancy, chromatin accessibility, etc. Most of these sequencing assays have been performed on populations of cells. However, such bulk measurements do not allow for easy characterization of systematic or stochastic variations in physical properties of the genome among cells within a given population. Over the past several years, a variety of genomic assays have been modified to allow characterization of single cells. These modifications sometimes involve physically segregating individual cells prior to sequencing, or alternatively involve successive rounds of DNA bar-coding to identify reads derived from single cells. Examples of single-cell genomics assays include single-cell RNA-seq (scRNA-seq) for gene expression [14], single-cell ATAC-seq (scATAC-seq) for chromatin accessibility [3], single-cell Hi-C (scHi-C) for 3D genome organization [10] and single-cell methylation analysis (scMethyl-seq) [13]. In each case, the result is a data set that, compared to a standard, bulk genomic assay, has an additional dimension corresponding to the cells in the sample population.

Single-cell measurements are valuable because they permit a view of the cell-to-cell variation of a given type of physical measurement of the genome. However, such measurements would be even more valuable if multiple different measurements could be obtained from the same individual cell. Such co-assays are feasible, albeit challenging and lower throughput, for pairs of assays, such as scRNA-seq and scATAC-seq [4] or scRNA-seq and scMethyl-seq [2], that measure orthogonal physical properties. However, other pairs of single-cell assays, such as scATAC-seq and scHi-C cannot be paired even in principle, because each assay operates on (and cleaves) the genomic DNA.

In this paper, we propose a manifold alignment algorithm based on the maximum mean discrepancy (MMD) measure, called MMD-MA, which can integrate different types of single-cell measurements. Our MMD-MA algorithm assumes that the cells are drawn from the same initial population – e.g., cells of the same type or a distribution of cell types from the same experimental conditions – but the algorithm does not require any correspondence information either among samples or among the features in different domains. The algorithm makes no parametric assumptions about the forms of the distributions underlying the various measurements. The only assumption is that the distributions share a latent structure with sufficient variability that the MMD term in the optimization can align the distributions. For example, if both underlying distributions are simple isotropic Gaussian distributions, then it will not be possible to reconstruct the relative orientation of the alignment. In practice, visualization of many different single-cell data sets using dimensionality reduction methods such as PCA, t-SNE or UMAP suggest that they commonly exhibit complex structure that, we hypothesize, should allow for alignment across data modalities. In particular, MMD-MA can be applied to many heterogeneous types of single cell measures, including gene expression, DNA accessibility, chromatin organization, methylation, and imaging data. Thus, the algorithm allows us, in principle, to obtain the insights offered by a single-cell co-assay by computationally integrating two or more separate sets of single-cell measurements derived from the same or similar populations of cells. We demonstrate the performance of the algorithm on three simulated data set as well as one real data set consisting of gene expression and methylation profiles of single cells.

2 Methods

2.1 The unsupervised manifold alignment problem

Our goal is to automatically discover a manifold structure that is shared among two or more sets of points that have been measured in different ways, i.e., which mathematically live in different spaces. For simplicity, we describe the case where we have just two types of measurements, though the approach generalizes to any number of input domains. Let the two sets of points be $X^{(1)} = (x_1^{(1)}, x_2^{(1)}, \dots, x_{n_1}^{(1)})$ from $\mathcal{X}^{(1)}$ and $X^{(2)} = (x_1^{(2)}, x_2^{(2)}, \dots, x_{n_2}^{(2)})$ from $\mathcal{X}^{(2)}$. The numbers of data points in the two domains are n_1 and n_2 , respectively. We do not require any correspondence information regarding the measurements across different domains or regarding the samples from different domains. Instead, we assume that both sets of points share a manifold structure, which we aim to discover in an unsupervised fashion.

To ensure the generality of our approach, we frame the optimization using kernels. Hence, we assume that we have a way of calculating similarities between pairs of entities from the same domain, using positive definite kernel functions $k_I : \mathcal{X}^{(I)} \times \mathcal{X}^{(I)} \rightarrow \mathbb{R}$ for $I = 1, 2$. The resulting kernel Gram matrices are denoted by $K_1 \in \mathbb{R}^{n_1 \times n_1}$ and $K_2 \in \mathbb{R}^{n_2 \times n_2}$, where $[K_I]_{ij} = k_I(x_i^{(I)}, x_j^{(I)})$ for $I = 1, 2$ and $1 \leq i, j \leq n_I$. As long as both kernel functions are positive definite, then we are guaranteed that each kernel corresponds to the scalar product operation in some induced feature space, and that there exists a space of functions \mathcal{H}_I , called a reproducing kernel Hilbert space (RKHS), mapping $\mathcal{X}^{(I)}$ to \mathbb{R} endowed with a Hilbert space structure. For example, if the input space \mathcal{X} is a vector space and we take the linear kernel $k(x, x') = x^\top x'$, then the RKHS is made of linear functions of the form $f(x) = w^\top x$, endowed with the norm $\|f\| = \|w\|$. If we take a nonlinear kernel such as the Gaussian RBF kernel $k(x, x') = \exp(-\|x - x'\|^2 / (2\sigma^2))$ with bandwidth $\sigma > 0$, then the RKHS contains nonlinear functions. The use of kernels allows the MMD-MA algorithm to operate in principal on any type of entity – vector, graph, string, etc. – for which a kernel function can be defined.

In order to find a shared structure between the points in $\mathcal{X}^{(1)}$ and $\mathcal{X}^{(2)}$, we propose to learn two mappings $\phi_1 : \mathcal{X}^{(1)} \rightarrow \mathbb{R}^p$ and $\phi_2 : \mathcal{X}^{(2)} \rightarrow \mathbb{R}^p$, so that input data in different spaces are all mapped to the same p -dimensional space \mathbb{R}^p and can be compared in that space. For each $I = 1, 2$, we consider each coordinate of ϕ_I in the RKHS \mathcal{H}_I of the corresponding kernel k_I , i.e., $\phi_I = (\phi_1^{(I)}, \dots, \phi_p^{(I)}) \in \mathcal{H}_I^p$. We then consider each function $\phi_j^{(I)} \in \mathcal{H}_I$ of the form $\phi_j^{(I)}(x) = \sum_{\ell=1}^{n_I} \alpha_{\ell j}^{(I)} k_I(x_\ell^{(I)}, x)$, for any $x \in \mathcal{X}^{(I)}$. This parametrization of $\phi_j^{(I)}$ in terms of $\alpha_{\ell j}^{(I)}$'s always exists by the representer theorem, provided we regularize the optimization problem with the RKHS norm of $\phi_j^{(I)}$, as we explain below. Now, if we denote by α_i the $n_i \times p$ matrix with entries $\alpha_{\ell j}^{(I)}$, then $K_I \alpha_I$ is the $n_I \times p$ matrix where the j -th row (for $j = 1, \dots, n_I$) is the p -dimensional image of $x_j^{(I)}$ by the mapping ϕ_I . In addition, $\alpha_I^\top K_I \alpha_I$ is the $p \times p$ matrix of inner products in the RKHS of the p functions $\phi_1^{(I)}, \dots, \phi_p^{(I)}$, which is for example equal to the $p \times p$ identity matrix I_p when ϕ_I is a projection onto a subspace of dimension p in the RKHS. In order to define MMD-MA, we now discuss the criteria to optimize for α_1 and α_2 in order to discover shared structures between the two views.

2.2 Characterizing the distribution distance in the shared space

Although we do not assume we know the individual correspondence between points in the two domains, or even that such a 1-to-1 mapping exists, we do assume that the two distributions of points are similar in the shared space. Thus, the optimal mapping matrices α_1 and α_2

10:4 Jointly Embedding Multiple Measurements

should make the two mapped sets of points in the shared space, namely $K_1\alpha_1$ and $K_2\alpha_2$, as similar as possible. To specify the distance between the two mapped manifolds in the shared space, we use an MMD term $\text{MMD}(K_1\alpha_1, K_2\alpha_2)^2$, which is a general, differentiable measure of how similar two clouds of points are [7]. Formally, MMD is defined through a positive definite kernel K_M over \mathbb{R}^p through the formula

$$\begin{aligned} \text{MMD}^2 \left(\{u_1^{(1)}, \dots, u_{n_1}^{(1)}\}, \{u_1^{(2)}, \dots, u_{n_2}^{(2)}\} \right) &= \frac{1}{n_1^2} \sum_{i,j=1}^{n_1} K_M(u_i^{(1)}, u_j^{(1)}) \\ &\quad - \frac{2}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} K_M(u_i^{(1)}, u_j^{(2)}) + \frac{1}{n_2^2} \sum_{i,j=1}^{n_2} K_M(u_i^{(2)}, u_j^{(2)}), \end{aligned}$$

where we denote $u_i^{(l)} = \phi_l(x_i^{(l)})$ to simplify notation. In this work, we use a Gaussian RBF kernel for K_M , where the bandwidth parameter σ is a user-specified parameter.

Chwialkowski et al. [5] propose two fast methods (with complexity linear in $n_1 + n_2$) to estimate MMD^2 , both of which are differentiable with respect to the positions of the points. Because the MMD is small when the distributions are similar, MMD-MA aims to minimize MMD^2 with respect to the embeddings.

2.3 The MMD-MA algorithm

Unfortunately, simply minimizing MMD between the two kernels is insufficient. Most notably, we need to ensure that the relationships among data points in the input space is preserved to some extent in the feature space; otherwise, the method may learn very complicated mappings that completely modify the relative positions of cells in order to have them aligned between the two views. For that purpose we introduce a distortion term $\text{dis}(\alpha_I) = \|K_I - K_I\alpha_I\alpha_I^\top K_I^\top\|_2$, which quantifies how the matrix of inner products between points in the original space (quantified by the kernel matrix K_i) differs from the matrix of inner products after mapping in the p -dimensional space. Penalizing $\text{dis}(\alpha_I)$ ensures that the distortion between the data in the original space and the data mapped to the low-dimensional space should be small. In addition, we may wish to ensure that the mappings to \mathbb{R}^p are (almost) projections from the high-dimensional RKHS, which we obtain by adding a penalty term $\text{pen}(\alpha_I) = \|\alpha_I^\top K_I^\top \alpha - I_p\|_2$.

Thus, MMD-MA optimizes, with respect to α_1 and α_2 , the following objective function:

$$\min_{\alpha_1, \alpha_2} \text{MMD}(K_1\alpha_1, K_2\alpha_2)^2 + \lambda_1(\text{pen}(\alpha_1) + \text{pen}(\alpha_2)) + \lambda_2(\text{dis}(\alpha_1) + \text{dis}(\alpha_2)). \quad (1)$$

where

$$\text{pen}(\alpha_I) = \|\alpha_I^\top K_I^\top \alpha - I_p\|_2, \quad (2)$$

and

$$\text{dis}(\alpha_I) = \|K_I - K_I\alpha_I\alpha_I^\top K_I^\top\|_2. \quad (3)$$

2.4 Solving the optimization problem

To find a stationary point of (1) we use a simple gradient descent scheme, and use Adam [9] to adjust the learning rate. Since the optimization problem is not convex, only a local minimum can be expected. Therefore, for a given data set we run the optimization procedure with 100 different random seed values and keep the solution which provides the lowest objective function value.

In practice, solving the optimization requires specifying several hyperparameters. These include the dimensionality p of the shared space, any parameters of the kernel functions K_1 , K_2 and K_M , and the tradeoff parameters λ_1 and λ_2 . In this work, we assume that p and the kernel parameters are user-specified, and we investigate the performance of the algorithm as we vary λ_1 and λ_2 .

3 Related work

We are aware of three other methods that address the unsupervised manifold alignment problem, which we briefly review here.

3.1 The joint Laplacian manifold alignment (JLMA) algorithm

The joint Laplacian manifold alignment (JLMA) algorithm [15] performs manifold alignment by constructing a joint Laplacian across multiple domains and then performing eigenvalue decomposition to find the optimal solution. The joint Laplacian formulation can also be interpreted as preserving similarities within each view and correspondence information about instances across views, which is captured by the joint Laplacian matrix. The loss function is $C(F) = \sum_{i,j} \|\mathbf{F}(i, \cdot) - \mathbf{F}(j, \cdot)\|^2 \mathbf{W}(i, j)$, where the summation is over all pairs of instances from all views. \mathbf{F} is the unified representation of all instances, and the output of the algorithm is the joint adjacency matrix \mathbf{W} . To avoid trivial solutions (i.e., mapping all instances to zero), JLMA includes a constraint $\mathbf{F}'\mathbf{D}\mathbf{F} = I$ where I is an identity matrix. Let $\mathbf{F} = [f_1, f_2, \dots, f_d]$. The optimization problem then becomes

$$\operatorname{argmin}_{\mathbf{F}: \mathbf{F}'\mathbf{D}\mathbf{F}=I} C(\mathbf{F}) = \operatorname{argmin}_{f_1, \dots, f_d} \sum_i f_i' \mathbf{L} f_i + \lambda_i (1 - f_i' \mathbf{D} f_i). \quad (4)$$

The optimal solution is the d smallest nonzero eigenvectors from the generalized eigen decomposition problem.

JLMA can be used in an unsupervised or supervised fashion. In supervised mode, the Laplacian L is given as input. In the unsupervised setting, the key step is to construct the cross-domain Laplacian submatrix of \mathbf{L} . Wang *et al.* use k -NN graphs to characterize local geometry and use the minimum distances from scaled permuted k -NN graphs to construct a cross-view Laplacian submatrix of \mathbf{L} . Thereafter, the rest of the algorithm is the same as the standard JLMA algorithm. Unfortunately, the computational cost of this initial step is quite high, even for small k values. To deal with this problem, Pe *et al.* use a B-spline curve to fit the local geometry and calculate cross-view matching scores from the curves [11]. Thus, in both cases, unsupervised manifold alignment is done via two steps: computing a cross-domain matching score, and identifying the correspondence via Equation 4.

3.2 The generalized unsupervised manifold alignment (GUMA) algorithm

The generalized unsupervised manifold alignment (GUMA) algorithm [6] is another method that does not require any correspondence information a priori. The approach assumes that instances in the two domains (e.g., in our case, two cells measured using different techniques) can be matched to one another in a one-to-one fashion. In particular, the algorithm formulates an optimization problem whose objective function includes a geometry matching term E_s across different domains, a feature matching term E_f , and a geometry preserving term E_p , subject to a 0-1 correspondence matrix \mathbf{F} and feature projections \mathbf{P}_i for each domain

(i.e. domain i). The optimization is performed using alternating minimization, alternating between optimizing \mathbf{F} and \mathbf{P}_i with the other fixed. The algorithm outputs both the instance correspondence between the domains and the feature mapping functions between the domains.

3.3 The manifold alignment generalized adversarial network (MAGAN) algorithm

MAGAN [1] consists of two GANs that learn reciprocal mappings between two domains; i.e., GAN1 learns the mapping from domain 1 to domain 2, and GAN2 learns the mapping from domain 2 to domain 1. Each GAN’s generator takes input in one domain and outputs in the other domain, with the hope that the discriminator in the other domain cannot distinguish the fake samples from true samples. The loss function of the generators consists of three terms. The reconstruction loss term L_r captures the difference between a sample and itself after being mapped to the different domain and then mapped back to the original domain. The discriminator loss term L_d makes sure that the mapped sample in the other domain has a high likelihood to fool the discriminator in that domain. And the correspondence loss L_c forces the learned mapping to agree with some prior correspondence, either “unsupervised correspondence” (e.g., some variables are shared between two domains) or “semi-supervised correspondence” (e.g., some labeled pairs cross two domains). The paper empirically demonstrates that the inclusion of correspondence information greatly improves the performance of the manifold alignment.

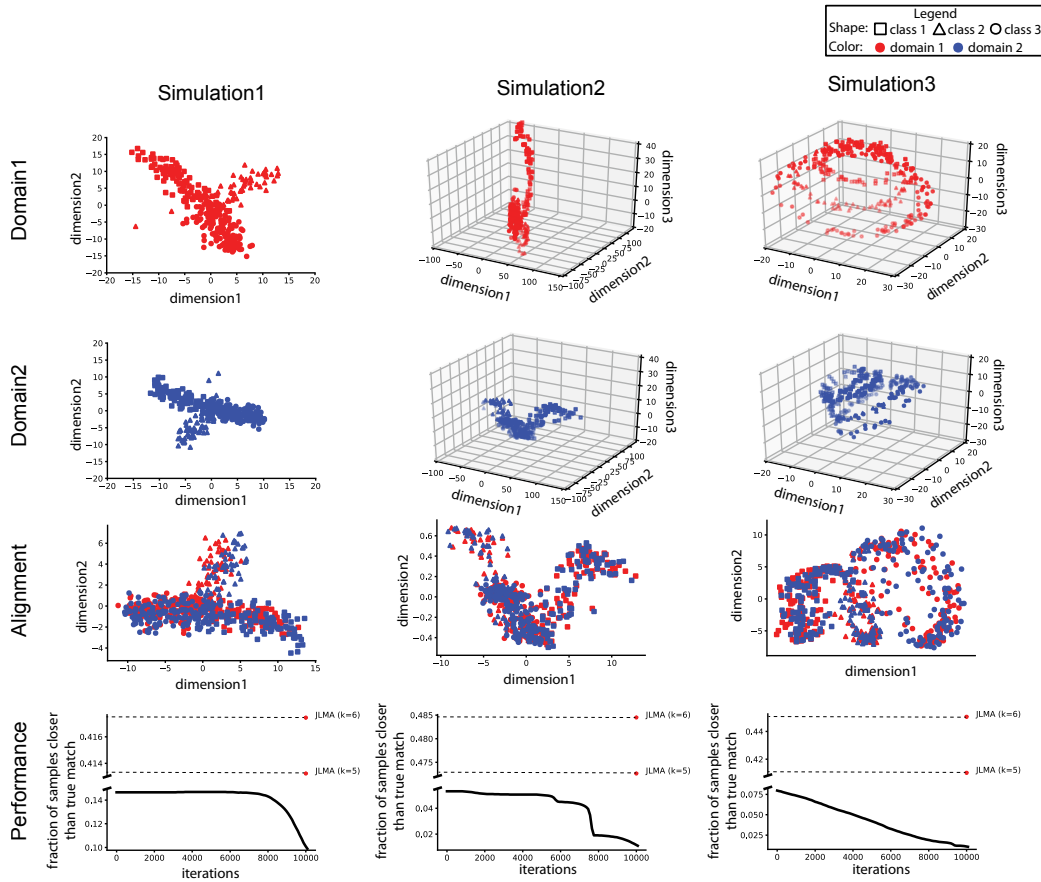
3.4 Comparison of these three algorithms with our algorithm

The MMD term in our formulation only ensures that the two distributions agree globally in the latent space, whereas both JLMA and GUMA have a term that ensures, for each instance, that the local geometry is preserved between domains. This is the difference between *manifold superimposing* [18, 17, 8] and *manifold alignment* discussed in the MAGAN paper [1]. Furthermore, GUMA’s assumption that individual cells can be matched 1-to-1 between the two input domains is not generally true, most obviously when $n_1 \neq n_2$. MAGAN [1] itself does not include a component for identifying a correspondence in an unsupervised fashion, and empirical results from the MAGAN paper suggest that the algorithm may not be useful if there is no known correspondence information between the two domains. When more than three domains must be aligned, the JLMA, GUMA, and our MMD-MA algorithms can be easily extended, whereas the formulation of MAGAN makes such an extension difficult.

4 Results

4.1 Three simulations

To validate the performance of MMD-MA, we generated three simulated data sets, each from a different d -dimensional manifold. The first manifold exhibits a branching structure in two-dimensional space (i.e., $d = 2$) to mimic a branching differentiation situation (first column of Figure 1). The second manifold structure is a nonlinear mapping of the first structure. The branching structure is mapped onto a Swiss roll manifold (second column of Figure 1). Samples in the first domain are mapped from the 2D space of simulation 1 into 3D space such that the three dimensions are $[x_1 \cos(3x_1), x_2, x_1 \sin(3x_1)]$, while samples in the second domain are mapped into 3D space by $[x_1 \sin(2x_1), x_2, x_1 \cos(2x_1)]$. The third manifold is a circular frustum in three-dimensional space (i.e., $d = 3$), which aims to mimic



■ **Figure 1 Three simulation experiments.** The first two rows show the MDS projection of the data points in domain 1 and domain 2, separately. The third row shows the projection of the data points in the shared embedding space. The last row plots the fraction of samples closer than the true match as MMD-MA iterates. Points are included from JLMA when $k=5$ and $k=6$.

the cell cycle superimposed on a linear differentiation process (third column of Figure 1). From each of these d -dimensional manifolds, we simulated $n = 300$ data points, and we refer to the corresponding $n \times d$ data matrix as Z .

For each manifold, we assume that we have two domains, and we generate a $d \times p_1$ mapping matrix T_1 and a $d \times p_2$ mapping matrix T_2 . Each element in the two mapping matrices is sampled from a standard Gaussian distribution. We set the observed data matrix from domain 1 to be ZT_1 and the observed data matrix from domain 2 to be ZT_2 . For example, we set $p_1 = 1000$ and $p_2 = 2000$. We also add Gaussian noise ($\sigma = 0.05$) to each element of the covariates. For all of the simulations, we set the kernel matrices K_1 and K_2 to be the inner product of the z -normalized observed data matrices. For each simulation, we intentionally mis-specify, as input to MMD-MA, the dimensionality of the latent space as $p = 5$, to simulate the scenario in which the true latent dimensionality is unknown a priori.

For all three numerical simulations, we plot the data points in the projected space, namely $K_1\alpha_1^{(o)}$ and $K_2\alpha_2^{(o)}$ (Figure 1). In all three cases, the two domains appear to be aligned correctly in the latent space. To quantify this alignment, we use the known correspondence between points in the two domains as follows. For each point x in one domain, we identify

■ **Table 1 Running time of MMD-MA and JLMA.** Times are provided for the three simulation experiments and the real single cell application.

	Sim 1	Sim 2	Sim 3	Methyl-Expr
MMD-MA	0:53	0:58	0:59	0:10
JLMA, $k = 5$	4:06	4:07	4:20	1:27
JLMA, $k = 6$	26:11	25:56	32:16	2:16

its (true) nearest neighbor in the other domain. We then rank all data points in the learned latent space by their distance from x , and we compute the fraction of points that are closer than the true nearest neighbor. Averaging this fraction across all data points in both domains yields the “average fraction of samples closer than the true match,” where perfect recovery of the true manifold structure yields values close to zero. In all three simulations, the observed fraction of samples closer than the true match decreases monotonically and approaches zero as the MMD-MA algorithm iterates.

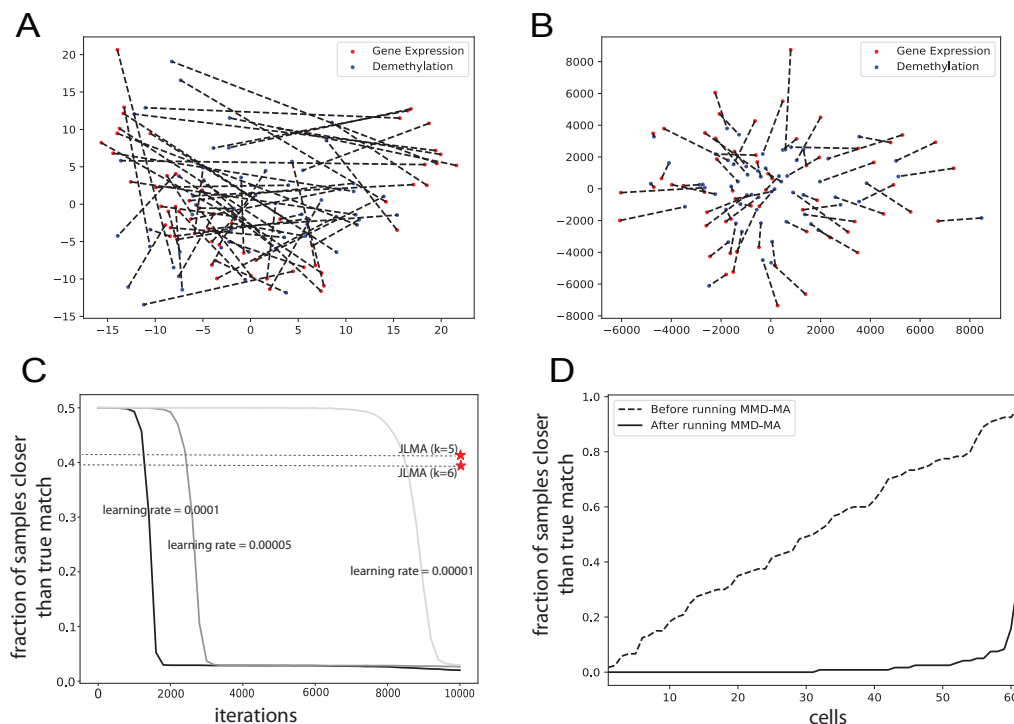
Finally, we attempted to compare the performance of other algorithms on the same simulated data sets. Unfortunately, we had difficulty running the GUMA algorithm using the implementation shared by the authors; hence, we leave GUMA out of our comparison. Similarly, we could not include the MAGAN algorithm because it requires some initial correspondence information, which we are assuming is not available. Consequently, we only compare to the JLMA algorithm as a baseline, using $k = 5$ (the default value) and $k = 6$. We find that, in all three simulations, our MMD-MA algorithm outperforms the baseline JLMA (bottom row of Figure 1).

The running time of MMD-MA is much lower than JLMA using either $k = 5$ or $k = 6$. Timings on an Intel Xeon Gold 6136 CPU at 3.00GHz (Table 1) show that MMD-MA runs under one minute, considerably faster than JLMA.

4.2 Real world application results

In a recent study, gene expression levels and methylation rates were profiled jointly in 61 single cells [2]. We use this co-assay data to validate our method by hiding the correspondence between genes from MMD-MA and then measuring how well the correspondence is recovered. Prior to analysis, we remove the genes where any of the cells have a missing value for either the methylation rate or gene expression. This step leaves, for the 61 cells, 2486 genes with both methylation rate and gene expression measured. We regard gene expression as domain 1 and methylation rate as domain 2. We pretend that we do not know the correspondence information, run our MMD-MA algorithm, and see how well our algorithm can align the two manifolds and recover the cell correspondence. For calculating the similarity kernel matrices K_1 and K_2 , we first perform z -score normalization on the gene expression levels and the methylation rates and then calculate the inner product for the elements in the cell-by-cell similarity matrices. As in the simulations, we embed the two domains into a latent space of dimensionality $p = 5$.

We first plot the Principal Component Analysis (PCA) projection of the single cells based on their gene expression levels and their methylation rates separately (Figure 2A). In this plot, when we connect the two dots corresponding to the same cell, we observe that each cell tends to be projected to two different locations in the latent space. Accordingly, the average fraction of data points closer than the true match is 0.49. We then run MMD-MA algorithm on this dataset and plot the PCA projection of the 61 single cells in terms of gene expression and methylation rate in the shared space recovered by MMD-MA (Figure 2B). In the shared space projection, we connect the two embeddings from different perspectives, and we observe



■ **Figure 2 Results from real world single cell applications.** (A) PCA projection of single cells based on their gene expression levels and their methylation rates separately, with dotted lines connecting the same cell. (B) Projection of the single cells in the shared space from the MMD-MA algorithm, with dotted lines connecting the same cell. (C) The average fraction of samples closer than the true match decreases as MMD-MA iterates. This result is consistent across different learning rates of the optimization. (D) The fraction of samples closer to each cell than its true match is plotted before and after MMD-MA, with the 61 cells in sorted order along the x-axis. For each cell, the average is computed separately for each domain, and then the two values are averaged together. The fraction is high and close to uniformly distributed before running the MMD-MA algorithm and reduces considerably as the algorithm learns the aligned shared space.

that the cells are embedded well in the shared space. Next, we calculate the fraction of samples closer to each cell than its true match in the shared space of dimensionality $p = 5$. This fraction decreases as MMD-MA iterates, reaching 0.024 in the end, and the trend is consistent across different learning rates of the optimization (Figure 2C). An alternative visualization of the per-cell fractions before and after optimization (Figure 2D) further illustrates that the MMD-MA algorithm successfully maps $>50\%$ of the cells closest to their true neighbor in the other domain.

4.3 MMD-MA's performance is robust to variations in hyperparameters

Running the MMD-MA algorithm requires specifying several hyperparameters. We investigated the robustness of the learned embedding relative to variations in these hyperparameters.

As noted previously, in all of our studies the dimensionality p of the latent space has been set to 5 even though the correct number should be $p = 2$ in the first two simulations, $p = 3$ in the third simulation, and is unknown for the Methyl-Expr data set. We observe that MMD-MA algorithm can still align the two manifolds even when the dimensionality parameter p is misspecified.

The trade-off parameters λ_1 and λ_2 determine how much the three terms contribute to

■ **Table 2** Properties and hyperparameters of the experiments.

	Sim 1	Sim 2	Sim 3	Methyl-Expr
number of Samples	300	300	300	61
dimension (Domain 1)	1000	1000	1000	2486
dimension (Domain 2)	2000	2000	2000	2486
λ_1	1e-6	1e-9	1e-5	1e-2
λ_2	1e-2	1e-7	1e-6	1e-6
σ	0.5	0.1	1.2	10000

the overall objective function. In this work, we set these trade-off parameters by monitoring whether the three terms have comparable magnitudes or whether one particular term dominates in the converged solution. We tested eight combinations of these trade-off parameters for each data set (Supplementary Table 3). In each case, we observe that the performance of MMD-MA is almost the same with different choices of trade-off parameters, although some trade-off parameters may lead to a different convergence path (Supplementary Figure 3 A–D).

The bandwidth parameter σ associated with the Gaussian kernel K_M in the MMD term determines how much each data point contributes to its neighborhood in the calculation of the MMD. The σ values used in our experiments are shown in Table 2. We also tested different values of σ and observed that the performance of MMD-MA is quite invariant to them (Supplementary Figure 3 E–H).

5 Discussion

In this paper, we propose an unsupervised manifold alignment algorithm, MMD-MA, for integrating multiple types of single-cell measurements carried out on disjoint populations of single cells drawn from a common source. The key advantage of our MMD-MA algorithm is that it does not require any correspondence information, either between the samples or between the features. In many real-world integration applications, such correspondence information is not available. Another advantage of our MMD-MA algorithm is that it has only weak distributional requirements for the domains to be aligned, namely, that the manifolds exhibit sufficient structure to allow for alignment. This flexibility gives MMD-MA the power to potentially integrate many different types of single cell measures, including gene expression, DNA accessibility, chromatin organization, methylation, and imaging data. Furthermore, the MMD-MA framework can easily be extended to more than two domains, allowing integration of, for example, scRNA-seq, scATAC-seq, and scHi-C of single cells. We have shown that MMD-MA works well in the presence of nonlinear mappings and is robust to the choice of several hyperparameters, including the trade-off parameters, the parameters associated with the MMD term, and the dimensionality of the shared space.

Currently, MMD-MA can be used to align hundreds or thousands of single cells in a reasonable running time. The gradient descent algorithm could be parallelized to save time if multiple cores are available. For future work, we will focus on scaling up the MMD-MA algorithm. Given decreasing sequencing costs, it is likely we will need to apply MMD-MA to align millions of single cells in the future. Running MMD-MA efficiently without storing large kernel matrices in memory will be a crucial issue to solve. A possible solution may rely on random projection [12] or Nystrom approximation [16], which are approximation methods for large-scale kernel matrices.

All of the data sets used in this study, including the original and mapped simulated data

and the Methyl-Expr data set, as well as the corresponding MMD-MA outputs, are available for download from <http://noble.gs.washington.edu/proj/mmd-ma>.

References

- 1 M. Amodio and S. Krishnaswamy. MAGAN: Aligning biological manifolds. In *Proceedings of the International Conference on Machine Learning*, 2018.
- 2 C. Angermueller, S. J. Clark, H. J. Lee, I. C. Macaulay, M. J. Teng, T. X. Hu, F. Krueger, S. A. Smallwood, C. P. Ponting, T. Voet, G. Kelsey, O. Stegle, and W. Reik. Parallel single-cell sequencing links transcriptional and epigenetic heterogeneity. *Nature Methods*, 13:229–232, 2016.
- 3 J. D. Buenrostro, B. Wu, U. M. Litzenburger, D. Ruff, M. L. Gonzales, M. P. Snyder, H. Y. Chang, and W. Greenleaf. Single-cell chromatin accessibility reveals principles of regulatory variation. *Nature*, 523(7561):486–490, 2015.
- 4 J. Cao, D. A. Cusanovich, V. Ramani, D. Aghamirzaie, H. A. Pliner, A. J. Hill, R. M. Daza, J. L. McFaline-Figueroa, J. S. Packer, L. Christiansen, F. J. Steemers, A. C. Adey, C. Trapnell, and J. Shendure. Joint profiling of chromatin accessibility and gene expression in thousands of single cells. *Science*, 361(6409):1380–1385, 2018.
- 5 K. Chwialkowski, A. Ramdas, D. Sejdinovic, and A. Gretton. Fast two-sample testing with analytic representations of probability measures. In *Advances in Neural Information Processing Systems*, pages 1981–1989, 2015.
- 6 Z. Cui, H. Chang, S. Shan, and X. Chen. Generalized unsupervised manifold alignment. In *Advances in Neural Information Processing Systems*, pages 2429–2437, 2014.
- 7 A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.
- 8 T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. arXiv, 2017. [arXiv:1703.05192](https://arxiv.org/abs/1703.05192).
- 9 D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.
- 10 T. Nagano, Y. Lubling, T. J. Stevens, S. Schoenfelder, E. Yaffe, W. Dean, E. D. Laue, A. Tanay, and P. Fraser. Single-cell Hi-C reveals cell-to-cell variability in chromosome structure. *Nature*, 502(7469):59–64, 2013.
- 11 Y. Pei, F. Huang, F. Shi, and H. Zha. Unsupervised image matching based on manifold alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(8):1658–1664, 2012.
- 12 A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.
- 13 S. A. Smallwood, H. J. Lee, C. Angermueller, F. Krueger, H. Saadeh, J. Peat, S. R. Andrews, O. Stegle, W. Reik, and G. Kelsey. Single-cell genome-wide bisulfite sequencing for assessing epigenetic heterogeneity. *Nature Methods*, 11:817–820, 2014.
- 14 F. Tang, C. Barbacioru, Y. Wang, E. Nordman, C. Lee, N. Xu, X. Wang, J. Bodeau, B. B. Tuch, A. Siddiqui, K. Lao, and M. A. Surani. mRNA-Seq whole-transcriptome analysis of a single cell. *Nature Methods*, 6:377–382, 2009.
- 15 C. Wang, P. Krafft, and S. Mahadevan. Manifold alignment. In Y. Ma and Y. Fu, editors, *Manifold Learning: Theory and Applications*. CRC Press, 2011.
- 16 C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 2001.
- 17 Z. Yi, H. Zhang, P. Tan, and M. Gong. DualGAN: Unsupervised Dual Learning for Image-to-Image Translation. In *ICCV*, pages 2868–2876, 2017.
- 18 J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. arXiv, 2017. [arXiv:1703.10593](https://arxiv.org/abs/1703.10593).

A Supplement

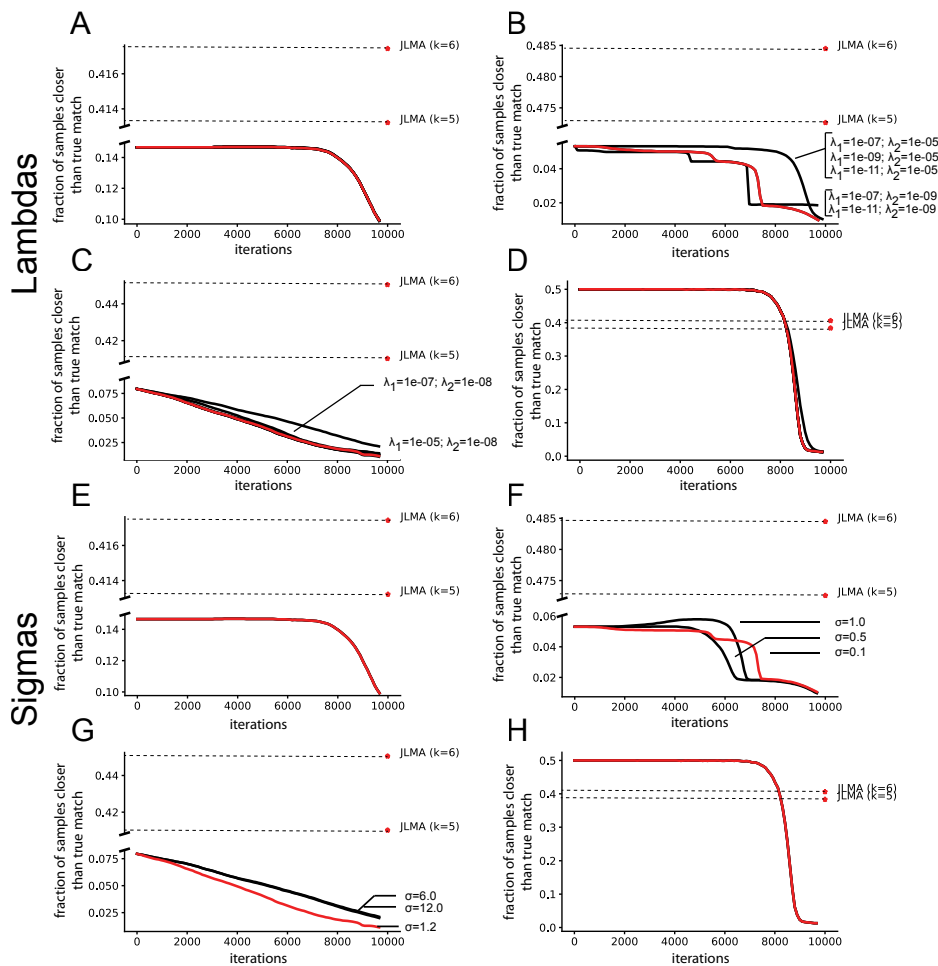


Figure 3 Testing sensitivity to hyperparameters. (A–D) The performance of MMD-MA when the trade-off parameters λ_1 and λ_2 are set differently in the three numerical simulations and the one real-world application, respectively. In each case, eight settings were chosen, and the plot only shows curves (black color) that differ from the one produced by the selected parameters (red color). (E–H) The performance of MMD-MA when σ is set differently in the three numerical simulations and the one real-world application, respectively. Again, only settings that yield results different from the selected results are shown.

■ Table 3 Hyperparameter values investigated for each data set.

Data set	λ_1	λ_2
Simulation 1	1e-04	1
	1e-04	1e-02
	1e-04	1e-04
	1e-06	1
	1e-06	1e-02
	1e-06	1e-04
	1e-08	1
	1e-08	1e-02
	1e-08	1e-04
	1e-08	1e-04
Simulation 2	1e-07	1e-05
	1e-07	1e-07
	1e-07	1e-09
	1e-09	1e-05
	1e-09	1e-07
	1e-09	1e-09
	1e-11	1e-05
	1e-11	1e-07
Simulation 3	1e-11	1e-09
	1e-03	1e-04
	1e-03	1e-06
	1e-03	1e-08
	1e-05	1e-04
	1e-05	1e-06
	1e-05	1e-08
	1e-07	1e-04
	1e-07	1e-06
1e-07	1e-08	
Methyl-Expr	1	1e-04
	1	1e-06
	1	1e-08
	1e-02	1e-04
	1e-02	1e-06
	1e-02	1e-08
	1e-04	1e-04
	1e-04	1e-06
1e-04	1e-08	

Revision Notice

This is a revised version of the eponymous paper that appeared in the proceedings of WABI 2019 (LIPIcs, volume 143, <http://www.dagstuhl.de/dagpub/978-3-95977-123-8>, published in September, 2019), in which Equations 2 and 3 have been changed. In each case, the right-hand side of each equation, in the original publication, was erroneously squared.

Dagstuhl Publishing – March 1, 2022.

Inferring Diploid 3D Chromatin Structures from Hi-C Data

Alexandra Gesine Cauer

Department of Genome Sciences, University of Washington, Seattle, WA, USA
gesine@uw.edu

Gürkan Yardımcı

Department of Genome Sciences, University of Washington, Seattle, WA, USA
gurkan@uw.edu

Jean-Philippe Vert

Google Brain, Paris, France
Centre for Computational Biology, MINES ParisTech, PSL University Paris, France
jpvert@google.com

Nelle Varoquaux 

Department of Statistics, UC Berkeley, CA, USA
nelle.varoquaux@gmail.com

William Stafford Noble

Department of Genome Sciences, University of Washington, Seattle, WA, USA
Paul G. Allen School of Computer Science and Engineering,
University of Washington, Seattle, WA, USA
william-noble@uw.edu

Abstract

The 3D organization of the genome plays a key role in many cellular processes, such as gene regulation, differentiation, and replication. Assays like Hi-C measure DNA-DNA contacts in a high-throughput fashion, and inferring accurate 3D models of chromosomes can yield insights hidden in the raw data. For example, structural inference can account for noise in the data, disambiguate the distinct structures of homologous chromosomes, orient genomic regions relative to nuclear landmarks, and serve as a framework for integrating other data types. Although many methods exist to infer the 3D structure of haploid genomes, inferring a diploid structure from Hi-C data is still an open problem. Indeed, the diploid case is very challenging, because Hi-C data typically does not distinguish between homologous chromosomes. We propose a method to infer 3D diploid genomes from Hi-C data. We demonstrate the accuracy of the method on simulated data, and we also use the method to infer 3D structures for mouse chromosome X, confirming that the active homolog exhibits a bipartite structure, whereas the active homolog does not.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases Genome 3D architecture, chromatin structure, Hi-C, 3D modeling

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.11

Funding WSN acknowledges support from the National Institutes of Health Common Fund 4D Nucleome Program (Grant U54 DK107979). NV was supported by a BIDS fellowship from the Gordon and Betty Moore Foundation (Grant GBMF3834) and by the Alfred P. Sloan Foundation (Grant 2013-10-27).

1 Introduction

The 3D organization of the genome plays an important role in regulating basic cellular functions, including gene regulation [30, 34], differentiation [21, 12], and the cell cycle [27]. Chromosome conformation capture techniques such as Hi-C measure the frequency of



© Alexandra Gesine Cauer, Gürkan Yardımcı, Jean-Philippe Vert, Nelle Varoquaux, and William Stafford Noble;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 11; pp. 11:1–11:13



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

interactions between pairs of loci, thereby allowing a systematic analysis of genome structure. Although Hi-C contact matrices yield valuable insights, modeling and visualizing genome structures in 3D can unveil relationships and higher-order structural patterns that are not apparent in the raw data [13, 37, 27, 24] by providing a humanly interpretable 3D structure, orienting genomic regions relative to various nuclear landmarks, and serving as a framework for integrating other data types [5]. Embedding contact count data in a 3D Euclidean space can also reduce noise in the underlying Hi-C data.

Previous methods to inferring chromatin structure from population Hi-C data fall into one of two broad categories. “Ensemble” approaches create populations of 3D structures that jointly explain the observed Hi-C data [29, 37, 8, 17, 20, 36, 15, 25, 39, 40, 19]. Theoretically, structural ensembles can mimic the heterogeneity of cells in a population. However, these methods are frequently underdetermined because there are often more parameters to estimate for a large population of cells than data points. Ensemble models can also be difficult to validate and interpret. “Consensus” approaches, on the other hand, make the assumption that bulk Hi-C data can be accurately summarized in a single, consensus 3D structure [13, 10, 35, 38, 2, 23, 16]. Modeling a single structure tends to be less computationally demanding than modeling an entire population of structures. Furthermore, the resulting model has the advantage of relatively straightforward visualization and interpretation.

For either ensemble or consensus approaches, a particular challenge is presented by Hi-C data derived from diploid organisms. As in most high-throughput sequencing experiments, a typical Hi-C experiment does not produce phased data; that is, the data does not distinguish between allelic copies. Thus, an observation of a single Hi-C contact between loci i and j corresponds to one of four possible events: either copy of locus i coming into contact with either copy of locus j . Any 3D inference method that aims to model diploid genomes must accurately account for this allelic uncertainty.

A variety of strategies have been developed to account for diploidy in Hi-C 3D models. In general, ensemble models face less of a challenge on this front, since the two allelic copies can be treated like additional members of the ensemble. Among consensus methods, by far the most common approach is to assume that the two homologous copies of a given chromosome share the same 3D structure [38, 23, 41] and then to model each chromosome separately.

We are aware of only three previous attempts to model diploidy in non-ensemble methods. Previously, we described an extension of our PASTIS software to handle the near-haploid cell line KBM7 [3]. We proposed to infer jointly the distribution of contact counts between homologs and the 3D structures by maximizing a constrained and relaxed likelihood. However, this relaxation is unsatisfying, as it yields non-integer counts modeled as random Poisson variables. More recently, two separate research groups have developed methods for modeling diploid genomes from single-cell data [7, 33]. However, these methods cannot be directly applied to bulk Hi-C data, which is much more widely available.

In this work, we propose a method to infer diploid consensus 3D models from Hi-C data. Our approach builds upon PASTIS [38], which infers 3D models by using a Poisson model of Hi-C counts coupled with a simple biophysical model of polymer packing. The key idea of extending PASTIS to infer diploid genomes is to explicitly model the uncertainty of allelic assignments for each observed read. We consider two distinct settings: the more challenging setting where the data is fully ambiguous, and the setting where a subset of the reads can be mapped to a single parental allele. To assist in inference, we incorporate several constraints into our objective function, reflecting our prior knowledge of genome architecture. Through extensive simulations, we demonstrate that our approach can successfully model two distinct homologous chromosome structures, given a sufficient number of reads, even

when the data is fully ambiguous. We also apply our approach to real Hi-C data derived from a first generation (F1) cross of two divergent mouse strains (F121 and *Castaneus*). The resulting diploid model of the X chromosome exhibits the expected “superdomain” structure [11], and is quite distinct from the inferred structure of the inactive X.

2 Method

Hi-C experiments involve sequencing pairs of interacting DNA fragments. Specifically, cells are cross-linked, DNA is digested using a restriction enzyme, and interacting fragments are then ligated together. Fragments are subsequently sequenced through paired-end sequencing, and each mate is associated with one interacting locus. Hi-C data can then be summarized in a symmetric $n \times n$ contact count matrix C , where each row and column corresponds to a genomic locus and each matrix entry c_{ij} to the number of time those two loci have been observed to interact.

For diploid organisms, reads from homologous chromosomes cannot be distinguished from one another, and the resulting Hi-C matrix aggregates contact counts from homologous chromosomes into a single Hi-C matrix (Figure 1). The challenge of inferring diploid structures from Hi-C data lies in disambiguating the contact counts from the two homolog chromosomes. We call these aggregated counts “ambiguous,” and denote by C^A the corresponding contact count matrix. If the parental genomes are known *a priori*, then a small proportion of reads can be mapped to each haplotype: contact counts from the two homolog chromosomes can be disambiguated based on heterozygous positions, yielding a single-allele Hi-C count matrix [30, 11]. We refer to these counts as “unambiguous” and denote the corresponding matrix by C^U . On the other hand, if only one mate can be mapped uniquely to one of the homologous chromosomes, then the contact count is only partially disambiguated between the two homologs. We refer to these as “partially ambiguous” contact counts, and we denote the corresponding matrix by C^P .

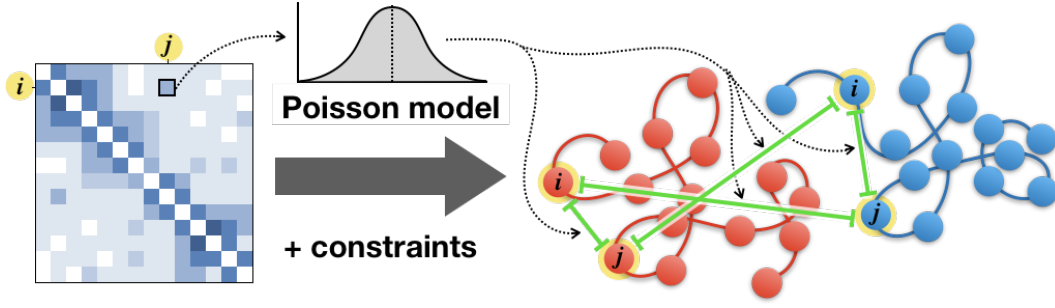
We model chromosomes as m evenly-spaced beads, and we denote by $\mathbf{X} = (x_1, \dots, x_m) \in \mathbb{R}^{3 \times m}$ the coordinate matrix of the structure. The variable m denotes the total number of beads in the genome, and $x_\ell \in \mathbb{R}^3$ corresponds to the 3D coordinate of the ℓ -th bead. In the case of a haploid structure, the number of beads corresponds to the number of rows and columns in the contact count matrix C : $n = m$.

2.1 Inferring haploid structures with a Poisson model

Before we turn to inferring diploid structures, let us first review the approach proposed by PASTIS [38] to infer haploid 3D structures from a bulk Hi-C contact map \mathbf{C} . PASTIS models the interaction frequency between genomic loci i and j as a random independent Poisson variable, where the intensity of the Poisson distribution is a decreasing function f of the Euclidean distance between the two beads (d_{ij}). Leveraging relationships found from studying biophysical properties of DNA as a polymer, PASTIS sets this function as follows: $f(d_{ij}) \sim d_{ij}^\alpha$, $\alpha < 0$. The α parameter can be set using prior knowledge (e.g., $\alpha = -3$), or inferred jointly with the 3D structure. Inference is thus performed by maximizing the likelihood of the following Poisson model:

$$c_{ij} \sim \text{Poisson}(b_i b_j \beta d_{ij}^\alpha), \quad (1)$$

where β scales for the total number of contacts in the matrix (“coverage”), and b_i and b_j are locus-specific biases that are estimated using a standard procedure [18].



■ **Figure 1** Inferring 3D structure using ambiguous diploid data. Each observed count (left) corresponds to a sum of four pairs of genomic loci (right). The Poisson model must be adjusted to account for this ambiguity.

Our strategy to infer diploid structures builds upon this approach. Note that inferring a diploid structure from “unambiguous” contact counts C^U is similar to inferring a haploid structure from a classic Hi-C experiment, with the only difference concerning the biases, which are computed using all contact counts available per locus.

2.2 Modeling contact counts of diploid structures with a Poisson model

We propose to extend PASTIS to diploid genomes by leveraging the properties of each type of Hi-C contact map: ambiguous, partially ambiguous, and unambiguous. Let us first take a closer look at the common scenario, where the data is fully ambiguous.

For a given ambiguous contact count matrix C^A , each observed contact count c_{ij}^A between a given pair of loci (i, j) corresponds to the sum of four different unambiguous contact counts (Figure 1):

$$c_{ij}^A = \sum_{\ell: \Phi(\ell)=i} \sum_{p: \Phi(p)=j} c_{\ell p}^U, \quad (2)$$

where $\Phi: [1, n] \rightarrow [1, m]$ is the mapping that associates bead ℓ with locus i . Leveraging the property that the sum of i Poisson variables of intensities λ_i is a Poisson variable of intensity $\sum_i \lambda_i$, we model the interaction count as

$$c_{ij}^A \sim \text{Poisson} \left(b_i b_j \beta^A \sum_{\ell: \Phi(\ell)=i} \sum_{p: \Phi(p)=j} d_{\ell p}^\alpha \right), \quad (3)$$

where m is the number of loci, n is the number of beads, $d_{\ell, p}$ is the Euclidean distance between beads ℓ and p , and β^A is a scaling factor determined by the coverage of the ambiguous contact count matrix.

Similarly, for a given partially ambiguous contact count matrix C^P , each observed contact count c_{ij}^P between a given pair of loci corresponds to the sum of two unambiguous contact counts, and is modeled by the interaction frequency of two pairs of loci.

$$c_{ij}^P \sim \text{Poisson} \left(b_i b_j \beta^P \sum_{\ell: \Phi(\ell)=i} d_{\ell j}^\alpha \right) \quad (4)$$

β^P is a scaling factor determined by coverage of the partially ambiguous contact count matrix.

We can thus cast the 3D structure inference as maximizing the log-likelihood

$$\begin{aligned}
\max_{\mathbf{X}} \mathcal{L}(X) &= \mathcal{L}_U(X) + \mathcal{L}_P(X) + \mathcal{L}_N(X) \\
&= \sum_{1 \leq i < j \leq m} c_{ij}^U \log(b_i b_j \beta^U d_{ij}^\alpha) - b_i b_j \beta^U d_{ij}^\alpha + \\
&\quad \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq n, i \neq j} c_{ij}^P \log \left(b_i b_j \beta^P \sum_{\ell: \Phi(\ell)=i} d_{\ell j}^\alpha \right) - b_i b_j \beta^P \sum_{\ell: \Phi(\ell)=i} d_{\ell j}^\alpha + \\
&\quad \sum_{1 \leq i < j \leq n} c_{ij}^A \log \left(b_i b_j \beta^A \sum_{\ell: \Phi(\ell)=i} \sum_{p: \Phi(p)=j} d_{\ell p}^\alpha \right) - b_i b_j \beta^A \sum_{\ell: \Phi(\ell)=i} \sum_{p: \Phi(p)=j} d_{\ell p}^\alpha
\end{aligned} \tag{5}$$

Note that this approach holds for polyploid genomes in addition to diploid genomes.

2.3 Incorporating prior knowledge

Because the resulting optimization is challenging, we add two constraints that reflect our prior knowledge about chromatin 3D structure: two neighboring beads should not be too far apart from one another, and homologs of most organisms occupy distinct territories [33, 32, 4, 28].

The first constraint maintains bead chain connectivity by minimizing the variance in the distance between neighboring beads:

$$h_1(\mathbf{X}) = (m - 1) \frac{\sum_{\ell=1}^{m-1} (d_{\ell, \ell+1})^2}{\left(\sum_{\ell=1}^{m-1} d_{\ell, \ell+1} \right)^2} - 1 \tag{6}$$

where ℓ and $\ell + 1$ are on the same chromosome. This type of constraint has been used previously in Simba3D [31].

The second constraint aims to disentangle the structures of the two homologs, and operates on the distance between homolog centers of mass:

$$h_2(\mathbf{X}) = \sum_c \left(\max \left(0, \left(r_c - \left\| \frac{1}{\text{card}(c_A)} \sum_{j \in c_A} \mathbf{X}_j - \frac{1}{\text{card}(c_B)} \sum_{p \in c_B} \mathbf{X}_p \right\| \right) \right)^2 \right), \tag{7}$$

where c denotes the chromosome, c_A and c_B denote the set of beads associated to the two homologs of chromosome c , and r_c is a predefined scalar that increases relative to the space between the two homologs of chromosome c . We note that such a penalty may be interpreted as a log-prior in a Bayesian setting, where the distance between homolog centers of mass of chromosome c is *a priori* normally distributed with mean r_c .

When unambiguous data is available, the values of r_c may be estimated via the distances between homolog centers of mass in an extremely coarse-grained structure inferred from unambiguous data alone. Alternatively, when unambiguous data is not available, r_c may be estimated as the mean distance between chromosome centers of mass in a coarse-grained structure inferred from ambiguous data, since this distance is expected to be similar to that between homologs.

We penalize the likelihood in Equation 5 and solve the following optimization problem via L-BFGS-B, a widely used quasi-Newton method[6]:

$$\max_{\mathbf{X}} \mathcal{L}(X) + \lambda_1 h_1(X) + \lambda_2 h_2(X), \tag{8}$$

where λ_1 and λ_2 are penalization parameters, the values of which were chosen via a grid search. A version of PASTIS that implements the diploid inference approach is available at <https://github.com/hiclib/pastis>.

2.4 Data

2.4.1 Simulated Hi-C data

To validate our approach, we generated 10 simulated genomes with coverage, number of beads, and ratios of disambiguated contact counts corresponding to those of Hi-C data from the mouse Patski cell line (described in Section 2.4.2) at 500 kb resolution. We also generated additional sets of 10 simulated genomes with the same number of beads, varying the proportion of ambiguous, unambiguous, and partially ambiguous contact counts.

To simulate “true” structures, we applied a random walk algorithm. This algorithm places beads successively along each chromosome, constraining each bead to lie within a given distance of the previous bead, provided the new bead does not overlap with any of the previously placed beads and that the entire homolog fits within a sphere of a predefined radius. We then derive unambiguous counts using the following model:

$$c_{ij} = \text{Poisson}(\beta d_{ij}^\alpha), \quad (9)$$

where $\alpha = -3$, corresponding to a previously used theoretical exponent for the contact-to-distance transfer function [38]. To convert unambiguous counts to ambiguous or partially ambiguous counts, we summed contacts from the appropriate pairs of loci. In all experiments, we simulated a 343-bead chromosome with 9.3×10^6 reads, which corresponds to the number of beads and reads in the real data we examined. All simulated Hi-C data used for this project is available at <https://noble.gs.washington.edu/proj/diploid-pastis/>.

2.4.2 Real Hi-C data

We applied our method to publicly available in situ DNase Hi-C of Patski fibroblast mouse kidney cells [11]. This line was derived from F1 female embryos, obtained by mating a BL6 female with a *Spretus* male. The BL6 female had an *Hprt* mutation, so hypoxanthine-aminopterin-thymidine medium was used to select for cells with X chromosome inactivation on the maternal allele. All real Hi-C data used for this project is available at <https://noble.gs.washington.edu/proj/diploid-pastis/>.

2.5 Structure similarity measures

We use the following quantitative measures of similarity between 3D structures to determine the quality of structures inferred from simulated data and assess the stability of chromatin structures across biological replicates.

Root mean square deviation (RMSD) is a common way of comparing two three dimensional structures described by their coordinates \mathbf{X} , $\mathbf{X}' \in R^{3 \times m}$. RMSD is defined as

$$RMSD = \min_{\mathbf{X}^*} \sqrt{\frac{\sum_{i=1}^m (\mathbf{X}_i - \mathbf{X}_i^*)^2}{m}}, \quad (10)$$

where \mathbf{X}^* is obtained by translating, rotating, and rescaling \mathbf{X}' ($\mathbf{X}^* = s\mathbf{R}\mathbf{X}' - \mathbf{t}$ where $\mathbf{R} \in R^{3 \times 3}$ is a rotation matrix, $\mathbf{t} \in R^3$ is a translation vector, and s is a scaling factor). RMSD values are computed independently on each homolog of each chromosome and summed.

Distance error [38] assesses the similarity between two distance matrices. This measure assigns more weight to long distances than RMSD. It is given by

$$distError = \min_{\mathbf{X}^*} \sqrt{\frac{\sum_{i \in \gamma} (d_i(\mathbf{X}) - d_i(\mathbf{X}^*))^2}{m}}, \quad (11)$$

where γ is a set of distances of interest (e.g., intra-chromosomal distances). The structure \mathbf{X}^* is obtained by rescaling \mathbf{X}' ($\mathbf{X}^* = s\mathbf{X}'$ where s is a scaling factor). To distinguish discrepancies in intra-chromosomal structure from those affecting the relative orientation of each pair of homologs or the relative orientation of different chromosome pairs, we compute distance error in two ways. Intra-chromosomal distance error is computed separately for each homolog of each chromosome, and γ encompasses distances between all beads of the given homolog. Inter-homolog distance error is computed separately for chromosome pair, and γ encompasses distances connecting all beads of two different homologs of a given chromosome. For both measures, values are summed for all chromosomes.

3 Results

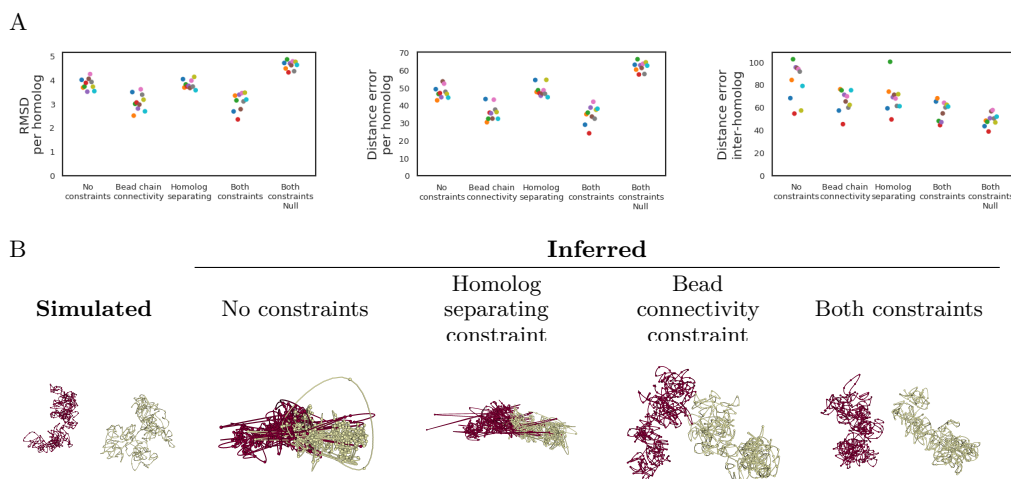
3.1 Constraints improve ambiguous inference

First, we assessed the accuracy of our method on simulated datasets (Section 2.4.1) using ambiguous data alone, with and without our proposed constraints. Because of the lack of disambiguated contact counts, we expected this inference task to be difficult. Our results demonstrated that the two sets of constraints – bead connectivity and homolog separation – are necessary for successful inference. In the absence of the constraints, ambiguous inference performed poorly (Figure 2). Specifically, inferred homolog structures overlapped one another, and adjacent beads sometimes had large gaps between one another. The homolog separation constraint (Equation 7) and the bead connectivity constraint (Equation 6) were specifically designed to address these problems. Therefore, we repeated the inference with each constraint individually and the two constraints in combination. In this experiment, we compared results generated with and without each constraint at the optimal λ values ($\lambda_1 = 10^8$ and $\lambda_2 = 10^{10}$, respectively). The results showed that RMSD and distance error are lowest when both constraints were incorporated (Figure 2), and error scores obtained from structures inferred with both constraints were significantly lower than those obtained from structures inferred without constraints (pairwise t-test, Bonferroni corrected p -value < 0.05 , Supplementary Table 1). 3D structures produced with the constraints had fewer large gaps between neighboring beads and exhibited distinct territories for the two homologs. With both constraints, optimization on a heterogeneous CPU cluster running at 1.90-2.4 GHz took an average of two hours to converge (averaged over 50 jobs).

As an additional control for the previous experiment, we sought to confirm that the Poisson model for ambiguous diploid contact counts improved inference above what could be attained by the constraints alone. Accordingly, we compared results generated with simulated ambiguous data to “null” structures, which were inferred with the same initialization and constraints but without the Poisson model. Both measures of intra-homolog similarity showed a clear improvement when the Poisson model was incorporated in inference (Figure 2). On the other hand, the inter-homolog distance error did not improve with the addition of the Poisson model, suggesting that the constraints are the primary influence in orienting the homologs relative to one another.

3.2 Best results obtained by incorporation of disambiguated data

We expected that more accurate structure inference could be achieved using data where one or more ends of each contact count was disambiguated, relative to fully ambiguous data. We also expected that unambiguous data, in which both ends of each contact count are disambiguated, would yield better models than partially ambiguous data, in which only



■ **Figure 2 Constraints improve ambiguous inference.** The simulated data consists of a single diploid chromosome with 9.3×10^6 reads and 343 beads, the size of which corresponds to mouse chromosome X at 500 kb resolution. (A) The quality of the inferred structure, as measured by three different error scores (y-axis), improves upon application of the bead connectivity constraint ($\lambda_1 = 10^8$) and the homolog separating constraint ($\lambda_2 = 10^{10}$). Best results are seen when both constraints are applied simultaneously. Each point corresponds to a single inferred structure, and colors indicate the simulated true structure from which counts were derived. “Null” indicates inference performed without the Poisson model. Corresponding p -values are in Supplementary Table 1. (B) A simulated chromosome is shown alongside inferred versions of the same chromosomes using various strategies. Each panel also lists the RMSD and distance error associated with the given structure, relative to the true structure.

one end of each contact count is disambiguated. To test these hypotheses, we simulated partially ambiguous data and unambiguous data. Across all similarity measures, inference with unambiguous data performed best, and inference with ambiguous data performed worst, as expected (Figure 3). Partially ambiguous contacts seem especially beneficial in inference of intra-homolog structure, since intra-homolog RMSD and distance error of structures inferred with partially ambiguous counts was significantly lower than intra-homolog RMSD and distance error of structures inferred with ambiguous counts (pairwise t-test, Bonferroni corrected p -value < 0.05 , Supplementary Table 2).

3.3 Inference successfully identifies the superdomain structure of the inactive X chromosome

Deng et al. [11] previously showed that inactive X chromosome adopts a bipartite structure with two large superdomains, whereas the active homolog does not. We sought to validate our approach by inferring the mouse X chromosome structure and examining the degree to which each homolog exhibits a bipartite structure. Bipartite structure was assessed via the “bipartite index,” which refers to the ratio of the frequency of counts within each superdomain to those between superdomains [11]. To determine the bipartite index of an inferred 3D structure, we induced counts by applying the biophysical model used during inference (Equation 9) to the distances between beads.

We inferred a 3D structure for the mouse X chromosome at 500 kb resolution and computed the bipartite index at each bin along the chromosome. The boundary between superdomains of the inactive X chromosome has been shown to center at position 72.8–72.9 Mb (mm9,

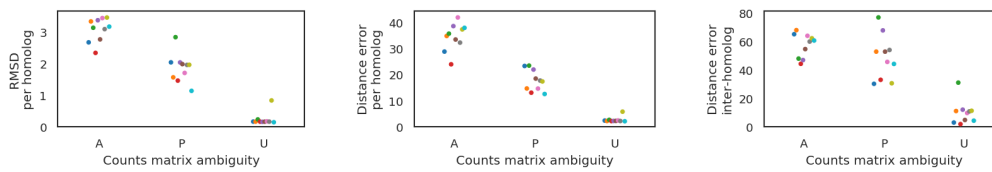


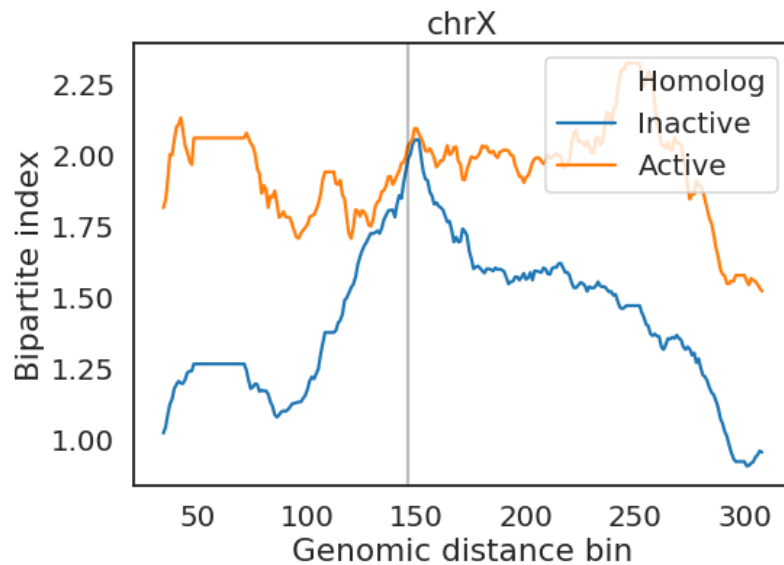
Figure 3 Inference with ambiguous and disambiguated data. The simulated data consists of a single chromosome with 9.3^6 reads and 343 beads, the size of which corresponds to mouse chromosome X at 500 kb resolution. The quality of the inferred structure, as measured by three different error scores (y-axis), improves when one or both ends of a contact are disambiguated, and best results are seen in the latter case. “A” indicates ambiguous data, “U” indicates unambiguous data, and “P” indicates partially ambiguous data. Each point corresponds to a single inferred structure, and colors indicate the simulated true structure from which counts were derived. Corresponding p -values are in Supplementary Table 2.

corresponding to bead 146 in our structure) [11]. In our analysis, the bipartite index of the inferred inactive homolog exhibited a prominent peak around position 75 Mb (corresponding to bead 150), whereas the active homolog only had a relatively small peak at this position (Figure 4). This observation suggests that the inference method has successfully recovered this known feature of the mouse inactive X chromosome.

4 Discussion

Three-dimensional structural inference of diploid genomes is a challenging problem because most Hi-C data is inherently ambiguous and does not discriminate between contact counts from the two homologs of a given chromosome. Even in the rare cases when parental genotype information is available, only a minority of reads can be disambiguated. As a consequence, many inference methods have modeled a single structure per diploid chromosome [38, 23, 41]. Such an approach assumes that the two homologous copies of a given chromosome have the same 3D structure and prevents structural inference of more than one chromosome at a time. Because of these limitations, the degree of structural similarity between homologous autosomes is not currently well understood.

In this work, we show how to carry out true diploid structural inference by modifying the objective function of PASTIS, a previously published haploid inference method [38]. PASTIS models each contact count via a Poisson distribution of a biophysical model between pairwise distances connecting the corresponding beads. In this work, we model each diploid contact count as the sum of biophysical models between all possible distances between the corresponding bead on each homolog. We combine this modified Poisson model with two constraints that limit the scope of possible solutions to more realistic structures. One constraint enforces even spacing of beads along the chain of the chromosome, and the other serves to spatially separate homologs. Using simulations, we show that the most accurate structures are obtained by inferring with the Poisson model in conjunction with both constraints. We note that the homologs of our simulated structures occupy distinct territories. While this is the case for many organisms, there are some exceptions [26, 33, 32, 4, 28]; therefore, the weight assigned to the homolog-separating constraint should be tuned for each organism based on prior knowledge. These analyses were performed at the relatively coarse resolution of 500 kb, and the relationship between resolution, coverage, computational cost, and accuracy of this method remains unexplored.



■ **Figure 4 Bipartite structure of the mouse inactive X chromosome.** The bipartite index (y-axis) at each genomic distance bin (x-axis) for the active (orange) and inactive (blue) homologs of the mouse X chromosome. The black line corresponds to the known boundary between superdomains of the inactive homolog at bin 146.

A limitation to this method involves the distribution of contact count data, which may be better fit by a negative binomial model than a Poisson model [9]. Unfortunately, our method of diploid inference relies on a specific property of Poisson models, namely, that the sum of multiple Poisson variables is also a Poisson variable. Another caveat involves the biophysical model used during inference (Equation 9), which may not accurately capture the relationship between contact counts and pairwise distances in all situations. For example, this relationship may vary depending on the organism, resolution, genomic distance range, and cell cycle status [42, 1, 2, 22, 14]. We also note that in the completely ambiguous case, it is possible that the inferred homologs represent different subpopulations within the sample rather than separating the two haplotypes.

We envision several ways in which diploid PASTIS could be further improved. First, diploid PASTIS could allow for joint estimation of the α parameter of the biophysical model alongside the 3D structure, as is possible for haploid PASTIS. Second, results could potentially be improved by incorporating a multiscale optimization strategy, in which a high-resolution structure is inferred in a stepwise fashion through multiple rounds of inference with gradually increasing resolution. Similarly, inference of the whole genome may be improved by a stepwise approach where each chromosome is first inferred individually before being placed in the context of the whole genome.

References

- 1 F. Ay, T. L. Bailey, and W. S. Noble. Statistical confidence estimation for Hi-C data reveals regulatory chromatin contacts. *Genome Research*, 24:999–1011, 2014.
- 2 F. Ay, E. M. Bunnik, N. Varoquaux, S. M. Bol, J. Prudhomme, J.-P. Vert, W. S. Noble, and K. G. Le Roch. Three-dimensional modeling of the *P. falciparum* genome during the erythrocytic cycle reveals a strong connection between genome architecture and gene expression. *Genome Research*, 24:974–988, 2014.

- 3 F. Ay, T. H. Vu, M. J. Zeitz, N. Varoquaux, J. E. Carette, J.-P. Vert, A. R. Hoffman, and W. S. Noble. Identifying multi-locus chromatin contacts in human cells using tethered multiple 3C. *BMC Genomics*, 16(121), 2015.
- 4 A. Bolzer, G. Kreth, I. Solovei, D. Koehler, K. Saracoglu, C. Fauth, S. Müller, R. Eils, C. Cremer, M. R. Speicher, and T. Cremer. Three-dimensional maps of all chromosomes in human male fibroblast nuclei and prometaphase rosettes. *PLOS Biology*, 3(5):e157, 2005.
- 5 E. M. Bunnik, K. B. Cook, N. Varoquaux, G. Batugedara, J. Prudhomme, A. Cort, L. Shi, C. Andolina, L. S. Ross, D. Brady, D. A. Fidock, F. Nosten, R. Tewari, P. Sinnis, F. Ay, J.-P. Vert, W. S. Noble, and K. G. Le Roch. Changes in genome organization of parasite-specific gene families during the *Plasmodium* transmission stages. *Nature Communications*, 15(9):1910, 2018.
- 6 R. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. doi:10.1137/0916069.
- 7 S Carstens, M Nilges, and M Habeck. Inferential structure determination of chromosomes from single-cell Hi-C data. *PLOS Computational Biology*, 12(12):e1005292, 2016.
- 8 S Carstens, M Nilges, and M Habeck. Bayesian inference of chromatin structure ensembles from population Hi-C data. *bioRxiv*, page 493676, 2018.
- 9 M. Carty, L. Zamparo, M. Sahin, A. Gonzalez, R. Pelosoof, O. Elemento, and C. S. Leslie. An integrated model for detecting significant chromatin interactions from high-resolution Hi-C data. *Nature Communications*, 8:15454, 2017.
- 10 J. Dekker, K. Rippe, M. Dekker, and N. Kleckner. Capturing chromosome conformation. *Science*, 295(5558):1306–1311, 2002.
- 11 X. Deng, W. Ma, V. Ramani, A. Hill, F. Yang, F. Ay, J. B. Berletch, C. A. Blau, J. Shendure, Z. Duan, W. S. Noble, and C. M. Disteche. Bipartite structure of the inactive mouse X chromosome. *Genome Biology*, 16:152, 2015.
- 12 J R Dixon, I Jung, S Selvaraj, Y Shen, J E Antosiewicz-Bourget, A Y Lee, Z Ye, A Kim, N Rajagopal, W Xie, et al. Chromatin architecture reorganization during stem cell differentiation. *Nature*, 518(7539):331, 2015.
- 13 Z. Duan, M. Andronescu, K. Schutz, S. McIlwain, Y. J. Kim, C. Lee, J. Shendure, S. Fields, C. A. Blau, and W. S. Noble. A three-dimensional model of the yeast genome. *Nature*, 465:363–367, 2010.
- 14 G. Fudenberg and L. A. Mirny. Higher-order chromatin structure: bridging physics and biology. *Curr Opin Genet Dev.*, 22(2):115–124, 2012.
- 15 L Giorgetti, R Galupa, E P Nora, T Pilot, F Lam, J Dekker, G Tiana, and E Heard. Predictive polymer modeling reveals coupled fluctuations in chromosome conformation and transcription. *Cell*, 157(4):950–963, 2014.
- 16 Y Hirata, A Oda, K Ohta, and K Aihara. Three-dimensional reconstruction of single-cell chromosome structure using recurrence plots. *Scientific reports*, 6:34982, 2016.
- 17 M. Hu, K. Deng, Z. Qin, J. Dixon, S. Selvaraj, J. Fang, B. Ren, and J. S. Liu. Bayesian inference of spatial organizations of chromosomes. *PLOS Comput Biol*, 9(1):e1002893, 2013.
- 18 M. Imakaev, G. Fudenberg, R. P. McCord, N. Naumova, A. Goloborodko, B. R. Lajoie, J. Dekker, and L. A. Mirny. Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nature Methods*, 9:999–1003, 2012.
- 19 I. Junier, R. K. Dale, C. Hou, F. Kepes, and A. Dean. CTCF-mediated transcriptional regulation through cell type-specific chromosome organization in the α -globin locus. *Nucleic Acids Research*, 40(16):7718–7727, 2012.
- 20 R. Kalhor, H. Tjong, N. Jayathilaka, F. Alber, and L. Chen. Genome architectures revealed by tethered chromosome conformation capture and population-based modeling. *Nature Biotechnology*, 30(1):90–98, 2011.

11:12 Inferring Diploid 3D Chromatin Structures from Hi-C Data

- 21 P H L Krijger, B Di Stefano, E de Wit, F Limone, C Van Oevelen, W De Laat, and T Graf. Cell-of-origin-specific 3D genome structure acquired during somatic cell reprogramming. *Cell Stem Cell*, 18(5):597–610, 2016.
- 22 T. B. K. Le, M. V. Imakaev, L. A. Mirny, and M. T. Laub. High-Resolution mapping of the spatial organization of a bacterial chromosome. *Science*, 342(6159):731–734, 2013.
- 23 A. Lesne, J. Riposo, P. Roger, A. Cournac, and J. Mozziconacci. 3D genome reconstruction from chromosomal contacts. *Nature Methods*, 11(11):1141–1143, 2014.
- 24 D Lin, G Bonora, G G Yardımcı, and W S Noble. Computational methods for analyzing and modeling genome structure and organization. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 11(1):e1435, 2019.
- 25 D. Meluzzi and G. Arya. Recovering ensembles of chromatin conformations from contact probabilities. *Nucleic Acids Res.*, 41(1):63–75, January 2013.
- 26 C W Metz. Chromosome studies on the Diptera. II. The paired association of chromosomes in the Diptera, and its significance. *Journal of Experimental Zoology*, 21(2):213–279, 1916.
- 27 T. Nagano, Y. Lubling, C. Várnai, C. Dudley, W. Leung, Y. Baran, N. M. Cohen, S. Wingett, P. Fraser, and A. Tanay. Cell-cycle dynamics of chromosomal organization at single-cell resolution. *Nature*, 547:61–67, 2017.
- 28 G Nir, I Farabella, C P Estrada, C G Ebeling, B J Beliveau, H M Sasaki, S H Lee, S C Nguyen, R B McCole, S Chatteraj, et al. Walking along chromosomes with super-resolution imaging, contact maps, and integrative modeling. *PLOS genetics*, 14(12):e1007872, 2018.
- 29 J Paulsen, M Sekelja, A R Oldenburg, A Barateau, N Briand, E Delbarre, A Shah, A L Sørensen, C Vigouroux, B Buendia, et al. Chrom3D: three-dimensional genome modeling from Hi-C and nuclear lamin-genome contacts. *Genome biology*, 18(1):21, 2017.
- 30 S. S. P. Rao, M. H. Huntley, N. Durand, C. Neva, E. K. Stamenova, I. D. Bochkov, J. T. Robinson, A. L. Sanborn, I. Machol, A. D. Omer, E. S. Lander, and E. L. Aiden. A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 59(7):1665–1680, 2014.
- 31 M. Rosenthal, D. Bryner, F. Huffer, S. Evans, A. Srivastava, and N. Neretti. Bayesian Estimation of 3D Chromosomal Structure from Single Cell Hi-C Data. *bioRxiv*, page 316265, 2018.
- 32 S Shah, Y Takei, W Zhou, E Lubeck, J Yun, C Linus Eng, N Koulena, C Cronin, C Karp, E J Liaw, et al. Dynamics and Spatial Genomics of the Nascent Transcriptome by Intron seqFISH. *Cell*, 2018.
- 33 L Tan, D Xing, C Chang, H Li, and X S Xie. Three-dimensional genome structures of single diploid human cells. *Science*, 361(6405):924–928, 2018.
- 34 Z Tang, O J Luo, X Li, M Zheng, Jacqueline J Zhu, P Szalaj, P Trzaskoma, A Magalska, J Włodarczyk, B Ruszczycki, et al. CTCF-mediated human 3d genome architecture reveals chromatin topology for transcription. *Cell*, 163(7):1611–1627, 2015.
- 35 H. Tanizawa, O. Iwasaki, A. tanaka, J. R. Capizzi, P. Wickramasignhe, M. Lee, Z. Fu, and K. Noma. Mapping of long-range associations throughout the fission yeast genome reveals global genome organization linked to transcriptional regulation. *Nucleic Acids Research*, 38(22):8164–8177, 2010.
- 36 H. Tjong, K. Gong, L. Chen, and F. Alber. Physical tethering and volume exclusion determine higher-order genome organization in budding yeast. *Genome Res*, 22(7):1295–1305, 2012.
- 37 H Tjong, Wenyuan Li, R Kalhor, C Dai, S Hao, K Gong, Y Zhou, Haochen Li, Xianghong J Z, M A Le Gros, et al. Population-based 3D genome structure analysis reveals driving forces in spatial genome organization. *Proceedings of the National Academy of Sciences*, 113(12):E1663–E1672, 2016.
- 38 N. Varoquaux, F. Ay, W. S. Noble, and J.-P. Vert. A statistical approach for inferring the 3D structure of the genome. *Bioinformatics*, 30(12):i26–i33, 2014.
- 39 S Wang, J Xu, and J Zeng. Inferential modeling of 3D chromatin structure. *Nucleic Acids Research*, 43(8):e54, 2015.

- 40 B. Zhang and P. G. Wolynes. Topology, structures, and energy landscapes of human chromosomes. *Proceedings of the National Academy of Sciences of the United States of America*, 112(19):6062–6067, 2015.
- 41 Z Zhang, G Li, K-C Toh, and W-K Sung. 3D chromosome modeling with semi-definite programming and Hi-C data. *Journal of Computational Biology*, 20(11):831–846, 2013.
- 42 Z. Zhang, G. Li, K.-C. Toh, and W.-K. Sung. Inference of spatial organizations of chromosomes using semi-definite embedding approach and Hi-C data. In *Proceedings of the 17th International Conference on Research in Computational Molecular Biology*, volume 7821 of *Lecture Notes in Computer Science*, pages 317–332, Berlin, Heidelberg, 2013. Springer-Verlag.

A Supplement

■ **Table 1 Constraints improve ambiguous inference.** Each entry is a Bonferroni adjusted p -value for a t -test applied to the specified pair of methods. Values <0.05 are in boldface.

		RMSD per homolog	Distance error per homolog	Distance error, inter-homolog
No constraints	Bead chain connectivity	0.0458	0.0104	0.275
No constraints	Homolog separating	1.3	0.222	4.14
No constraints	Both constraints	0.00309	0.00667	0.0179
No constraints	Both constraints + Null	0.0000404	0.00000773	0.0000883
Bead chain connectivity	Homolog separating	0.00731	0.00588	1.62
Bead chain connectivity	Both constraints	4.89	8.74	0.159
Bead chain connectivity	Both constraints + Null	0.000018	0.00000054	0.000263
Homolog separating	Both constraints	0.0018	0.00713	0.183
Homolog separating	Both constraints + Null	0.0000397	0.152	0.103
Both constraints	Both constraints + Null	0.0000179	0.00000387	0.981

■ **Table 2 Inference with ambiguous and disambiguated data.** Each entry is a Bonferroni adjusted p -value for a t -test applied to the specified pair of methods. Values <0.05 are in boldface.

		RMSD per homolog	Distance error per homolog	Distance error, inter-homolog
Ambiguous	Partially ambiguous	0.000231	0.0000669	0.654
Ambiguous	Unambiguous	3.36E-09	2.88E-08	0.00000369
Partially ambiguous	Unambiguous	0.00000462	0.00000345	0.00000342

Consensus Clusters in Robinson-Foulds Reticulation Networks

Alexey Markin 

Department of Computer Science, Iowa State University, Ames, IA, USA
amarkin@iastate.edu

Oliver Eulenstein

Department of Computer Science, Iowa State University, Ames, IA, USA
<https://www.cs.iastate.edu/people/oliver-eulenstein>
oeulens@iastate.edu

Abstract

Inference of phylogenetic networks – the evolutionary histories of species involving speciation as well as reticulation events – has proved to be an extremely challenging problem even for smaller datasets easily tackled by supertree inference methods. An effective way to boost the scalability of distance-based supertree methods originates from the Pareto (for clusters) property, which is a highly desirable property for phylogenetic consensus methods. In particular, one can employ strict consensus merger algorithms to boost the scalability and accuracy of supertree methods satisfying Pareto; cf. SuperFine. In this work, we establish a Pareto-like property for phylogenetic networks. Then we consider the recently introduced RF-Net method that heuristically solves the so-called *RF-Network problem* and which was demonstrated to be an efficient and effective tool for the inference of hybridization and reassortment networks. As our main result, we provide a constructive proof (entailing an explicit refinement algorithm) that the Pareto property applies to the RF-Network problem when the solution space is restricted to the popular class of tree-child networks. This result implies that strict consensus merger strategies, similar to SuperFine, can be directly applied to boost both accuracy and scalability of RF-Net significantly. Finally, we further investigate the optimum solutions to the RF-Network problem; in particular, we describe structural properties of all optimum (tree-child) RF-networks in relation to strict consensus clusters of the input trees.

2012 ACM Subject Classification Applied computing → Computational biology; Mathematics of computing → Graph theory

Keywords and phrases Phylogenetics, phylogenetic tree, phylogenetic network, reticulation network, Robinson-Foulds, Pareto, RF-Net

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.12

Funding This material is based upon work supported by the National Science Foundation under Grant No. 1617626.

Acknowledgements The authors want to thank the reviewers for their valuable and constructive comments.

1 Introduction

Inferring evolutionary histories of species is a crucial area of study in the biological sciences [10]. The inference of such histories as phylogenetic trees, while still an immensely challenging problem, is gradually becoming tractable on the scale of thousand(s) of species genomes [26, 4].

On the other hand, today, it is well known that evolutionary histories of many species involve complex evolutionary events such as hybridization, reassortment, recombination, and horizontal gene transfer [14]. In this case, evolutionary histories are modeled using *phylogenetic networks* that contain *reticulation vertices* in addition to classical speciation



© Alexey Markin and Oliver Eulenstein;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 12; pp. 12:1–12:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

vertices. Unfortunately, whereas phylogenetic networks potentially represent significantly more powerful tools than phylogenetic trees, the reticulate evolutionary model is of greater complexity than the standard speciation model. In this regard, the current state-of-the-art tools for the estimation of phylogenetic networks cannot yet compete with the inference methods for trees in terms of accuracy and scalability.

Recently, a novel method for the inference of hybridization and reassortment networks has been introduced, called RF-Net [17]. RF-Net was shown to outperform its closest counterparts in terms of scalability and be able to estimate networks with more than 150 taxa credibly. However, there are still significant limitations, particularly in terms of the maximum tractable number of reticulation vertices.

RF-Net follows the *extended hybridization framework* established in [17]. More precisely, the classical *hybridization framework*, formulated in the influential work of Baroni et al. [2], seeks a network that would display each of the input phylogenetic trees exactly. The extended framework then additionally accounts for *errors* that are typically present in input trees for supertree/super-network studies [3, 25]. That is, it defines a cost of embedding an input tree into a candidate network that measures how close that input tree is to be displayed in the network. More formally, the *embedding cost* is defined as the minimum distance between the input tree and each tree displayed in the candidate network (Figure 1 illustrates the concept of displayed trees). Whereas, generally, any established distance measurement for phylogenetic trees can be used to define the embedding cost, RF-Net uses the popular Robinson-Foulds (RF) metric [24]. A related concept was also explored by Yu et al. [29], where the parsimonious ILS principle is combined with the hybridization framework.

Given a collection of input trees, RF-Net attempts to solve the problem of finding a phylogenetic network with at most r reticulation vertices that minimizes the overall embedding cost of the input trees, i.e., the overall sum of the individual embedding costs, as well as the number of reticulations. We refer to this problem as the *RF-network problem*. The RF-network problem is NP-hard [17], and it is similar to the standard supertree (median tree) problem formulations that seek supertrees minimizing the overall distance towards the input trees. Such distance-based supertree methods are widely used for a task of large-scale species tree reconstruction [3]. For example, ASTRAL, a popular supertree software package, seeks a tree minimizing the *quartet distance* towards the input trees [19]. Further, RF-based supertree methods, e.g. [1, 27], as well as gene tree parsimony (GTP) supertree methods, e.g., [9, 12], have been successfully applied by the phylogenetic community (cf. [11, 22]).

One of the most important properties for the distance-based supertree methods is the so-called *Pareto for clusters*¹ property [23]. Pareto for clusters is a highly desired property, both from theoretical as well as application perspectives [6, 20]. This property was introduced in the context of *consensus methods* that seek to “summarize” a collection of input trees over the same species set. A consensus method is Pareto if every cluster that appears in *all* input trees (a *strict consensus cluster*) also appears in the consensus of these trees obtained by that method. This notion was then naturally adopted for distance-based supertree methods when restricted to the consensus setting (i.e., input trees have the same leaf-sets) [15]. Given that there could be multiple optimum solutions to a distance-based supertree problem, a respective distance measurement is said to (i) *satisfy Pareto* if *each* optimum solution contains all the strict consensus clusters from the input trees and (ii) *satisfy weak Pareto* if at least one optimum solution has that property [21]. If a distance measurement satisfies (weak) Pareto then the respective supertree problem, in a consensus setting, can be solved using a

¹ This Pareto property introduced by Neumann [23] in the context of phylogenetic consensus methods is not to be confused with the Pareto efficiency/optimality term originating from the field of economics [18].

parameterized approach, where, first, a strict consensus tree of the input trees is constructed, and then each polytomy in the strict consensus tree is resolved using the original supertree method. In case the input trees are relatively similar, this approach yields a much more efficient and accurate supertree method [21].

Typically, however, the input trees can be unrooted and can have incomplete sets of taxa. In this case, the scalability boost can be achieved using the *strict consensus merger (SCM)* methods [13, 26]. Such strategy was first formulated and evaluated by Swenson et al. [26]; their method, SuperFine, was shown to improve the scalability and accuracy of supertree methods. Generally, any distance-based supertree method with the respective distance measurement satisfying at least weak Pareto can largely benefit from using SCM algorithms. For example, recently, the NP-hard gene duplication supertree problem (GD) [16] was efficiently and effectively approached [20] by combining the greedy SCM method [13] with the exact dynamic programming GD solution [8] and the popular GD heuristic, DupTree [28].

Our contribution. In this work, we establish a Pareto-like property in the context of phylogenetic networks and prove that the Robinson-Foulds embedding cost satisfies it. This result immediately implies that SCM methods can be used to improve the scalability and accuracy of the RF-Net method significantly. We demonstrate this result for the commonly used (restricted) class of networks called *tree-child networks* [7].

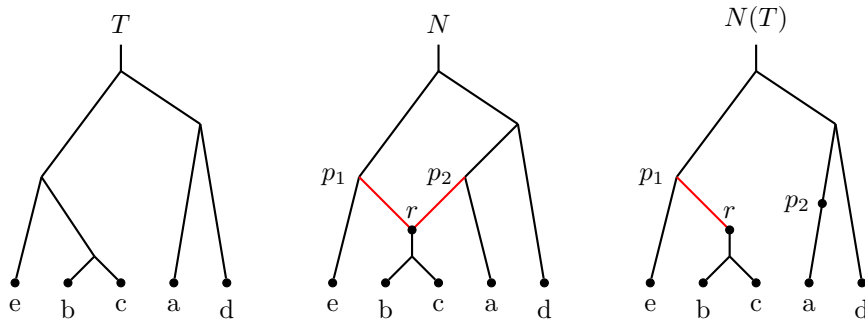
More precisely, we define the notion of C -separated networks for some cluster C . A network is C -separated if one can remove a single edge (u, v) from a network and obtain two disconnected subnetworks/subgraphs, such that the subnetwork rooted at v has the leaf-set C . For example, the network N from Figure 1 is $\{b, c\}$ -separated, but not $\{e, b, c\}$ -separated. Intuitively, this notion implies that the evolutionary history of the species in C (up to their common ancestor) is not inter-related with lineages of any other species in the network.

Founded on this notion, our main result is as follows: when the solution space is constrained to the class of *tree-child networks*, there always exists an *optimum* RF-network that is C -separated for every strict consensus cluster C . Adopting the above-discussed terminology for distance-based supertree methods, this result implies that the RF embedding cost satisfies weak Pareto for tree-child networks. It is important to note that, since our proof is by construction, it explicitly provides an algorithm that can bring a non- C -separated network to the C -separated form with the same or an improved overall embedding cost. To achieve our result, we introduce three edit operations on networks and prove that they preserve specific non-trivial properties. Then we use these operations to design the said algorithm and complete the proof.

Further, we describe conditions for the existence of non- C -separated optimum RF-networks. We prove that such networks can appear only due to *uncertainty in the ordering of some reticulation events* that is not resolved by the input trees. That way, when uncertainty is not present, the RF embedding cost satisfies (strong) Pareto for tree-child networks, rather than weak Pareto. In fact, we prove that even if for some collection of input trees a non- C -separated optimum RF-network N exists, it must be equivalent to another RF-network N' up to a *reordering of reticulation events* (this notion is formally defined in Section 3.2), such that N' is both optimum and C -separated.

2 Preliminaries

A (*phylogenetic*) *network* is a directed acyclic graph (DAG) with a designated root and with all other vertices either of in-degree one and out-degree two (*tree vertices*), in-degree two and out-degree one (*reticulation vertices*), or in-degree one and out-degree zero (*leaves*). All



■ **Figure 1** An example of tree T displayed in network N with one reticulation vertex, r . T is displayed in N by removing the reticulation edge (p_2, r) and obtaining a subdivision $N(T)$.

leaves are bijectively labeled by a label-set X and are identified with the elements of X . For convenience, networks are *planted*, i.e., the root has in-degree zero and out-degree one. For a network N the root and leaves are denoted by $\rho(N)$ and $L(N)$ respectively. For every vertex v we denote the set of children, parent(s), and sibling(s) by $\text{Ch}(v)$, $\text{Pa}(v)$, and $\text{Sb}(v)$ respectively. Note that if v is the child of a reticulation w then, we define $\text{Sb}(v)$ to be the two siblings of w .

We distinguish *reticulation edges* – edges entering a reticulation vertex – and *tree edges* – edges entering a tree vertex. A *tree-path* is a directed path that consists of tree edges. We say that *vertex v has a tree-path to vertex w* if either $v = w$ or there is a tree-path from v to w .

A vertex v is a *descendant* of w if there is a directed path from w to v (we consider each vertex to be a descendant of itself); w is also called an *ancestor* of v . Alternatively, we say that v is *below* w or w is *above* v . A (*hardwired*) *cluster* of vertex v , denoted by C_v , is the set of all leaves that are descendants of v . Generally, we call any set of leaves a *cluster*.

A (*phylogenetic*) *tree* is a network with no reticulation vertices. A *least common ancestor (LCA)* of two vertices v, w in a tree, denoted by $\text{lca}(v, w)$, is the lowest vertex x such that v and w are descendants of x . Two clusters (leaf-sets) C_1 and C_2 are called compatible if there exists a tree that contains both of them. Equivalently, the clusters are compatible if either $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$.

Displayed trees. A tree T is *displayed* in a network N (with the same leaf set), if one can remove exactly one reticulation edge from each reticulation vertex, then remove all potentially appearing non-labeled vertices with out-degree zero, and obtain a subdivision of T – denoted $N(T)$. See an example in Figure 1. We say that an edge of a network is used to display tree T if there is a way to obtain such subdivision of T without removing this edge. Note that in general there could be several ways to display tree T .

Tree-child networks. A network is called *tree-child* if each vertex has at least one outgoing tree edge. It is easy to see that each vertex in a tree-child network must have a tree-path going to some leaf. This is a crucial property that we will employ in this work.

For the convenience of further discussion, we define a surjective mapping from the vertices of a tree-child network N , denoted here V_N , to the vertices of a tree T displayed in N . Let us fix a way to display T in N . It is not difficult to observe that for a tree-child network removing reticulation edges in order to display a tree will not result in out-degree 0 vertices. That is, the fixed subdivision $N(T)$ of T contains all vertices of N . Since $N(T)$ is a subdivision of T , each vertex w from V_N that has two children in $N(T)$ must have a unique equivalent

in T . We then map each such w to the corresponding equivalent vertex in T . Further, we map leaves of N to the leaves of T with the same label. Finally, for each v in V_N that has exactly one child in $N(T)$, let w denote the highest descendant of v in $N(T)$ that was already mapped (i.e., w is either a leaf or has two children); we then map v to the same vertex in T that w is mapped to. For example, in Figure 1 vertices a and p_2 from N are both mapped to leaf a of T .

We then say that a vertex $v \in N$ corresponds to vertex $v' \in T$ if v is mapped to v' .

Embedding cost. We define the cost of embedding a tree T in a network N on the same leaf set using the standard Robinson-Foulds (RF) distance [24]. The cost should be zero, when the tree is displayed in the network and positive otherwise. Then the cost is defined as follows: let \mathcal{P}_N be a set of all trees displayed in N , then

$$\delta(T, N) := \min_{S \in \mathcal{P}_N} RF(T, S),$$

where $RF(T, S)$ is the Robinson-Foulds (cluster) distance defined as the size of the symmetric difference between the cluster representations of two trees.

A tree S , displayed in N , that has the minimum RF distance to T is called *an embedding of T (in N)*. Note that in general T can have multiple embeddings.

3 Consensus clusters in networks

We now establish a definition central to this work.

► **Definition 1** (*C-separated networks*). *Given a cluster C , network N is called C -separated if it contains an edge (u, v) such that removing this edge disconnects the network into two networks (components) N_1 and N_2 with*

■ $L(N_1) = C; \quad L(N_2) = L(N) \setminus C;$

■ *and no edges going across the components in the underlying undirected graph.*

For example, network N from Figure 1 is $\{b, c\}$ -separated.

Let \mathcal{T} be a set of trees over the same leaf set. A cluster C is said to be a *consensus cluster (of \mathcal{T})* if for each $T \in \mathcal{T}$ there is a vertex v such that $C_v = C$.

Given a network N over the leaf set as \mathcal{T} , we define the distance between \mathcal{T} and N as follows:

$$d(\mathcal{T}, N) := w_e \cdot \sum_{T \in \mathcal{T}} \delta(T, N) + w_r \cdot R(N),$$

where $R(N)$ is the number of reticulation vertices in N and $w_e, w_r > 0$ are integer coefficients that weigh the overall embedding cost and the number of reticulations respectively. Note that, for convenience, we define the distance slightly differently from the original definition in [17], where a limit on the number of reticulations was given instead of a weight; however, it does not affect the results stated in this work.

3.1 Consensus clusters in embeddings

We now formulate our core result. To prove it we introduce our main machinery that is also used later to demonstrate stronger results.

► **Theorem 2.** *Let N be a tree-child network with the minimum $d(\mathcal{T}, N)$ distance; then for each consensus cluster C of \mathcal{T} and each $T_i \in \mathcal{T}$, every embedding of T_i in N has cluster C .*

The remainder of the section is dedicated to the proof of the theorem.

Proof strategy. Assume, for the purpose of contradiction, that an embedding of some T_i , $S_i \in \mathcal{P}_N$, does not contain a consensus cluster C . Observe that this implies that N is not C -separated, since each tree displayed in a C -separated network must have cluster C .

We are going to transform N into a C -separated network with a smaller distance to \mathcal{T} . In order to do that, we define three transformations on networks that preserve displayed clusters that are compatible with C .

Formally, the transformations should satisfy the property defined in Definition 3.

► **Definition 3** (C -compatible transformation). *A network transformation operation that can transform N into a network N' is said to be C -compatible if for each tree T displayed in N there exists a tree T' displayed in N' , such that T' contains **all** clusters of T that are compatible with C .*

Let V_C be the set of vertices in N that have a tree-path to some leaf in C . For convenience we classify the *tree vertices* in V_C as follows.

► **Definition 4** (V_C vertices classification). *A tree vertex $v \in V_C$ is exactly of one of the following types:*

- **Type 1:** *if v is a leaf; or if children of v both belong to V_C and both children are tree vertices.*
- **Type 2:** *if both children of v belong to V_C but one of the children is a reticulation vertex.*
- **Type 3:** *If only one child of v , u_1 , belongs to V_C , while the other, u_2 , does not. Note that in this case u_1 has to be a tree vertex, while u_2 could be a tree vertex or a reticulation vertex.*

We now define our first network rearrangement operation and prove that it is C -compatible.

► **Definition 5** (Network transformation T1). *If N contains an edge (w, z) such that $w, z \in V_C$, w, z are tree vertices, w is of Type 2, and z is of Type 3 then transformation T1 proceeds as follows:*

- (i) *Let c denote the sibling of z and u denote the child of z such that $u \notin V_C$;*
- (ii) *Remove edges (w, c) and (z, u) and add edges (w, u) and (z, c) .*

That is, the transformation swaps specific children of z and w . Note that z becomes a Type 2 vertex and w – Type 3; that way T1 “moves” a Type 2 vertex down and a Type 3 vertex up. See the illustration in Figure 2a.

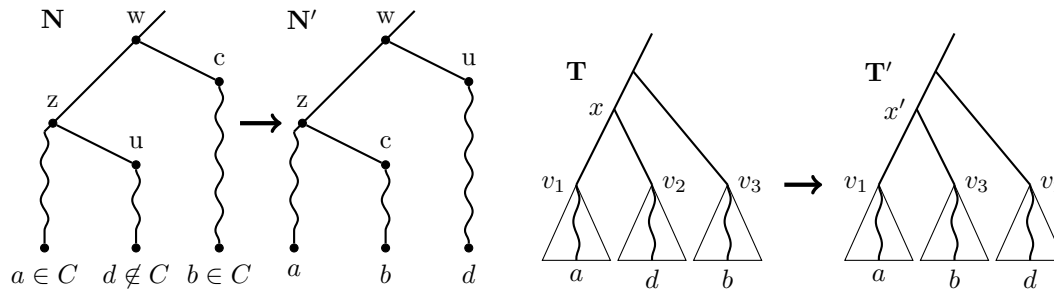
We now prove that this transformation could not increase the embedding distance of an input tree with cluster C to the network.

► **Lemma 6.** *T1 is C -compatible.*

Proof. Assume that T1 was performed as depicted in Figure 2a. Note that, by definition, z and c belong to V_C and therefore have tree-paths to some leaves a and b in C respectively (as shown in the figure as well). On the other hand, $u \notin V_C$ and therefore u has a tree-path to some $d \notin C$ (it could be that $u = d$ and/or $c = b$).

Consider some T displayed in N . We need to show that exists T' displayed in N' that contains all clusters of T compatible with C . First of all, observe that if T can be displayed in N by removing either (or both) of the edges (w, c) and (z, u) then T is also displayed in N' and the statement is trivially true.

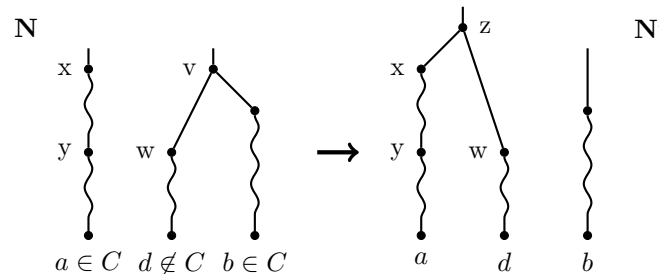
Otherwise, let x denote $\text{lca}_T(a, d)$ (as shown in Figure 2b). Since each edge (w, z) , (z, u) , and (w, c) are used for displaying T (as well as all edges on the tree-paths shown in Figure 2a, since they are tree edges and cannot be removed) then vertex z in N should correspond to x



(a) T1 transformation.

(b) Displayed tree transformation.

■ **Figure 2** (a): A network N transformed into network N' via transformation T1 as per Definition 5; wavy lines indicate tree-paths to leaves. (b): A respective transformation of a tree T displayed in N (assuming that both edges (z, u) and (w, c) are used to display T) to a tree T' displayed in N' .



■ **Figure 3** An illustration of transformation T2. Network N is transformed into N' . Potentially empty tree-paths are shown via wavy lines.

in T . Further, vertex c should correspond to the sibling of x , v_3 . Therefore, v_3 should have leaf b below it. It is then not difficult to see that a tree T' obtained by swapping subtrees rooted at v_3 and v_2 is displayed in N' (see Figure 2b).

Finally, observe that T differs from T' only by one cluster; namely, C_x , which is not present in T' . However, C_x is not compatible with C , since (i) C_x contains $a \in C$ and $d \notin C$ and (ii) $C \not\subseteq C_x$ because $b \notin C_x$. ◀

We now define two more C -compatible rearrangement operations.

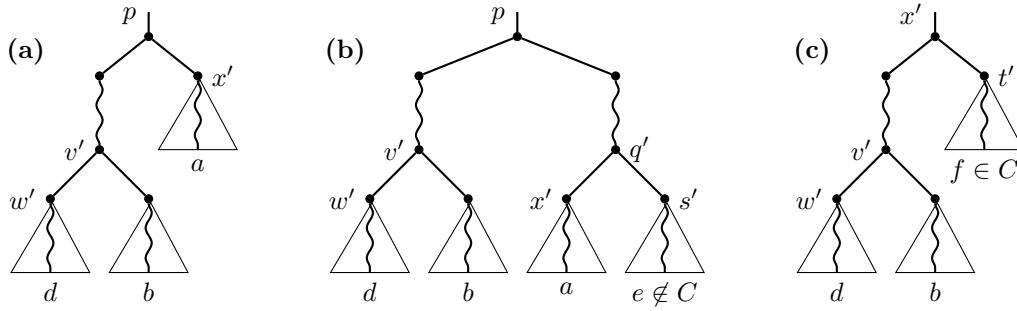
► **Definition 7** (Network transformations T2 and T3). *Let x be a tree vertex in V_C of Type 2 or Type 1 such that there is no tree vertex above x of Type 2 or 1 with a path to x . Further, let y be the highest vertex of Type 1 with a tree-path from x (note that y could be equal to x); we then require that there is no vertex of Type 3 on the path from x to y . The transformations are defined as follows:*

T2. *Given edge (v, w) , such that $v \in V_C$, $w \notin V_C$, and v is not an ancestor of x :*

- (i) *remove edge (v, w) and contract v ;*
- (ii) *subdivide $(Pa(x), x)$ with a new vertex z and add edge (z, w) .*

T3. *Same as T2 with the difference that $v \notin V_C$, while $w \in V_C$, and w is not an ancestor of x (note that v could be an ancestor of x).*

► **Remark 8.** Note that T1, T2, and T3 could be seen as specifically constrained versions of the subnet prune and regraft operation (SNPR) defined and studied by Bordewich et al. [5].



■ **Figure 4** Three possible scenarios – (a), (b), and (c) – for a tree T displayed in N . Tree T' displayed in a transformed network N' (as shown in Figure 3) can be obtained in all three cases by regrafting the subtree rooted at w' just above vertex x' .

► **Lemma 9.** T_2 and T_3 are C -compatible.

Proof. The proofs for T_2 and T_3 are similar; therefore, we provide a proof for T_2 only. Let N be transformed to N' as depicted in Figure 3.

Consider some T displayed in N . We need to show that exists T' displayed in N' that contains all clusters of T compatible with C . Similarly to the proof of Lemma 6, observe that if (v, w) is a reticulation edge and there exists a way to display T in N without using this edge, then T is displayed in N' as well and the statement holds. We now fix a way to display T in N . Let x' then denote the vertex in T such that x (from N) corresponds to x' . Further, let $v' = \text{lca}_T(b, d)$ be the vertex such that v corresponds to v' and let w' be the child of v' such that w corresponds to w' . Observe that a tree T' displayed in N' is obtained from T by (i) removing edge (v', w') and suppressing vertex v' and (ii) subdividing edge $(\text{Pa}(x'), x')$ with a new vertex z' and adding an edge (z', v') . That is, subtree rooted at w' is regrafted above x' .

We now distinguish three cases for our proof depending on the location of x' relative to v' in T . Let p denote $\text{lca}_T(v', x')$. Then the three cases are as follows (see also the illustration of these cases in Figure 4):

- (a) x' is a child of p (see Figure 4(a)). Then T' does not have only those clusters of T that correspond to the intermediate vertices on the path from p to w' . However, each such cluster could not be compatible with C , since it contains $d \notin C$ and $b \in C$, but does not contain $a \in C$ (that is, it is neither a subset nor a superset of C , nor it is disjoint from C).
- (b) There is at least one vertex on the path from p to x' (see Figure 4(b)). Let q' and s' denote the parent and sibling of x' respectively (as depicted on the figure). We argue that there must exist leaf $e \notin C$ below s' .

Consider a fixed subdivision $N(T)$ and let t be the lowest ancestor of x in $N(T)$ with two children (that is, t corresponds to t'). It is not difficult to observe that there could be at most one reticulation edge on the path from t to x in $N(T)$ and that edge has to originate from t (due to the tree-child property). Recall now that there is no vertex of Type 1 or 2 above x .

If there is a reticulation edge on the path from t to x , let s denote the child of t that is a tree vertex. Note that s could not belong to V_C , since in that case t would be a vertex of Type 2 (note that the other child of t , that is a reticulation, must have a tree-path to x and therefore a tree-path to $a \in C$ – that is, it belongs to V_C). Hence, there exists $e \notin C$ with a tree-path from s . Given that s corresponds to s' in T , leaf e must be present below s' as well.

Further, if there is no reticulation edge on the path from t to x , then s , defined as the child of t that is not on this path, is not in V_C as well. This is the case, since otherwise t would be a vertex of Type 1 or 2. Therefore, leaf e exists in that case as well.

We use these observations to finish the proof for (b). Note that tree T' does not have the clusters of T that correspond to the intermediate vertices on the paths from p to w' and from p to x' . However, using the argument similar to (a), it is not difficult to see that none of these clusters is compatible with C .

- (c) $p = x'$ (see Figure 4(c)). Let t' be the child of x' that is not an ancestor of v' . To prove that the lemma holds in that case as well, we are going to show that there must exist leaf $f \in C$ below t' . Note that f could be equal to a .

Consider a fixed subdivision $N(T)$ and let u be the vertex in $N(T)$ that corresponds to x' and has two children. Observe that by our constraints u could be any vertex on the path from x to y (including x and y). Therefore, by the constraints in the definition of T2, u must be of Type 2 or 1 and hence both its children must be in V_C . Clearly, one of the children of u corresponds to t' and therefore such leaf f must exist.

Similarly to (a), tree T' does not contain only those clusters of T that correspond to the intermediate vertices on the path from x' to w' . It is then not difficult to see that these clusters are not compatible with C (via the same argument as in (a)).

These cases cover all possible scenarios, since by constraints given in Definition 7 v' cannot be an ancestor of x' . Note that for transformation T3 such a case (i.e., v' is an ancestor of x') is possible; however, using the same type of arguments as above, it is not difficult to check that for this case C -compatibility is preserved as well. ◀

► **Lemma 10.** *A not C -separated network N can be transformed into a C -separated network N' only using operations T1, T2, and T3.*

Proof. Consider the set of connected components (in the undirected sense) induced by the vertices in V_C . Let $A \in \mathcal{C}$ be such component; that is, all vertices in A are in V_C and every edge incident to A connects to a vertex that is not in V_C . We are going to potentially exclude some “redundant” vertices from A , while keeping it connected. Let v be a *maximal* vertex in A , i.e., its parent(s) is(are) not in A . If v has exactly one child that is in A , then exclude v from set A . Keep iterating this procedure until all maximal vertices in A are either leaves or have both children in A .

Now, let \mathcal{C}' be the set of components obtained from \mathcal{C} by performing the described above pruning procedure on each component. Note that after the pruning all maximal vertices in \mathcal{C}' components are tree-vertices of Type 1 or 2.

Next, we are going to use transformations T1, T2, and T3 on N in order to aggregate all components in \mathcal{C}' together and separate them from the rest of the network.

For each $A \in \mathcal{C}'$ let $I(A)$ be the set of vertices in A that are incident to an edge outside of A . Note that $I(A)$ includes all maximal vertices in A . Further, define $I(\mathcal{C}') := \cup_{A \in \mathcal{C}'} I(A)$. Since N is a DAG, there a vertex x in $I(\mathcal{C}')$, such that no other vertex in this set is an ancestor of x . Clearly, x must be a maximal vertex of some component in \mathcal{C}' .

Assume that x is a vertex of Type 2 (if x is of Type 1, then this step is not needed). Let y be the highest vertex of Type 1 with a tree path from x . Consider now all vertices on the tree path from x to y (excluding y); let us denote these vertices as $(v_1 = x, v_2, \dots, v_k)$. Note that each v_i is either of Type 2 or 3. Transformation T1 then allows us to rearrange the types of these vertices in the way that there will be an index $1 \leq j \leq k$ such that for all $i \geq j$, v_i is a Type 2 vertex and for all $i < j$, v_i is a Type 3 vertex. That way we modified the component $C_x \in \mathcal{C}'$ that contains x . Since v_1, \dots, v_{j-1} (if $j > 1$) vertices have only one

child in V_C by the definition of Type 3 vertices, then applying the pruning procedure again will exclude v_1, \dots, v_{j-1} from C_x . Therefore, we re-define $x := v_j$. Observe that x is still a vertex with the property that no other vertex in $I(C')$ is an ancestor of x . Further, there is no vertex of Type 3 between x and y .

Consider now all edges of type (v, w) , such that v is in some component $A \in \mathcal{C}'$ and $w \notin V_C$. We perform transformation T2 (in any order) by reconnecting each such w above x . Further, consider all maximal vertices in \mathcal{C}' except for x , we use T3 to reconnect those vertices above x (note that after each such operation T3 we need to re-assign $x = \text{Pa}(x)$). It is not difficult to see that as the result of this procedure all vertices in V_C now lie in the same connected component. Finally, consider all reticulation vertices v in that component, such that one of the edges (w, v) is outside of the component. We perform T3 to relocate the source of such reticulation edges above x . ◀

The proof of Theorem 2 then follows from the above results. That is, Lemma 10 allows us to obtain a C -separated network N' with the property that for each S_j displayed in N exists S'_j displayed in N' , such that S'_j has all the clusters of S_j compatible with C (guaranteed by Lemmas 6 and 9). Note now that all clusters in input trees T_j are compatible with C ; hence, $RF(T_j, S'_j) \leq RF(T_j, S_j)$. Moreover, we assumed in the beginning that S_i does not contain C , while the respective tree S'_i displayed in N' must contain it. Therefore:

$$d(\mathcal{T}, N') < d(\mathcal{T}, N).$$

3.2 Consensus clusters in RF-networks

Theorem 2 says that for a tree-child network N with minimum $d(\mathcal{T}, N)$ distance (for convenience, we refer to such networks as *minimum RF-networks*), each embedding of the input trees must contain *all* the consensus clusters. While this does not directly imply that each minimum RF-network is C -separated for each consensus cluster, it is not difficult to observe that at least one minimum RF-network has this property.

► **Theorem 11.** *For a collection of trees \mathcal{T} there exists a minimum RF-network N such that N is C -separated for each consensus cluster C of \mathcal{T} .*

Proof. Take any minimum RF-network N . If N is not C -separated for some consensus cluster C , the transformations T1-T3 defined in the previous section can be used to bring N to the C -separated form (using the procedure from the proof of Lemma 10). As was demonstrated, T1-T3 do not increase the embedding distance for any $T \in \mathcal{T}$; thus the resulting network N' is also a minimum RF-network.

Further, it is not difficult to observe that if N is C_1 -separated and the procedure from Lemma 10 is used to make it C_2 -separated – where C_1 and C_2 are two distinct compatible clusters – the resulting network will remain C_1 -separated. Hence, the theorem holds. ◀

While this theorem has important implications for practitioners on its own, we now show a much stronger result claiming that all minimum RF-networks are “almost” C -separated for each consensus cluster C . More precisely, we show that a minimum RF-network is not C -separated only if there is uncertainty induced by the set of input trees.

► **Definition 12.** *We say that two networks N_1 and N_2 are equivalent up to an ordering of reticulation events, if there is a tree-path (u_1, \dots, u_k) in N_1 such that each u_i has one reticulation child and N_2 can be obtained from N_1 by re-ordering the vertices on that path along with their outgoing reticulation edges. That is, N_2 can be obtained from N_1 by transformations similar to T1.*

► **Theorem 13.** *Let N be a tree-child network that displays a set of trees \mathcal{Q} with the minimal number of reticulations (i.e., if at least one reticulation edge is removed from N , it will not display at least one of the trees from \mathcal{Q}). Then for a consensus cluster C of \mathcal{Q} either N is C -separated or N is equivalent to a C -separated network N' up to an ordering of reticulation events, such that N' displays all trees from \mathcal{Q} as well.*

The proof is omitted for brevity.

Using Theorem 2 we obtain the following corollary.

► **Corollary 14.** *Given an input tree-set \mathcal{T} , consider a minimum RF-network N for \mathcal{T} that is not C -separated for some consensus cluster C . Let \mathcal{Q} be the set of all embeddings for all trees $T \in \mathcal{T}$. Then there exists a C -separated minimum RF-network N' that displays each tree in \mathcal{Q} and N' is equivalent to N up to an ordering of reticulation events.*

This corollary implies that a minimum RF-network is not C -separated *only if there is uncertainty in the ordering of some reticulation events that is not resolved by the embeddings of the input trees.*

4 Conclusion


We make the first step towards boosting the scalability of methods for the inference of hybridization and reassortment networks. We establish a Pareto-like property for phylogenetic networks and prove that the recently introduced *RF embedding cost* satisfies it for tree-child networks. This result allows one to use strict consensus merger strategies to significantly magnify the scalability of the RF-Net method, particularly, when input trees are relatively similar. Our results arise from studying the structure of optimum tree-child RF-Networks in relation to the strict consensus clusters from the input trees. We anticipate that future work would shed light on whether the Pareto properties that we discovered in this work could be generalized to other classes of networks, e.g., reticulation-visible networks.

References

- 1 Mukul S. Bansal, J. Gordon Burleigh, Oliver Eulenstein, and David Fernández-Baca. Robinson-Foulds supertrees. *Algorithms Mol Biol*, 5:18, 2010.
- 2 Mihaela Baroni, Charles Semple, and Mike Steel. A framework for representing reticulate evolution. *Annals of Combinatorics*, 8(4):391–408, 2005.
- 3 Olaf R.P. Bininda-Emonds, editor. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 4 of *Computational Biology*. Springer Verlag, 2004.
- 4 Olaf R.P. Bininda-Emonds, John L. Gittleman, and Mike A. Steel. The (Super)tree of life: Procedures, problems, and prospects. *The (Super)tree of life: Procedures, problems, and prospects*, 33:265–289, 2002.
- 5 Magnus Bordewich, Simone Linz, and Charles Semple. Lost in space? Generalising subtree prune and regraft to spaces of phylogenetic networks. *J Theor Biol*, 423:1–12, 2017.
- 6 David Bryant. A classification of consensus methods for phylogenetics. In M. Janowitz, F.-J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts, editors, *Discrete Mathematics and Theoretical Computer Science*, volume 61, pages 163–185. Am Math Soc, Providence, RI, 2003.
- 7 Gabriel Cardona, Francesc Rossello, and Gabriel Valiente. Comparison of tree-child phylogenetic networks. *IEEE/ACM TCBB*, 6(4):552–569, 2009.
- 8 Wen-Chieh Chang, Paweł Górecki, and Oliver Eulenstein. Exact solutions for species tree inference from discordant gene trees. *J Bioinform Comput Biol*, 11(5):1342005, 2013.

- 9 Cedric Chauve, Nadia El-Mabrouk, Laurent Guéguen, Magali Semeria, and Eric Tannier. *Duplication, Rearrangement and Reconciliation: A Follow-Up 13 Years Later*, pages 47–62. Springer, London, 2013.
- 10 Theodosius Dobzhansky. Nothing in biology makes sense except in the light of evolution. *The american biology teacher*, 75(2):87–91, 2013.
- 11 Juliane C Dohm, André E Minoche, Daniela Holtgräwe, Salvador Capella-Gutiérrez, Falk Zakraewski, Hakim Tafer, Oliver Rupp, Thomas Rosleff Sørensen, Ralf Stracke, Richard Reinhardt, et al. The genome of the recently domesticated crop plant sugar beet (*Beta vulgaris*). *Nature*, 505(7484):546, 2014.
- 12 Oliver Eulenstein, Snehalata Huzurbazar, and David A Liberles. *Evolution after Gene Duplication*, chapter Reconciling Phylogenetic Trees, pages 185–206. John Wiley, 2010.
- 13 Daniel H Huson, Scott M Nettles, and Tandy J Warnow. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *J Comput Biol*, 6(3-4):369–386, 1999.
- 14 Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.
- 15 Harris T. Lin, J. Gordon Burleigh, and Oliver Eulenstein. Consensus properties for the deep coalescence problem and their application for scalable tree search. *BMC Bioinf*, 13 Suppl 10:S12, 2012.
- 16 Bin Ma, Ming Li, and Louxin Zhang. From Gene Trees to Species Trees. *SIAM J. Comput.*, 30(3):729–752, 2000. doi:10.1137/S0097539798343362.
- 17 Alexey Markin, Tavis K Anderson, Venkata SKT Vadali, and Oliver Eulenstein. Robinson-Foulds Reticulation Networks. *bioRxiv*, 2019. doi:10.1101/642793.
- 18 Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.
- 19 Siavash Mirarab, Rezwana Reaz, Md S Bayzid, Théo Zimmermann, M Shel Swenson, and Tandy Warnow. ASTRAL: genome-scale coalescent-based species tree estimation. *Bioinformatics*, 30(17):541–548, 2014.
- 20 Jucheol Moon and Oliver Eulenstein. Synthesizing large-scale species trees using the strict consensus approach. *J Bioinform Comput Biol*, 15(03), 2017.
- 21 Jucheol Moon, Harris T Lin, and Oliver Eulenstein. Consensus properties and their large-scale applications for the gene duplication problem. *J Bioinform Comput Biol*, 14(03), 2016.
- 22 Thomas J Near, Ron I Eytan, Alex Dornburg, Kristen L Kuhn, Jon A Moore, Matthew P Davis, Peter C Wainwright, Matt Friedman, and W Leo Smith. Resolution of ray-finned fish phylogeny and timing of diversification. *PNAS*, 109(34):13698–13703, 2012.
- 23 DA Neumann. Faithful consensus methods for n-trees. *Math Biosci*, 63(2):271–287, 1983.
- 24 David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Math Biosci*, 53:131–147, 1981.
- 25 Mike Steel and Allen Rodrigo. Maximum likelihood supertrees. *Syst Biol*, 57(2):243–250, April 2008.
- 26 M Shel Swenson, Rahul Suri, C Randal Linder, and Tandy Warnow. SuperFine: fast and accurate supertree estimation. *Syst Biol*, 61(2):214, 2011.
- 27 Pranjal Vachaspati and Tandy Warnow. FastRFS: fast and accurate Robinson-Foulds Supertrees using constrained exact optimization. *Bioinformatics*, 33(5):631–639, 2016.
- 28 André Wehe, Mukul S Bansal, J Gordon Burleigh, and Oliver Eulenstein. DupTree: a program for large-scale phylogenetic analyses using gene tree parsimony. *Bioinformatics*, 24(13):1540–1541, 2008.
- 29 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.

Weighted Minimum-Length Rearrangement Scenarios

Pijus Simonaitis¹ 

LIRMM – Université Montpellier, France
pijus.simonaitis@lirmm.fr

Annie Chateau

LIRMM – Université Montpellier, France

Krister M. Swenson 

LIRMM, CNRS – Université Montpellier, France
swenson@lirmm.fr

Abstract

We present the first known model of genome rearrangement with an arbitrary real-valued weight function on the rearrangements. It is based on the dominant model for the mathematical and algorithmic study of genome rearrangement, Double Cut and Join (DCJ). Our objective function is the sum or product of the weights of the DCJs in an evolutionary scenario, and the function can be minimized or maximized. If the likelihood of observing an independent DCJ was estimated based on biological conditions, for example, then this objective function could be the likelihood of observing the independent DCJs together in a scenario. We present an $O(n^4)$ -time dynamic programming algorithm solving the MINIMUM COST PARSIMONIOUS SCENARIO (MCPS) problem for co-tailed genomes with n genes (or syntenic blocks). Combining this with our previous work on MCPS yields a polynomial-time algorithm for general genomes. The key theoretical contribution is a novel link between the parsimonious DCJ (or 2-break) scenarios and quadrangulations of a regular polygon. To demonstrate that our algorithm is fast enough to treat biological data, we run it on syntenic blocks constructed for Human paired with Chimpanzee, Gibbon, Mouse, and Chicken. We argue that the Human and Gibbon pair is a particularly interesting model for the study of weighted genome rearrangements.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Weighted genome rearrangement, Double cut and join (DCJ), Edge switch, Minimum-weight quadrangulation

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.13

Supplement Material Code available at bitbucket.org/pijus_simonaitis/mcps_wabi2019/.

Funding This work is partially supported by the Labex NUMEV flagship project GEM, and by the CNRS project Osez l'Interdisciplinarité.

1 Introduction

Since its introduction in 2005, Double Cut and Join (DCJ) has become the foundational model for mathematical and algorithmic study of genome rearrangement. Its power to simulate a diverse combination of chromosomal events along with its relatively simple mathematical structure has facilitated a diverse expansion and use of DCJ. Grounded on primary work that outlines how to compute DCJ scenarios between pairs of genomes [42, 9], many active research topics have blossomed including the sampling of scenarios [30], whole

¹ Corresponding author



genome duplication [41, 4], orthology assignment and evolution under gene duplication and indels [33, 44], family-free genomic distances [29], cancer genomics [43], ancestral genome reconstruction [3, 2], and phylogenetic inference [28, 3].

Yet to further the utility of DCJ, there is a potential for the addition of biological constraints to the model. This article, when combined with our previous work [35], represents the first known algorithm – for any known model of genome rearrangement – where an arbitrary real-valued weight function on rearrangements can be given as part of the input. The objective function can be the sum or product of the weights of the DCJs in an evolutionary scenario, and the function can be minimized or maximized. Say, for example, that one was to weight a DCJ based on the likelihood of observing it given particular biological conditions. Then our objective function could compute the maximum likelihood scenario over all minimum-length scenarios.

Our dynamic programming algorithm is possible thanks to the existence of equivalence classes on weighted parsimonious scenarios.

1.1 Background

Most research on weighted genome rearrangement has only considered constraints on the gene sequences, or the types of rearrangements. Length-weighted reversals and centered reversals are such examples [7]. Other examples weight the types of rearrangements relative to each other, possible types being inversion, transposition, inverse transposition, insertion/deletion (indel), and tandem duplication random loss [13, 17, 25]. The preservation of common intervals or groups of co-localized genes can also be enforced. Hartman, Middendorf, and Bernt present a summary of this work in [24]. The same authors have combined these approaches by studying type-weighted rearrangement scenarios preserving common intervals [23].

Over the last few years research has emerged that incorporates external biological information for weighting genome rearrangements. This includes methods that consider the sizes of intergenic regions [12, 20, 14], or that minimize the number of DCJs cutting regions that are distant in physical space [39, 36]. In [32], we computed scenarios that maximize the Hi-C contacts between breakpoints of rearrangements using a greedy algorithm for weighted pairs of gene adjacencies. It was left an open question as to whether an exact polynomial-time algorithm to do this exists. This article answers a similar question, by describing an exact polynomial-time algorithm for weighted pairs of gene extremities (rather than adjacencies).

We base our article on the model called *Double Cut and Join* (DCJ) [42, 9]. Each chromosome is represented by an ordering of directed genes (or syntenic blocks), and each gene is represented by two extremities. Extremities that are adjacent in this ordering are paired, and transformations of these pairs occur by swapping extremities of two pairs. DCJ can naturally be interpreted as a graph edit model with the use of the *breakpoint graph*, where there is an edge between gene extremities a and b for each adjacent pair. A DCJ operation replaces an edge pair $\{\{a, b\}, \{c, d\}\}$ of the graph by $\{\{a, c\}, \{b, d\}\}$ or $\{\{a, d\}, \{b, c\}\}$.

This edge edit operation on a graph is called a *2-break*. 2-breaks – also known as edge swaps, switches, rewirings, or flips [21] – have been studied in the restricted setting of genome rearrangement [5, 9] and sorting strings by mathematical transpositions [1, 18], but also in more general settings of generating random networks [21] and network design [11, 19].

1.2 The Problem

The foundational problem is that of finding a minimum length (or *parsimonious*) 2-break scenario transforming an arbitrary multi-graph into another one having the same degree sequence [11, 9, 30]. For an arbitrary real valued cost function ω defined on the set of

2-breaks, we treat the problem of finding a parsimonious 2-break scenario minimizing the sum of the ω -costs of its 2-breaks, the ω -MINIMUM COST PARSIMONIOUS SCENARIO problem (see Section 5 for a more precise definition of our cost function).

In this article we present an $O(n^4)$ -time algorithm solving ω -MCPS for the perfect matchings on n vertices. These graphs represent *co-tailed* pairs of genomes, which are the pairs of genomes that share the same telomeric adjacencies [31].

Consider the ω -MCPS problem for general genomes.

► **Problem 1** (ω -MINIMUM COST PARSIMONIOUS SCENARIO or ω -MCPS).

INPUT: A pair of genomes (A, B) , and a real valued cost function ω defined for the set of DCJ rearrangements on the gene extremities of A and B .

OUTPUT: A minimum length DCJ scenario transforming A into B that minimizes the sum of the ω -costs of its operations.

Our previous work guarantees that our solution for ω -MCPS on co-tailed genomes can be extended to ω -MCPS on general genomes, with an overhead that is linear in the number of linear chromosomes [35]. Note that ω -MCPS asks for a parsimonious scenario minimizing the sum of costs. Our algorithm from Section 5 can be easily modified to find a parsimonious scenario minimizing the product of costs, or maximizing the sum or the product of costs. For the sake of simplicity we only treat the case of minimizing the sum of costs.

Experiments presented in Section 6 show that our algorithm is efficient enough for species as distant as Human and Chicken. We construct breakpoint graphs between Human and four other species including Chimpanzee, Gibbon, Mouse, and Chicken, and demonstrate that our algorithm can be used in practice despite its elevated time complexity.

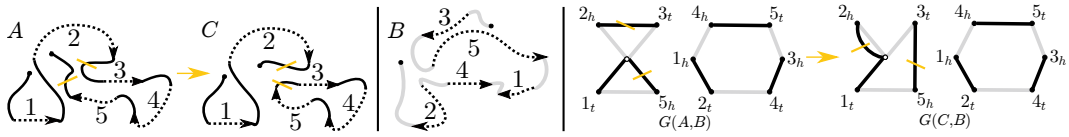
Designing an informative cost function remains a challenging open problem. In Section 7 we provide a short overview of the recent results linking evolution by genome rearrangement to various chromatin features. We also propose the use of *sure* 2-breaks as a testing ground for potential cost functions. These are the rearrangements that are present in every parsimonious scenario transforming one genome into another. We observe that such rearrangements comprise at least 7% of the parsimonious scenarios for the studied species.

2 Genomes, Breakpoint Graphs and DCJ Scenarios

A *genome* consists of *chromosomes* that are linear or circular orders of genes separated by potential *breakpoint* regions. In Figure 1 the tail of an arrow represents the *tail extremity*, and the head of an arrow represents the *head extremity* of a gene. A set of *adjacencies* between the gene extremities can uniquely represent a genome. An *adjacency* is either *internal*: an unordered pair of the extremities that are adjacent on a chromosome, or *external*: a single extremity adjacent to one of the two ends of a linear chromosome. We will suppose that two genomes are partitioned into n genes or syntenic blocks each occurring exactly once in each genome. We use the *breakpoint graph* to represent a pair of genomes, and our goal is to transform one genome into another using a sequence of *DCJ* operations.

► **Definition 1** (co-tailed genomes). *Two genomes are co-tailed if their sets of external adjacencies are equal.*

► **Definition 2** (breakpoint graph [5]). *$G(A, B)$ for genomes A and B is an 2-edge-colored Eulerian undirected multi-graph. Its vertices are the $2n$ gene extremities and an additional vertex \circ . For every internal adjacency $\{a, b\} \in A$ (respectively $\{a, b\} \in B$) there is a*



■ **Figure 1** Three genomes A , B and C are depicted together with the breakpoint graphs $G(A, B)$ and $G(C, B)$. A and B consists of a single linear chromosome, while C consists of a linear and a circular chromosome. The three genomes contain the five genes depicted as dashed arrows. The adjacencies of A are $\{\{1_t\}, \{1_h, 2_t\}, \{2_h, 3_t\}, \{3_h, 4_t\}, \{4_h, 5_t\}, \{5_h\}\}$. A DCJ $\{2_h, 3_t\}, \{5_h\} \rightarrow \{5_h, 3_t\}, \{2_h\}$ transforms A into C . The corresponding graph transformation $G(A, B) \rightarrow G(C, B)$ is a 2-break transforming one simple alternating cycle into two.

black (respectively gray) edge $\{a, b\}$ in $G(A, B)$ and for every external adjacency $\{a\} \in A$ (respectively $\{a\} \in B$) there is a black (respectively gray) edge $\{a, \circ\}$ in $G(A, B)$. There is also a number of black and gray loops $\{\circ, \circ\}$ ensuring that the black and gray degrees of \circ are equal.

► **Definition 3** (double cut and join [42]). A DCJ cuts one or two adjacencies of a genome and joins the resulting ends of the chromosomes back in one of the four following ways: $\{a, b\}, \{c, d\} \rightarrow \{a, c\}, \{b, d\}$; $\{a, b\}, \{c\} \rightarrow \{a, c\}, \{b\}$; $\{a, b\} \rightarrow \{a\}, \{b\}$; and $\{a\}, \{b\} \rightarrow \{a, b\}$.

3 Parsimonious 2-break Scenarios for 2-edge-colored Graphs

The problem of transforming a genome A into B using a sequence of DCJ operations can be reduced to the problem of transforming the breakpoint graph $G(A, B)$ into $G(B, B)$ using 2-breaks on its black edges (see Figure 1) [35].

► **Definition 4** (2-break). A 2-break $\{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$ is a graph transformation replacing edges $\{u, v\}$ and $\{q, s\}$, with edges $\{u, q\}$ and $\{v, s\}$. The vertices and the edges of this 2-break are respectively $\{u, v, q, s\}$ and $\{\{u, v\}, \{q, s\}, \{u, q\}, \{v, s\}\}$.

The problem of transforming a breakpoint graph using 2-breaks can be formulated in a more general setting of transforming an arbitrary multi-graph H_1 into another H_2 . A 2-break preserves the degrees of the vertices, thus H_1 can only be transformed into a graph H_2 having the same degree sequence as H_1 . Given H_1 and H_2 with equal degree sequences, one could combine them into an 2-edge-colored multi-graph using a bijection between the equal degree vertices, coloring the edges of H_1 in black and the edges of H_2 in gray. Such an Eulerian 2-edge-colored graph admits a decomposition into edge-disjoint alternating cycles. An alternating cycle is *simple*, if it cannot be further decomposed into edge-disjoint alternating cycles. A simple alternating cycle is a *circle*, if the black and gray degrees of its vertices are equal to 1. The *size* of a simple cycle is its number of vertices. The breakpoint graph $G(A, B)$ in Figure 1 has two simple cycles, but only one of them is a circle.

► **Definition 5** (2-break scenario and terminal graph). A 2-break scenario for an 2-edge-colored multi-graph G is a sequence of 2-breaks transforming the black edges of G into its gray edges. We call an 2-edge-colored graph with equal multisets of black and gray edges a terminal graph.

See Figure 2 (a) for an example of a 2-break scenario and a terminal graph. The problem of finding a minimum length (or *parsimonious*) 2-break scenario is closely related to the problem of finding a MAXIMUM ALTERNATING EDGE-DISJOINT CYCLE DECOMPOSITION

(MAECD) of a graph [9, 11, 30, 15]. The key observation is that a 2-break in a parsimonious scenario increases the size of a MAECD of a graph by 1. Theorem 6 leads to a couple of easy corollaries that we will use in our proofs.

► **Theorem 6** (Bienstock and Günlük in [11]). *The minimum length of a 2-break scenario for an Eulerian 2-edge-colored multi-graph is equal to the number of its vertices divided by two minus the size of its MAECD.*

► **Corollary 7.** *The first 2-break τ in a parsimonious scenario ρ for a circle transforms it into a union of two vertex-disjoint circles. The rest of ρ does not contain τ .*

► **Corollary 8.** *A parsimonious scenario for a graph having two connected components can be partitioned into two sub-sequences that are parsimonious scenarios for these components. If parsimonious scenarios ρ_1 and ρ_2 for these components are given, then their shuffle, a sequence that can be partitioned into two sub-sequences equal to ρ_1 and ρ_2 , is a parsimonious scenario for the graph.*

4 Equivalence Classes of the Parsimonious Scenarios for a Circle

In Section 5 we concern ourselves with the cost of a scenario, that is, the sum of the costs of its constituent 2-breaks. This way two scenarios consisting of the same 2-breaks, but possibly in a different order, have the same cost. This motivates the following definition.

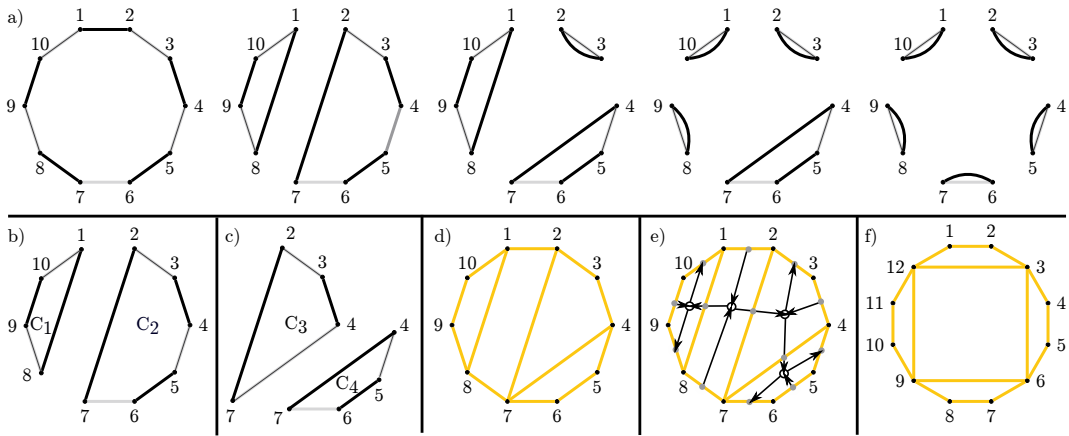
► **Definition 9** (equivalent scenarios). *2-break scenarios are equivalent if they consist of the same 2-breaks.*

A *Scenario graph* of a parsimonious scenario ρ for a circle is introduced in Section 4.1, as a graph whose edges are exactly the edges of the 2-breaks in ρ . Two parsimonious scenarios for a circle are then proved to be equivalent if and only if their scenario graphs are equal in Section 4.4. In Section 4.1 we show that the scenario graph of a parsimonious scenario is planar and provide an embedding of this graph that is a partition of a regular even polygon into quadrilaterals, also known as a “quadrangulation” or a “complete quadrillage”. In Section 4.4 we show that for any *quadrangulation of a circle* there exists an equal scenario graph, thus establishing a bijection between the quadrangulations and the equivalence classes of the parsimonious scenarios for a circle. A bijection between the 2-breaks in a scenario and the faces of its scenario graph is established in Section 4.2. This link enables us to partition a parsimonious scenario for a circle into the subsequences that are parsimonious scenarios for its *sub-circles* in Section 4.3, and ultimately allows us to efficiently search the space of all the parsimonious scenarios in Section 5.

► **Definition 10** (sub-circle). *Take an odd length path in a circle. If the edges incident to its endpoints are black (respectively gray), then add a gray (respectively black) edge incident to the endpoints of this path to obtain a circle. We say that the added edge delimits the sub-circle. See Figure 2 (b) for an example of sub-circles.*

4.1 A Scenario Graph is a Quadrangulation of a Circle

We show that a scenario graph for a parsimonious scenario of a circle is planar, and provide an embedding of this graph that is a partition of that circle into quadrilaterals (i.e. cycles of length four).



■ **Figure 2** An example of a parsimonious scenario for a circle, its sub-circles and the scenario graph. a) A circular straight-line embedding Σ_C of a circle C and a parsimonious 2-break scenario ρ of length 4 transforming this circle into a terminal graph. b) The first 2-break of ρ transforms C into a union of two vertex-disjoint circles C_1 and C_2 , these are the sub-circles of C delimited by the edges $\{1, 8\}$ and $\{2, 7\}$ respectively. c) The edge $\{4, 7\}$ delimits the sub-circles C_3 and C_4 of C_2 . C_4 is also a sub-circle of C , but C_3 is not. d) A circular straight-line drawing $\Sigma_{\mathcal{S}(C, \rho)}$ of the scenario graph $\mathcal{S}(C, \rho)$. e) We draw an embedding of the trajectory graph [34] \mathcal{T} of ρ on top of the scenario graph. *Operation* nodes of \mathcal{T} (in white) correspond to the faces of $\mathcal{S}(C, \rho)$, and *adjacency* nodes of \mathcal{T} (in gray) correspond to the edges of $\mathcal{S}(C, \rho)$. The edges incident to the black (respectively gray) edges of the circle are directed outwards (respectively inwards). The rest of the edges are directed in such a way as to guarantee that the in and out degrees of the rest of the vertices of \mathcal{T} are equal. f) An example of a scenario graph with a face that is not incident to any of the edges of the circle.

► **Definition 11** (scenario graph). For a circle C and its parsimonious 2-break scenario ρ , define an undirected simple graph $\mathcal{S}(C, \rho)$ called a scenario graph. The vertices of $\mathcal{S}(C, \rho)$ are the vertices of C . If C has more than two vertices, then the edges of $\mathcal{S}(C, \rho)$ are the edges of the 2-breaks in ρ . Otherwise, $\mathcal{S}(C, \rho)$ contains a single edge incident to its two vertices.

See Figure 2 (d) for an example of a *circular straight-line drawing* of a scenario graph.

► **Definition 12** (circular straight-line drawing of a scenario graph). A *circular straight-line drawing* of a graph is a drawing on a plane with the vertices of the graph arranged on a circle and the edges drawn as straight lines. If the edges in the drawing do not cross, then the drawing is an embedding. Fix a circular straight-line embedding Σ_C of a circle C . The sub-circles of C inherit their circular straight-line embeddings from Σ_C . The scenario graph $\mathcal{S}(C, \rho)$ for a parsimonious scenario ρ for C inherits a circular straight-line drawing $\Sigma_{\mathcal{S}(C, \rho)}$ from Σ_C .

► **Theorem 13.** For every circle C and its parsimonious scenario ρ , the scenario graph $\mathcal{S}(C, \rho)$ is planar, $\Sigma_{\mathcal{S}(C, \rho)}$ is its embedding and all the internal faces of $\Sigma_{\mathcal{S}(C, \rho)}$ are quadrilaterals.

Proof. C has an even number of vertices by construction. If C has two vertices, then its scenario graph has a single edge and no internal faces. Suppose that the statement is true for every circle having at most $2k - 2 \geq 2$ vertices and take a circle C with $2k$ vertices together with its parsimonious scenario ρ .

Due to Corollary 7, the first 2-break τ of ρ transforms C into a union of two vertex disjoint circles. Denote these circles by C_1 and C_2 (see Figure 2 (b)). Due to Corollary 8, the rest of ρ can be partitioned into ρ_1 and ρ_2 , that are scenarios for C_1 and C_2 respectively.

If $\tau = \{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$, then the edges of $\Sigma_{\mathcal{S}(C, \rho)}$ can be obtained by adding the edges $\{u, v\}$ and $\{q, s\}$ to the union of the edges of $\mathcal{S}(C_1, \rho_1)$ and $\mathcal{S}(C_2, \rho_2)$ (see Figure 2 (d)). $\Sigma_{\mathcal{S}(C_1, \rho_1)}$ and $\Sigma_{\mathcal{S}(C_2, \rho_2)}$ satisfy the inductive hypothesis, so their edges do not cross. $\{u, v\}$ and $\{q, s\}$ do not cross the other edges and together with $\{u, q\}$ and $\{v, s\}$ bound the only face of $\Sigma_{\mathcal{S}(C, \rho)}$ not belonging to $\Sigma_{\mathcal{S}(C_1, \rho_1)}$ or $\Sigma_{\mathcal{S}(C_2, \rho_2)}$. ◀

► **Definition 14** (Dual graph). *The dual graph of an embedding of a planar graph G is a graph that has a vertex for each face of G . The dual graph has an edge whenever two faces of G are separated from each other by an edge.*

The dual graph of an embedding of a scenario graph, up to some minor modifications, is a *trajectory graph* introduced by Shao, Lin, and Moret (see Figure 2 (e) for an example). In [34] these authors show that the trajectory graph of a parsimonious scenario for a circle is a tree, which could be used to prove Theorem 13. By fixing an embedding of a scenario graph, we are not only able to describe the equivalence classes of the parsimonious scenarios for a circle, but also to search them efficiently.

4.2 A Bijection Between the 2-breaks in a Scenario and the Faces of a Scenario Graph

► **Lemma 15** (Proven in Appendix A.1). *For an embedding of a connected graph $G = (V, E)$ with $|V|$ edges incident to the outer face and all the inner faces being quadrilaterals, the number of internal faces $|F_{int}|$ is $\frac{|V|}{2} - 1$.*

► **Proposition 16.** *For a circle C and a parsimonious scenario ρ for C , there exists a bijection between the 2-breaks in ρ and the internal faces of $\Sigma_{\mathcal{S}(C, \rho)}$ that associates to every 2-break in ρ the face bounded by its edges.*

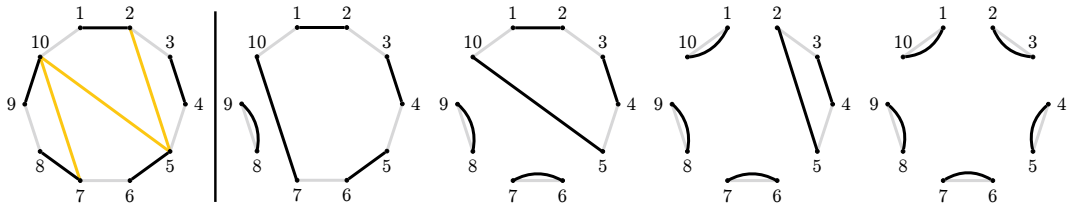
Proof. Take a circle $C = (V, E)$ and its parsimonious scenario ρ . There are $\frac{|V|}{2} - 1$ 2-breaks in ρ due to Theorem 6 and $\frac{|V|}{2} - 1$ internal faces in $\Sigma_{\mathcal{S}(C, \rho)}$ due to Lemma 15. Take a 2-break τ in ρ . By construction, the edges of τ form a cycle of length 4 in $\mathcal{S}(C, \rho)$. $\Sigma_{\mathcal{S}(C, \rho)}$ is an embedding of $\mathcal{S}(C, \rho)$ and in the part of the plane bounded by this cycle there are no vertices. This means that the edges of τ bound a face in $\Sigma_{\mathcal{S}(C, \rho)}$. All the 2-breaks in ρ are unique due to Corollary 7, thus we obtain an injection between the 2-breaks in ρ and the faces in $\Sigma_{\mathcal{S}(C, \rho)}$. As the sizes of these two sets are equal, the function is bijection. ◀

► **Corollary 17** (Proven in Appendix A.2). *Consider a circle C , a parsimonious scenario ρ for C , and an internal face f of $\Sigma_{\mathcal{S}(C, \rho)}$ bounded by the edges $\{u, v\}, \{v, s\}, \{s, q\}$ and $\{q, u\}$, where $\{u, v\}$ is a black edge of C . Then ρ contains a 2-break $\{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$.*

4.3 Partitioning Scenario into Scenarios for Sub-circles

In this section we demonstrate how a parsimonious scenario for a circle can be partitioned into scenarios for its sub-circles. Theorem 19 is at the heart of the dynamic programming algorithm for MCPS presented in Section 5.

► **Lemma 18** (Proven in Appendix A.3). *For every circle C , a parsimonious scenario ρ for C , and an edge e of $\Sigma_{\mathcal{S}(C, \rho)}$, ρ can be partitioned into two (possibly empty) parsimonious scenarios, one for each sub-circle delimited by e .*



■ **Figure 3** An illustration of how to find a parsimonious scenario whose scenario graph is equal to a given quadrangulation of a circle. On the left, the inner edges of a quadrangulation are depicted on top of a circle. To the right, a parsimonious scenario realizing this quadrangulation is given.

► **Theorem 19.** *For every circle C having at least four vertices, a parsimonious scenario ρ for C , and an internal face f of $\Sigma_{S(C,\rho)}$, ρ can be partitioned into four (possibly empty) parsimonious scenarios, one for each sub-circle delimited by the edges bounding f , plus the 2-break whose edges bound f .*

Proof. Proposition 16 provides a 2-break τ in ρ whose edges bound f . An edge bounding f delimits two sub-circles of C , denote the one containing only two vertices of τ to be *external* (the other one contains all four vertices of τ). Repeat Lemma 18 for the four edges bounding f to obtain parsimonious scenarios for the four external sub-circles. See Figure 2 (f) for an example, where all four external sub-circles are of size larger than 2. By construction, these scenarios do not intersect and together with τ partition ρ . ◀

4.4 A Bijection Between the Equivalence Classes of Scenarios and Quadrangulations

The results in this subsection are presented for the sake of completeness. They establish bijections between the equivalence classes of the parsimonious scenarios for a circle, their scenario graphs and the quadrangulations of that circle, however they are not explicitly used in the later sections.

► **Theorem 20** (Proven in Appendix A.4). *For every circle C and its parsimonious scenarios ρ and μ , their scenario graphs are equal if and only if these scenarios are equivalent.*

► **Definition 21** (quadrangulation of a circle). *Take a circular straight-line embedding of a circle C and forget the colors of its edges. If C has two vertices then keep a single edge joining them to obtain an embedding Σ_S . Otherwise add straight-line edges to obtain an embedding Σ_S with all the internal faces being quadrilaterals.*

► **Proposition 22** (Proven in Appendix A.5). *For every circle C and its quadrangulation Σ_S there exists a parsimonious scenario ρ such that $\Sigma_{S(C,\rho)} = \Sigma_S$.*

Quadrangulations of an even regular polygon, also known as complete quadrillages, correspond to rooted ternary trees just as triangulations of a polygon correspond to rooted binary trees [6]. This means that for a circle with $2n + 2$ vertices there are $\frac{1}{2n+1} \binom{3n}{n} \approx \frac{1}{n^{3/2}} \left(\frac{27}{4}\right)^n$ equivalence classes. For comparison, the number of the parsimonious scenarios for such a circle is $(n + 1)^{(n-1)}$ [31].

5 Minimum Cost Parsimonious Scenario for a Circle

Consider a circle C , the 2-breaks that act on the vertices of C , and a cost function ω mapping these 2-breaks to real numbers. In other words, every 2-break $\tau = \{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$, with u, v, q, s being vertices of C , gets assigned a ω -cost $\omega(\tau)$. The ω -cost

$\omega(\rho)$ of a scenario ρ is the sum of the costs of its constituent 2-breaks. The ω -cost $\omega(C)$ of a circle C is the minimum cost of a parsimonious scenario for C . Without loss of generality we suppose the vertices of C are named $\{1, \dots, |V|\}$ while respecting their clockwise order on the circle, and that $\{1, 2\}$ is a black edge as in Figure 2. For the vertices $i, j \in V$ with $i + j$ being odd, define $C[i, j]$ to be the sub-circle of C containing the path in C going clockwise from i to j . We denote its ω -cost by $\omega[i, j]$.

A quadrilateral is *valid for a circle* if it appears in some quadrangulation of that circle. Take vertices $1 \leq i < r < s < j \leq |V|$. They are the vertices of a valid quadrilateral for C if and only if $i + r, r + s, s + j, i + j$ are all odd (see Figure 2). Due to Theorem 19, such a quadrilateral partitions a parsimonious scenario for $C[i, j]$ into parsimonious scenarios for $C[i, r], C[r, s], C[s, j]$, along with the 2-break acting on the edges bounding the quadrilateral. In Lemma 23 we show how $\omega[i, j]$ can be found by iterating through the valid quadrilaterals for $C[i, j]$ containing the edge $\{i, j\}$. In Theorem 24 we conclude that $\omega(C)$ can be computed by iterating once through all the valid quadrilaterals of C .

► **Lemma 23** (Proven in Appendix A.6). *For a sub-circle $C[i, j]$ of a circle C with $i < j$, its cost $\omega[i, j]$ is 0 if it has two vertices, otherwise*

$$\omega[i, j] = \min_{r,s} (\omega[i, r] + \omega[r, s] + \omega[s, j] + \omega(\tau))$$

for vertices $i < r < s < j$ of a valid quadrilateral for $C[i, j]$, where $\tau = \{\{i, r\}, \{s, j\}\} \rightarrow \{\{i, j\}, \{r, s\}\}$ if i is odd, and $\tau = \{\{i, j\}, \{r, s\}\} \rightarrow \{\{i, r\}, \{j, s\}\}$ if i is even.

► **Theorem 24.** *If the ω -cost of a 2-break can be computed in constant time, then the ω -cost of a circle can be computed in $O(|V|^4)$ time using dynamic programming.*

Proof. Take a circle C . The ω -costs of its size 2 sub-circles can be initialized in $O(|V|)$ time. Suppose the ω -costs of its sub-circles of size at most $2k - 2 > 0$ to be precomputed. We will show that the ω -costs of its sub-circles of size $2k$ can be computed in $O(|V|k^2)$ time. There are $|V| - 2k + 1$ sub-circles of C of size $2k$. Due to Lemma 23, the ω -cost of such a sub-circle is equal to the minimum of the set of size $\Theta(k^2)$. Every element of this set can be computed in constant time, as the ω -costs of the smaller sub-circles are precomputed and the ω -cost of a 2-break can be found in constant time. This means that the time complexity of computing the ω -costs for all the sub-circles of C of size $2k$ is $\Theta((|V| - k)k^2)$. This leads to an $O(|V|^4)$ -time dynamic programming algorithm for computing $\omega(C)$. ◀

In [35] we demonstrated how the ω -cost of a breakpoint graph can be found using an algorithm finding the ω -cost of a circle. This leads directly to Theorem 25.

► **Theorem 25.** *Take two genomes having m linear chromosomes and sharing n syntenic blocks. If the ω -cost of a 2-break can be computed in constant time, then the ω -cost of their breakpoint graph can be computed in $O(mn^4)$ time.*

6 Experiments

We use OrthoCluster [45] to construct syntenic blocks between Human and four other species including Chimpanzee, Gibbon, Mouse, and Chicken. See Appendix B for details regarding this process. Several runs of OrthoCluster are performed for the species while setting the input parameter of the minimum number of genes in a syntenic block to be $\ell \in \{1, 2, 4, 8\}$. Details concerning these runs are summarized in Table 1. Ideally we would like to use the blocks with $\ell = 1$, however the blocks containing a small number of genes could be

less reliable due to annotation and assembly errors. For every set of syntenic blocks we constructed a breakpoint graph, computed the size of its largest simple cycle, that we call the *circumference*, and the length of a parsimonious scenario. Choosing an informative cost function is out of the scope of this work, however, we test the running time of our algorithm with a cost function ω_p that is equal to 1 for every 2-break (the full dynamic program ran despite all parsimonious scenarios having the same cost). If a breakpoint graph G has a circle C of size four, then any parsimonious scenario for G must contain a 2-break transforming C into a terminal graph. We call such 2-breaks *sure* and report the ratio of the number of sure 2-breaks to the length of a parsimonious scenario in Table 1.

6.1 The Elevated Time Complexity of Computing MCPS is not Prohibitive and Sure 2-breaks are Abundant

While the breakpoint graphs obtained with Chimpanzee have low circumference and could still be analyzed by hand, this is impossible for Gibbon and even more so for Mouse and Chicken. Computing the ω_p -cost took seconds for Chimpanzee and Gibbon, less than 3 minutes for Mouse, and less than 10 minutes for Chicken (see Table 1).

The parsimonious scenarios in all cases include a significant share of sure 2-breaks. We can test a given cost function on these 2-breaks, knowing that they must appear in any parsimonious scenario. This approach has been used before, for example, when ape/human deletions were recently studied [26, 22], however we are not aware of such a study for inversions or DCJs.

7 Biological Constraints

Recent studies revealed the role played by the structure of topologically associating domains (TADs), active chromatin, and 3D co-localization of the breakpoint regions in evolution by genome rearrangements. DNA breaks have been shown to occur in active chromatin regions coming into 3D contact in the nucleus [40, 10, 38]. Gibbon rearrangements were shown to occur at TAD boundaries, with most TADs maintained as intact modules during and after rearrangement [27]. A genome-wide depletion of deletions in active chromatin, and at TAD boundaries was observed across primate evolution [22]. Deletions causing TAD fusion were shown to be rare and under negative selection [26].

Recently published high resolution maps of the breakpoint regions, Hi-C, and ChIP-seq data for the highly rearranged Gibbon genome [27] combined with all the biological data available for Human genome, make these two species a very interesting target for the study of weighted genome rearrangements. This is especially so, since their parsimonious DCJ scenario contains $> 24\%$ of sure 2-breaks. These can be studied in detail to better understand the mechanisms behind the individual rearrangements. This dataset complements the set of inversions detected in human individuals [37, 16].

The algorithm presented in this article could be used to find parsimonious scenarios maximizing the number of DNA breaks at active chromatin or TAD boundaries. Breakpoint co-locality is another parameter that has been maximized in a scenario [39, 36]. The ω -cost could consider information on chromatin state, TAD boundaries and co-locality simultaneously, meaning that all the features could be used at the same time.

MCPS could also be used with the DCJ scenarios preserving common intervals [8]. In our framework, we could assign costs to the DCJs according to how well they preserve the common intervals between the genomes.

■ **Table 1** This table summarizes our analysis of the breakpoint graphs between Human and the four other species. ℓ is the minimum number of genes in a syntenic block. *Blocks* is the number of syntenic blocks. The *Coverage* is the proportion of the Human assembly covered by a syntenic block. *Scenario length* is the length of a parsimonious scenario. *Circumference* is the size of the largest simple cycle in a breakpoint graph. *Sure 2-breaks* is the proportion of a parsimonious scenarios that is composed of sure 2-breaks. *Time* is real time required to run our algorithm for ω_p -MCPS on Intel® Xeon® Processor E5-2650 v3 (25M Cache, 2.30 GHz).

ℓ	Blocks	Coverage	Scenario length	Circumference	Sure 2-breaks	Time
Chimpanzee						
1	202	90%	116	16	33%	<2s
2	100	90%	47	10	51%	<2s
4	68	90%	26	6	73%	<2s
8	50	89%	15	6	66%	<2s
Gibbon						
1	285	87%	195	80	24%	<7s
2	202	86%	130	56	28%	<3s
4	171	86%	105	60	32%	<3s
8	139	83%	82	36	37%	<3s
Mouse						
1	659	83%	523	116	17%	<3m
2	442	82%	366	94	15%	<1m
4	335	80%	287	94	11%	<1m
8	257	76%	224	72	9%	<20s
Chicken						
1	1697	75%	1429	210	12%	<10m
2	978	73%	851	98	10%	<2m
4	628	69%	558	82	7%	<2m
8	341	60%	305	60	8%	<15s

8 Conclusion and Future Work

Our recently introduced framework [35] deals with the φ -MCPS problem, which allows for costs on gene extremities and their adjacencies, extends results for co-tailed genomes to more general genome pairs. The ω -MCPS problem is a particular case of the φ -MCPS problem, as ω -costs depend only on gene extremities. Previous work provided polynomial-time algorithms for some particular φ -costs [39, 36, 14]. Finding a general polynomial-time algorithm for the φ -MCPS problem, however, remains an open problem.

Our characterization of parsimonious scenarios into equivalence classes could prove useful for tasks such as sampling weighted parsimonious scenarios. The problem of counting the number of the parsimonious scenarios within an equivalence class remains open, as does the problem of counting and sampling the ω -MCPS scenarios.

Our algorithm for ω -MCPS was shown to be efficient enough for species as distant as Human and Chicken, and we briefly discussed possibly informative biological constraints and cost functions. Mathematical modeling of these constraints and a choice of a particular cost function, however, are left for future work.

References

- 1 A. Amir and A. Levy. String rearrangement metrics: A survey. In *Algorithms and Applications*, pages 1–33. Springer, 2010.
- 2 Y. Anselmetti, W. Duchemin, E. Tannier, C. Chauve, and S. Bérard. Phylogenetic signal from rearrangements in 18 Anopheles species by joint scaffolding extant and ancestral genomes. *BMC genomics*, 19(2):96, 2018.
- 3 Y. Anselmetti, N. Luhmann, S. Bérard, E. Tannier, and C. Chauve. Comparative Methods for Reconstructing Ancient Genome Organization. In *Comparative Genomics*, pages 343–362. Springer, 2018.
- 4 P. Avdeyev, N. Alexeev, Y. Rong, and M.A. Alekseyev. A unified ILP framework for genome median, halving, and aliquoting problems under DCJ. In *RECOMB International Workshop on Comparative Genomics*, pages 156–178. Springer, 2017.
- 5 V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.
- 6 Y. Baryshnikov. On Stokes sets. In *New developments in singularity theory*, pages 65–86. Springer, 2001.
- 7 C. Baudet, U. Dias, and Z. Dias. Sorting by weighted inversions considering length and symmetry. *BMC bioinformatics*, 16(19):S3, 2015.
- 8 S. Bérard, A. Chateau, C. Chauve, C. Paul, and E. Tannier. Computation of perfect DCJ rearrangement scenarios with linear and circular chromosomes. *Journal of Computational Biology*, 16(10):1287–1309, 2009.
- 9 A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In *International Workshop on Algorithms in Bioinformatics*, pages 163–173. Springer, 2006.
- 10 C. Berthelot, M. Muffato, J. Abecassis, and H.R. Crollius. The 3D organization of chromatin explains evolutionary fragile genomic regions. *Cell reports*, 10(11):1913–1924, 2015.
- 11 D. Bienstock and O. Günlük. A degree sequence problem related to network design. *Networks*, 24(4):195–205, 1994.
- 12 P. Biller, C. Knibbe, L. Guéguen, and E. Tannier. Breaking good: accounting for the diversity of fragile regions for estimating rearrangement distances. *Genome Biol Evol*, 8:1427–39, 2016.
- 13 M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172(1):GC11–GC17, 1996.
- 14 L. Bulteau, G. Fertin, and E. Tannier. Genome rearrangements with indels in intergenes restrict the scenario space. *BMC bioinformatics*, 17(14):426, 2016.
- 15 A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM journal on discrete mathematics*, 12(1):91–110, 1999.
- 16 M.J.P. Chaisson, A.D. Sanders, X. Zhao, A. Malhotra, D. Porubsky, T. Rausch, E.J. Gardner, O.L. Rodriguez, L. Guo, R.L. Collins, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nature communications*, 10, 2019.
- 17 P.E.C. Compeau. A generalized cost model for DCJ-indel sorting. In *International Workshop on Algorithms in Bioinformatics*, pages 38–51. Springer, 2014.
- 18 F. Farnoud and O. Milenkovic. Sorting of permutations by cost-constrained transpositions. *IEEE Transactions on Information Theory*, 58(1):3–23, 2012.
- 19 T. Feder, A. Guetz, M. Mihail, and A. Saberi. A local switch Markov chain on given degree graphs with application in connectivity of peer-to-peer networks. In *Foundations of Computer Science, 2006. FOCS’06. 47th Annual IEEE Symposium on*, pages 69–76. IEEE, 2006.
- 20 G. Fertin, G. Jean, and E. Tannier. Algorithms for computing the double cut and join distance on both gene order and intergenic sizes. *Algorithms for Molecular Biology*, 12(1):16, 2017.
- 21 B.K. Fostdick, D.B. Larremore, J. Nishimura, and J. Ugander. Configuring random graph models with fixed degree sequences. *SIAM Review*, 60(2):315–355, 2018.
- 22 G. Fudenberg and K.S. Pollard. Chromatin features constrain structural variation across evolutionary timescales. *Proceedings of the National Academy of Sciences*, 116(6), 2019.

- 23 T. Hartmann, M. Bernt, and M. Middendorf. An exact algorithm for sorting by weighted preserving genome rearrangements. *IEEE/ACM transactions on computational biology and bioinformatics*, 16(1):52–62, 2019.
- 24 T. Hartmann, M. Middendorf, and M. Bernt. Genome Rearrangement Analysis: Cut and Join Genome Rearrangements and Gene Cluster Preserving Approaches. In *Comparative Genomics*, pages 261–289. Springer, 2018.
- 25 T. Hartmann, N. Wieseke, R. Sharan, M. Middendorf, and M. Bernt. Genome rearrangement with ILP. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(5), 2018.
- 26 L. Huynh and F. Hormozdiari. TAD fusion score: discovery and ranking the contribution of deletions to genome structure. *Genome biology*, 20(1):60, 2019.
- 27 N.H. Lazar, K.A. Nevonen, B. O’Connell, C. McCann, R.J. O’Neill, R.E. Green, T.J. Meyer, M. Okhovat, and L. Carbone. Epigenetic maintenance of topological domains in the highly rearranged gibbon genome. *Genome research*, 28(7):983–997, 2018.
- 28 Y. Lin, V. Rajan, and B.M.E. Moret. TIBA: a tool for phylogeny inference from rearrangement data with bootstrap analysis. *Bioinformatics*, 28(24):3324–3325, 2012.
- 29 F.V. Martinez, P. Feijao, M.D.V. Braga, and J. Stoye. On the family-free DCJ distance. In *International Workshop on Algorithms in Bioinformatics*, pages 174–186. Springer, 2014.
- 30 I. Miklós and E. Tannier. Approximating the number of Double Cut-and-Join scenarios. *Theoretical Computer Science*, 439:30–40, 2012.
- 31 A. Ouangraoua and A. Bergeron. Combinatorial structure of genome rearrangements scenarios. *Journal of Computational Biology*, 17(9):1129–1144, 2010.
- 32 S. Pulicani, P. Simonaitis, E. Rivals, and K.M. Swenson. Rearrangement scenarios guided by chromatin structure. In *RECOMB International Workshop on Comparative Genomics*, pages 141–155. Springer, 2017.
- 33 M. Shao, Y. Lin, and B. Moret. An exact algorithm to compute the DCJ distance for genomes with duplicate genes. In *International Conference on Research in Computational Molecular Biology*, pages 280–292. Springer, 2014.
- 34 M. Shao, Y. Lin, and B.M.E. Moret. Sorting genomes with rearrangements and segmental duplications through trajectory graphs. In *BMC bioinformatics*. BioMed Central, 2013.
- 35 P. Simonaitis, A. Chateau, and K.M. Swenson. A General Framework for Genome Rearrangement with Biological Constraints. In *RECOMB International conference on Comparative Genomics*, pages 49–71. Springer, 2018.
- 36 P. Simonaitis and K.M. Swenson. Finding local genome rearrangements. *Algorithms for Molecular Biology*, 13(1):9, 2018.
- 37 P.H. Sudmant, T. Rausch, E.J. Gardner, R.E. Handsaker, A. Abyzov, J. Huddleston, Y. Zhang, K. Ye, G. Jun, M.H. Fritz, et al. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75, 2015.
- 38 K.M. Swenson and M. Blanchette. Large-scale mammalian genome rearrangements coincide with chromatin interactions. *Bioinformatics*, July 2019.
- 39 K.M. Swenson, P. Simonaitis, and M. Blanchette. Models and algorithms for genome rearrangement with positional constraints. *Algorithms for Molecular Biology*, 11(1):13, 2016.
- 40 A. Veron, C. Lemaitre, C. Gautier, V. Lacroix, and M.-F. Sagot. Close 3D proximity of evolutionary breakpoints argues for the notion of spatial synteny. *BMC Genomics*, 12, 2011.
- 41 R. Warren and D. Sankoff. Genome halving with double cut and join. *Journal of Bioinformatics and Computational Biology*, 7(02):357–371, 2009.
- 42 S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.
- 43 R. Zeira and R. Shamir. Sorting cancer karyotypes using double-cut-and-joins, duplications and deletions. *Bioinformatics (Oxford, England)*, 2018.
- 44 R. Zeira and R. Shamir. Genome Rearrangement Problems with Single and Multiple Gene Copies: A Review. In *Bioinformatics and Phylogenetics*, pages 205–241. Springer, 2019.
- 45 X. Zeng, M.J. Nesbitt, J. Pei, K. Wang, I.A. Vergara, and N. Chen. OrthoCluster: a new tool for mining synteny blocks and applications in comparative genomics. In *Proceedings of the 11th international conference on Extending database technology*. ACM, 2008.

A Proofs

A.1 Lemma 15

► **Lemma.** *For an embedding of a connected graph $G = (V, E)$ with $|V|$ edges incident to the outer face and all the inner faces being quadrilaterals, the number of internal faces $|F_{int}|$ is $\frac{|V|}{2} - 1$.*

Proof. Every internal face of G is bounded by four edges, G has $|V|$ edges incident to the outer face, and every edge of G is separating two faces. By counting edges we obtain the equality $4|F_{int}| + |V| = 2|E|$. Euler's formula for planar graphs is $|V| - |E| + |F| = 2$, thus $|E| = |V| + |F| - 2 = |V| + |F_{int}| - 1$. Combining these two equalities we obtain $4|F_{int}| + |V| = 2|V| + 2|F_{int}| - 2$ and finally $|F_{int}| = \frac{|V|}{2} - 1$. ◀

A.2 Corollary 17

► **Corollary.** *Consider a circle C , a parsimonious scenario ρ for C , and an internal face f of $\Sigma_{\mathcal{S}(C, \rho)}$ bounded by the edges $\{u, v\}$, $\{v, s\}$, $\{s, q\}$ and $\{q, u\}$, where $\{u, v\}$ is a black edge of C . Then ρ contains a 2-break $\{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$.*

Proof. If $\{u, v\}$ is among the edges of at least two different 2-breaks in ρ , then $\{u, v\}$ is incident to at least two different internal faces of $\Sigma_{\mathcal{S}(C, \rho)}$ due to Proposition 16. However $\{u, v\}$ is a black edge of C , thus it is incident to a single internal face of $\Sigma_{\mathcal{S}(C, \rho)}$, and that face is f . This means that $\{u, v\}$ is among the edges of a single 2-break τ in ρ .

Due to Proposition 16, the edges of τ are exactly the edges of f . This means that τ is either $\{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$, or $\{\{u, q\}, \{v, s\}\} \rightarrow \{\{u, v\}, \{q, s\}\}$. $\{u, v\}$ is not present in the terminal graph obtained from C after ρ is performed. This means that ρ includes a 2-break replacing $\{u, v\}$. Combining these two observations we conclude that τ is $\{\{u, v\}, \{q, s\}\} \rightarrow \{\{u, q\}, \{v, s\}\}$. ◀

A.3 Lemma 18

► **Lemma.** *For every circle C , a parsimonious scenario ρ for C , and an edge e of $\Sigma_{\mathcal{S}(C, \rho)}$, ρ can be partitioned into two (possibly empty) parsimonious scenarios, one for each sub-circle delimited by e .*

Proof. If C has two vertices, then the sub-circles in question are two copies of C , and empty scenarios satisfy the statement. Suppose that it is true for any circle having at most $2k - 2 \geq 2$ vertices and take a circle with $2k$ vertices together with its parsimonious scenario ρ and an edge e of $\Sigma_{\mathcal{S}(C, \rho)}$.

Denote the two sub-circles delimited by e by C_3 and C_4 . Take the first 2-break τ of ρ and suppose that its vertices do not belong to the same sub-circle. In this case at least one of its edges is incident to the vertices that do not belong to the same sub-circle. This edge crosses e , which contradicts the planarity of $\Sigma_{\mathcal{S}(C, \rho)}$. This means that the vertices of τ belong to the same sub-circle. Without loss of generality we can suppose that it is C_3 . τ transforms C into a union of two vertex-disjoint circles. Denote the one containing the endpoints of e by C_2 and the other one by C_1 . Due to Corollary 7, the rest of ρ can be partitioned into two sub-sequences ρ_1 and ρ_2 , that are respectively scenarios for C_1 and C_2 .

The inductive hypothesis holds for the triplet C_2 , ρ_2 and e , providing us with a partition of ρ_2 into two parsimonious scenarios for the sub-circles of C_2 delimited by e . One of these sub-circles is C_4 . Denote the other by C_5 and denote the parsimonious scenarios obtained for them by ρ_4 and ρ_5 respectively.

By removing from ρ the 2-breaks in ρ_4 and a 2-break τ , we obtain a sequence of 2-breaks ρ' , that is a shuffle of the parsimonious scenarios ρ_1 for C_1 and ρ_5 for C_5 . Due to Corollary 8, ρ' is a parsimonious scenario for the union of C_1 and C_5 , which is the graph obtained from C_3 after τ is performed. Finally, by adding τ at the beginning of ρ' we obtain a parsimonious scenario for C_3 , which together with ρ_4 satisfies the statement. \blacktriangleleft

A.4 Theorem 20

We first prove the following lemma.

► **Lemma.** *For every circle C , a parsimonious scenario ρ for C , and a 2-break in ρ replacing two black edges of C , the sequence of 2-breaks with this 2-break performed first and followed by the rest of ρ is also a parsimonious scenario for C .*

Proof. Take a circle C , its parsimonious scenario ρ and a 2-break τ in ρ replacing two black edges. Due to Proposition 16, there exists a face in $\Sigma_{\mathcal{S}(C,\rho)}$ bounded by the edges of τ . Using Theorem 19 we obtain a partition of ρ into four scenarios for the sub-circles of C delimited by the edges of τ and τ itself, however two of these scenarios are for the circles with 2 vertices, thus of length 0. Denote the other two by ρ_1 and ρ_2 . This means that ρ without τ is a shuffle of two parsimonious scenarios for the vertex-disjoint circles. Using Corollary 8 we obtain that ρ without τ is a parsimonious scenario for the union of these circles, that is obtained from C after τ is performed. This means that moving τ to the beginning of ρ we obtain a parsimonious scenario for C . \blacktriangleleft

► **Theorem.** *For every circle C and its parsimonious scenarios ρ and μ , their scenario graphs are equal if and only if these scenarios are equivalent.*

Proof. If a circle C has two vertices, then its parsimonious scenarios are empty and the statement is true. Suppose that the statement is true for every circle with at most $2k - 2 \geq 2$ vertices. Take a circle C with $2k$ vertices and two parsimonious scenarios ρ and μ for C .

First suppose that ρ and μ contain exactly the same 2-breaks but possibly in different order. By construction, the edges of $\Sigma_{\mathcal{S}(C,\rho)}$ are the edges of the 2-breaks in ρ , and these are exactly the same as the edges of the 2-breaks in μ , meaning that $\Sigma_{\mathcal{S}(C,\rho)} = \Sigma_{\mathcal{S}(C,\mu)}$.

Now suppose that $\Sigma_{\mathcal{S}(C,\rho)} = \Sigma_{\mathcal{S}(C,\mu)}$. Take τ , the first 2-break of ρ . By construction, it replaces two black edges of C and its edges bound a face in $\Sigma_{\mathcal{S}(C,\rho)}$ due to Proposition 16. $\Sigma_{\mathcal{S}(C,\rho)} = \Sigma_{\mathcal{S}(C,\mu)}$, thus the latter contains the same face that we denote by f . Due to Corollary 17, μ contains a 2-break replacing the black edges bounding f , this means that μ contains τ . Using the lemma proven above with a 2-break τ we get that the sequence of 2-breaks obtained by moving μ to the beginning of μ is a parsimonious scenario for C that we denote by μ' . Using the first part of this proof we obtain that $\Sigma_{\mathcal{S}(C,\mu')}$ is equal to $\Sigma_{\mathcal{S}(C,\mu)}$ and thus $\Sigma_{\mathcal{S}(C,\rho)}$.

Due to Corollary 7, τ transforms C into two vertex-disjoint circles C_1 and C_2 . Denote the subgraph of $\Sigma_{\mathcal{S}(C,\rho)}$ induced by the vertices of C_1 (respectively C_2) by $\Sigma_{\mathcal{S}_1}$ (respectively $\Sigma_{\mathcal{S}_2}$). Due to Corollary 8, the rest of ρ (respectively μ') can be partitioned into two sub-sequences ρ_1, ρ_2 (respectively μ'_1, μ'_2) that are scenarios for C_1 and C_2 . By construction, $\Sigma_{\mathcal{S}(C,\rho_1)} = \Sigma_{\mathcal{S}_1}$ and $\Sigma_{\mathcal{S}(C,\rho_2)} = \Sigma_{\mathcal{S}_2}$ (respectively $\Sigma_{\mathcal{S}(C,\mu'_1)} = \Sigma_{\mathcal{S}_1}$ and $\Sigma_{\mathcal{S}(C,\mu'_2)} = \Sigma_{\mathcal{S}_2}$). Using the inductive hypothesis we obtain that ρ_1 and μ'_1 contain exactly the same 2-breaks and so does ρ_2 and μ'_2 , which means that ρ and μ' contain exactly the same 2-breaks. \blacktriangleleft

A.5 Proposition 22

► **Proposition.** *For every circle C and its quadrangulation Σ_S there exists a parsimonious scenario ρ such that $\Sigma_{S(C,\rho)} = \Sigma_S$.*

Proof. If a circle has 2 vertices, then an empty scenario satisfies the statement. Suppose that the statement is true for every circle having at most $2k - 2 \geq 2$ vertices and take a circle C with $2k$ vertices together with its quadrangulation Σ_S (see Figure 3 for an example).

Due to Lemma 15, Σ_S has $k - 1$ internal face and k outer edges that are black in C . An outer edge of C belongs to at most one internal face of Σ_S , thus we obtain that Σ_S has an internal face f bounded by the edges among which there are at least two black edges of C .

Perform a 2-break on these black edges of C transforming it into a vertex-disjoint union of two sub-circles C_1 and C_2 . The subgraphs of Σ_S induced by the vertices of C_1 and C_2 are quadrangulations of C_1 and C_2 that we denote by Σ_{S_1} and Σ_{S_2} . They satisfy the inductive hypothesis and provide us with parsimonious scenarios ρ_1 and ρ_2 for circles C_1 and C_2 . Combining them with the initial 2-break we obtain a parsimonious scenario ρ for C with $\Sigma_{S(C,\rho)} = \Sigma_S$. ◀

A.6 Lemma 23

► **Lemma.** *For a sub-circle $C[i, j]$ of a circle C with $i < j$, its cost $\omega[i, j]$ is 0 if it has two vertices, otherwise*

$$\omega[i, j] = \min_{r,s} (\omega[i, r] + \omega[r, s] + \omega[s, j] + \omega(\tau))$$

for vertices $i < r < s < j$ of a valid quadrilateral for $C[i, j]$, where $\tau = \{\{i, r\}, \{s, j\}\} \rightarrow \{\{i, j\}, \{r, s\}\}$ if i is odd, and $\tau = \{\{i, j\}, \{r, s\}\} \rightarrow \{\{i, r\}, \{j, s\}\}$ if i is even.

Proof. If $j = i + 1$, then $C[i, j]$ has two vertices, its parsimonious scenario is of length 0, and thus of cost 0. We start by showing a lower bound on $\omega[i, j]$. Take a scenario ρ_o of minimum cost among the parsimonious scenarios for $C[i, j]$, and a face f in $\Sigma_{S(C[i,j],\rho_o)}$ incident to the edge $\{i, j\}$. Denote its other two vertices by r and s with $r < s$. By construction $i + r$, $r + s$ and $s + j$ are all odd. Due to Theorem 19, ρ_o can be partitioned into four parsimonious scenarios ρ_1, ρ_2, ρ_3 and ρ_4 for the sub-circles of $C[i, j]$ delimited by the edges bounding f plus a 2-break τ whose edges bound f . The sub-circles of $C[i, j]$ in question are respectively $C[i, r]$, $C[r, s]$, $C[s, j]$ and a circle of size 2 containing the edge $\{i, j\}$. If i is even, then $\{i, i + 1\}$ is gray and $\{i, j\}$ is black by construction. Using Corollary 17 we obtain that τ is $\{\{i, j\}, \{r, s\}\} \rightarrow \{\{i, r\}, \{j, s\}\}$. Similarly τ is $\{\{i, r\}, \{s, j\}\} \rightarrow \{\{i, j\}, \{r, s\}\}$ if i is odd. This establishes that $\omega(C) = \omega(\rho_o) = \omega(\rho_1) + \omega(\rho_2) + \omega(\rho_3) + \omega(\rho_4) + \omega(\tau) \geq \omega[i, r] + \omega[r, s] + \omega[s, j] + 0 + \omega(\tau)$.

Now we show the upper bound on $\omega(C)$. Take $i < r < s < j$ with odd $i + r, r + s, s + j$, and minimum cost parsimonious scenarios ρ_1, ρ_2 and ρ_3 for the sub-circles $C[i, r]$, $C[r, s]$ and $C[s, j]$. If i is even, then $\{i, i + 1\}$ is gray, thus $\{i, j\}$ is black in $C[i, j]$, and $\{r, s\}$ is gray in $C[r, s]$. This means that we can perform ρ_2 on $C[i, j]$, and then follow it by the 2-break τ equal to $\{\{i, j\}, \{r, s\}\} \rightarrow \{\{i, r\}, \{j, s\}\}$, followed by ρ_1 and ρ_3 . This is a parsimonious scenario for $C[i, j]$ that we denote ρ , and $\omega[i, j] \leq \omega(\rho) = \omega(\rho_2) + \omega(\tau) + \omega(\rho_1) + \omega(\rho_3) = \omega[r, s] + \omega(\tau) + \omega[i, r] + \omega[s, j]$. If i is odd, then similarly we obtain that $\{i, r\}$ and $\{s, j\}$ are both black in $C[i, r]$ and $C[s, j]$ respectively, and that ρ_1 followed by ρ_3 , a 2-break τ equal to $\{\{i, r\}, \{s, j\}\} \rightarrow \{\{i, j\}, \{r, s\}\}$ and ρ_2 is a parsimonious scenario for $C[i, j]$ that we denote by ρ . This leads to the inequality $\omega[i, j] \leq \omega(\rho) = \omega(\rho_1) + \omega(\rho_3) + \omega(\tau) + \omega(\rho_2) = \omega[i, r] + \omega[s, j] + \omega(\tau) + \omega[r, s]$. ◀

B Methods

B.1 Downloading and Preprocessing Orthologs

Protein coding Human (GRCh38.p12), Chimpanzee (Pan_tro_3.0), Gibbon (Nleu_3.0), Mouse (GRCm38.p6) and Chicken (GRCg6a) genes were downloaded from **Ensemble Release 96 using Biomart**. Orthologous groups of protein coding genes between Human and the rest of the species were downloaded from the same database. Ensemble provides orthologous pairs with a **confidence score** that is either low or high. Low confidence pairs were filtered out together with the orthologous pairs containing genes in unlocalized or unplaced scaffolds. At this point we were left with 17319 orthologous pairs for Chimpanzee, 14923 for Gibbon, 16848 for Mouse, and 13024 for Chicken.

B.2 OrthoCluster: Identifying syntenic blocks

For the construction of syntenic blocks we chose to use OrthoCluster [45] due to its ease of use, speed, and allowance for different parametric constraints. OrthoCluster takes the orthologous groups as input and identifies the syntenic blocks. It handles many-to-many orthologs and overlapping genes, thus no further filtering of the previously prepared orthologs was required. OrthoCluster deals with various types of mismatches, however in this study we required the genes in the syntenic blocks to share exactly the same order and strandedness, and contain no mismatches. We processed these blocks by deleting the ones that were included in some other blocks, and by merging those blocks that were consecutive in both genomes and either 1) had the same order and strandedness, or 2) opposite order and strandedness in both genomes. Table 1 contains the coverage of the blocks before merging, and the number of the blocks after merging. The final blocks might have some overlaps due to the overlapping genes.

B.3 OrthoCluster Identifies Blocks of High Coverage Without Performing a Whole Genome Alignment

Golden path length of the Human assembly GRCh38.p12 is around 3.1Gbp. For a set of syntenic blocks, we divide the number of the nucleotides from Human included in at least one of the blocks by this number to obtain the *coverage of the blocks*. The number of the blocks and their coverage computed using OrthoCluster are comparable to those of the blocks available on **Ensemble Release 96 Compara** database. These were computed using whole genome alignments and their number of blocks and coverage are respectively 192 and 92% for Chimpanzee, 277 and 90% for Gibbon, 363 and 88% for Mouse, 398 and 68% for Chicken.

Fast and Accurate Structure Probability Estimation for Simultaneous Alignment and Folding of RNAs

Milad Miladi 

Bioinformatics Group, Department of Computer Science, University of Freiburg, Germany
miladim@informatik.uni-freiburg.de

Martin Raden 

Bioinformatics Group, Department of Computer Science, University of Freiburg, Germany
mmann@informatik.uni-freiburg.de

Sebastian Will 

Theoretical Biochemistry Group (TBI), Institute for Theoretical Chemistry,
University of Vienna, Austria
will@tbi.univie.ac.at

Rolf Backofen 

Bioinformatics Group, Department of Computer Science, University of Freiburg, Germany
Signalling Research Centres BIOS and CIBSS, University of Freiburg, Germany
backofen@informatik.uni-freiburg.de

Abstract

Motivation: Simultaneous alignment and folding (SA&F) of RNAs is the indispensable gold standard for inferring the structure of non-coding RNAs and their general analysis. The original algorithm, proposed by Sankoff, solves the theoretical problem exactly with a complexity of $O(n^6)$ in the full energy model. Over the last two decades, several variants and improvements of the Sankoff algorithm have been proposed to reduce its extreme complexity by proposing simplified energy models or imposing restrictions on the predicted alignments.

Results: Here we introduce a novel variant of Sankoff's algorithm that reconciles the simplifications of PMcomp, namely moving from the full energy model to a simpler base pair-based model, with the accuracy of the loop-based full energy model. Instead of estimating pseudo-energies from unconditional base pair probabilities, our model calculates energies from conditional base pair probabilities that allow to accurately capture structure probabilities, which obey a conditional dependency. Supporting modifications with surgical precision, this model gives rise to the fast and highly accurate novel algorithm Pankov (*Probabilistic Sankoff-like simultaneous alignment and folding of RNAs inspired by Markov chains*). Pankov benefits from the speed-up of excluding unreliable base-pairing without compromising the loop-based free energy model of the Sankoff's algorithm. We show that Pankov outperforms its predecessors LocARNA and SPARSE in folding quality and is faster than LocARNA. Pankov is developed as a branch of the LocARNA package and available at <https://github.com/mmiladi/Pankov>.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Computational biology; Applied computing → Molecular structural biology

Keywords and phrases RNA secondary structure, Structural bioinformatics, Alignment, Algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.14

Funding MM and MR are supported by the German Research Foundation (DFG *BA2168/14-1* and *BA2168/16-1*). SW is supported by the Austrian Science Fund (FWF *I 2874-N28*). RB is supported by the German Research Foundation (DFG) under Germany's Excellence Strategy (CIBSS – EXC-2189 – Project ID 390939984).

Acknowledgements The authors would like to thank the anonymous reviewers.



© Milad Miladi, Martin Raden, Sebastian Will, and Rolf Backofen;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 14; pp. 14:1–14:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In all forms of life, RNAs play essential roles that go beyond coding as messenger RNAs for the synthesis of proteins. Non-coding RNAs (ncRNAs) directly regulate cellular mechanisms, where some are known to be conserved for billions of years [2]. ncRNAs often have only weak sequence conservation, since their (conserved) structure crucially determines their function. Therefore, inferring the conserved secondary structure of homologs – most often, based on RNA alignments, is central for the discovery and annotation of functional RNAs.

RNA structural alignment algorithms can be classified depending on whether they fold and align simultaneously or in turn [19]. The gold standard for computing reliable alignments (and common structures) of RNAs is still the simultaneous algorithm proposed by Sankoff in 1985 [18]. By simultaneously aligning and folding the RNAs, it resolves the vicious cycle that reliable RNA alignments must consider RNA structures (especially for RNAs of medium to low sequence identity [6]), while computational structure prediction is typically unreliable without comparative information. For a pair of RNA sequences, the algorithm finds the optimal alignment and two compatible secondary structures that minimize the total of sequence-alignment distance score and the free energies of the predicted structures. With a run-time complexity of $O(n^6)$ and $O(n^4)$ for memory, the method requires extreme computational resources, such that its application is largely restricted to small instances and data sets. Efficient alignment algorithms are needed for the multiple alignments of the clusters that can be obtained from large scale clustering of data from high-throughput sequencing experiments [16].

Thus, several approaches have adapted Sankoff's original algorithm to reduce its computational costs. Two main lines of the variants can be distinguished. Methods like Dynalign [13] and FoldAlign [9] reduce the computational complexity by heuristic restrictions, e.g. introducing banding strategies or limiting the maximum size of the comparable sub-structures. Since such methods need to perform expensive energy computations in the nearest neighbor model [14], their applications are still limited without considering heuristic restrictions that in turn could compromise their accuracy.

A highly viable alternative was proposed with the PMcomp algorithm [10], which replaces the nearest neighbor energy model with a still accurate probabilistic model. This model allows to drastically simplify the algorithms, which strongly reduces the computational overhead and supports further algorithmic optimizations in PMcomp's successors. For reducing the overhead, PMcomp-type algorithms evaluate structures based on base pair probabilities, which are precomputed by McCaskill's algorithm [15] instead of calculating nearest neighbor energy terms. Moreover, PMcomp-type algorithms such as LocARNA and similar approaches with a probabilistic energy model [22, 20, 8, 11] further speed up by reducing the search space. To this end, LocARNA considers only base-pairs that pass a defined probability threshold. This sparsification improves over PMcomp's complexity of $O(n^6)$ time and $O(n^4)$ space, each, by a quadratic factor (resulting in the $O(n^4)$ time and $O(n^2)$ space requirements of LocARNA).

With the algorithm SPARSE [21], we introduced a second level of sparsification using loop-closing aware recursions to filter based on the joint probability of base-pairs and associated loop-closing base-pairs. This sparsification reduces the time complexity of the alignment algorithm to $O(n^2)$, starting from the precomputed probabilities. The joint probabilities were computed based on our extension of the McCaskill's algorithm [17].

We emphasize that all these methods, starting with the original Sankoff algorithm, consider only non-crossing structures, even if crossing base pairs occur relatively frequently in physical structures [3]. This is generally justified, since most secondary structures are

dominated by non-crossing base pairs; in turn, the limitation to non-crossing structures allows dynamic programming techniques, which are far more efficient and flexible than comparable techniques that consider (even limited forms of) base pair crossings.

In this work, we utilize the joint probabilities in a novel way – not only for strong sparsification as in SPARSE but as well to evaluate RNA structure more accurately in a PMcomp-type algorithm. We start with showing that joint (or equivalently, conditional) probabilities allow to precisely capture structure probabilities in the nearest neighbor energy model. This corresponds to the exact capture of the nearest neighbor energies themselves. Remarkably, while previous work discussed only the stacking base-pair helices [1], we cope with all loop types. Based on the exact model, we suggest careful simplifications, that allow incorporation of the model into alignment and folding (in the variant of the SPARSE algorithm). Based on the novel precise probabilistic model, we propose the novel algorithm Pankov with $O(n^2)$ time complexity. As fundamental novelty over its predecessors, it applies an accurate full-loop energy model for evaluating the structures.

Performing an established benchmark, we show that Pankov is in practice faster than LocARNA *and* significantly improves structure prediction over both SPARSE and LocARNA. Compared to SPARSE, it even improves the sequence alignment quality.

2 Methods

2.1 Preliminaries

Basic notations

An *RNA sequence* A is a string over the alphabet $\{\mathbf{A}, \mathbf{C}, \mathbf{G}, \mathbf{U}\}$. A *base pair* a of A is a pair (a^L, a^R) ($1 \leq a^L < a^R \leq |A|$) such that the respective sequence positions are complementary, i.e. AU, GC or GU. A *non-crossing RNA structure* S of A , in the following called *structure*, is a set of base pairs, where each two different base pairs (i, j) and (i', j') of S do not cross, i.e. $i < i' < j < j'$, and do not share any end, i.e. i, j, i' , and j' are pairwise different. To treat the external bases pairs of an RNA structure, we introduce a pseudo-base-pair $a_0 := (0, |A| + 1)$, which formally encloses all base pairs of A .

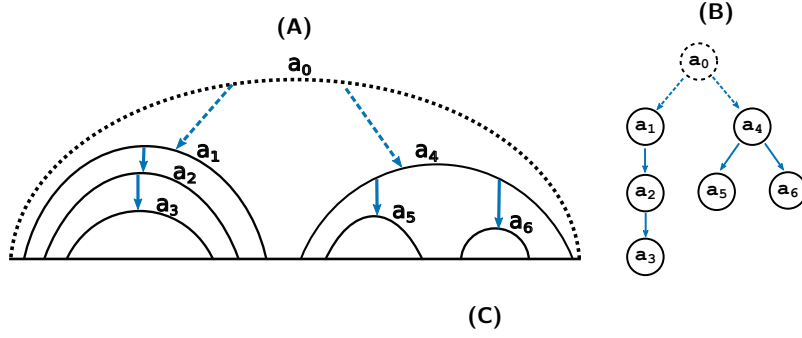
Tree structure

A nested RNA secondary structure S can be represented as a rooted structure tree, exemplified in Figure 1(A,B), where base-pairs are encoded as nodes and the enclosing base-pairs are the *parents* of the directly enclosed base-pairs. The $\text{chi}(a \in S)$ function provides the set of children base pairs that are directly enclosed by a given base pair a . Thus, the cardinality of $\text{chi}(a)$ is zero for hairpin loops (a_3, a_5, a_6 in Fig. 1), at least two for multi-loops (a_4) and one otherwise (a_1, a_2), which represents stackings, bulges or interior loops. Furthermore, the pseudo base pair a_0 recursively encloses all base pairs of any structure that can be formed by A .

Energy and probability of an RNA structure

The *energy of a structure* S can be estimated using the nearest neighbor energy model, which is based on a loop-decomposition of the structure, where a loop is defined as the substructure defined by a base pair a and its enclosed base pairs $\text{chi}(a)$. The energy model provides *contributions* $E^{\text{loop}}(a, \text{chi}(a))$ for such loops, which are summed up, i.e.

$$E(S) = \sum_{a \in S} E^{\text{loop}}(a, \text{chi}(a)). \quad (1)$$



$$\begin{aligned} \mathbf{P}^{\text{PMcomp}}(\mathbf{S}) &= \mathbf{P}(\mathbf{a}_1) \cdot \mathbf{P}(\mathbf{a}_2) \cdot \mathbf{P}(\mathbf{a}_3) \cdot \mathbf{P}(\mathbf{a}_4) \cdot \mathbf{P}(\mathbf{a}_5) \cdot \mathbf{P}(\mathbf{a}_6) \\ \mathbf{P}^{\text{Pankov}}(\mathbf{S}) &= \mathbf{P}(\mathbf{a}_3 \parallel \mathbf{a}_2) \cdot \mathbf{P}(\mathbf{a}_2 \parallel \mathbf{a}_1) \cdot \mathbf{P}(\mathbf{a}_1 \parallel \mathbf{a}_0) \cdot [\mathbf{P}(\mathbf{a}_5 \parallel \mathbf{a}_4) \cdot \mathbf{P}(\mathbf{a}_6 \parallel \mathbf{a}_4)] \cdot \mathbf{P}(\mathbf{a}_4 \parallel \mathbf{a}_0) \end{aligned}$$

■ **Figure 1 (A, B)** Illustrations of an exemplary structure S and the respective structure tree. PMcomp energy model is composed of the probabilities of base-pairs, shown with plain black arcs and nodes. Pankov energy model additionally incorporates in-loop probabilities of paired base-pairs that are illustrated with blue arrows. (C) Top: PMcomp structure model assume independence between the base-pair probabilities by multiplying the probabilities. Bottom: Pankov uses a loop-aware scheme to compute the probability of the structure that is efficiently calculated from pre-computed loop-conditional probabilities $P(a \parallel a')$ based on the parent-child relation of the base-pairs.

By assuming a Boltzmann distribution of the structures based on the principles of statistical mechanics, there is a bijection between energies and probabilities of structures. Thus, the probability $P(S)$ of the structure S is related to its energy $E(S)$ by

$$P(S) = \exp(-E(S)/RT)/Z = \frac{1}{Z} \prod_a \exp(-E^{\text{loop}}(a, \text{chi}(a))/RT), \quad (2)$$

based on its loops' Boltzmann weights and the partition function $Z = \sum_S \exp(-E(S)/RT)$, which is efficiently computed by McCaskill's algorithm [15].

If inverted, this allows transforming structure probabilities back to energies:

$$E(S) = -RT \cdot \log(P(S)) + E^{\text{ens}}, \quad (3)$$

where $E^{\text{ens}} = -RT \cdot \log(Z)$ denotes the ensemble energy. This notation can be further used to derive the common definition of the probability of a base pair a $P(a)$ as

$$P(a) = \sum_{S \ni a} P(S), \quad (4)$$

which can be efficiently computed by McCaskill's algorithm.

2.2 PMcomp assumes independence of base-pair probabilities

PMcomp alignment and folding score

The alignment and folding score of PMcomp [10], which is assigned to an alignment \mathcal{A} and RNA structures S_A and S_B , can be formulated as

$$\text{alignment-score}^{\text{PMcomp}}(S_A, S_B, \mathcal{A}) = \sum_{a \in S_A} \Psi_a^A + \sum_{b \in S_B} \Psi_b^B + \sum_{(i_A, i_B) \in \mathcal{A}} \sigma(i_A, i_B) + N_{\text{indel}} \gamma.$$

The first two terms define the structural component of the score that is discussed in the next section. The last two terms define the sequence component of the score. σ is the base similarity for two matched sequence positions i_A and i_B from sequence A and B , resp., and γ is the gap penalty. N_{indel} is the number of insertions and deletions in \mathcal{A} .

PMcomp structure scoring model

Here, we focus on the structure component of PMcomp's alignment and folding score, since we want to investigate how well the probabilistic model reflects the thermodynamic energy model. The score of a structure S of sequence A normalizes and sums up the log-scores of its base pair probabilities, i.e.

$$\text{score}^{\text{PMcomp}}(S) = \sum_{a \in S} \Psi_a^A = \sum_{a \in S} \log(P(a)/P_{\min}) = \log\left(\prod_{a \in S} P(a)\right) - |S \setminus \{a_0\}| \cdot \log(P_{\min}). \quad (5)$$

Here, base pair probabilities $P(a)$ are normalized via P_{\min} , the minimal probability of a significant base-pair, such that less probable base-pairs are unfavored. The logarithm transfers the probabilistic model back to the energy space similar to Eq. 3.

Putting the normalization term aside, the PMcomp's structure score contains a notion of structure probability (see first log term on the right of Eq 5), which we denote with

$$P^{\text{PMcomp}}(S) = \prod_{a \in S} P(a). \quad (6)$$

Noticeably, base pairing events are assumed to be independent, which is in violation with the underlying Nearest-Neighbor energy model, as we will show next. Thus, PMcomp's structure score does not relate well with the energy of the respective structure.

2.3 Exact computation of structure probabilities based on conditional loop probabilities

Here we prove that the equilibrium probability of a structure within the ensemble of possible structures can be expressed exactly based on conditional loop probabilities. This provides the theoretical foundation for discussing the Pankov energy model in the subsequent section.

► **Theorem 1.** *Let $P(S)$ be the probability of structure S and have $P(\text{loop}(a, \text{chi}(a)) \mid a)$ as the conditional probability of the loop in S closed by base-pair a , the following equation holds:*

$$P(S) = \prod_a P(\text{loop}(a, \text{chi}(a)) \mid a) \quad (7)$$

Proof. The free energy of the secondary structure S is composed of its loop energies E^{loop} in the nearest-neighbor thermodynamic model (Eq. 1), which implies that its probability can be computed from the respective Boltzmann weights of loops (Eq. 2). Decomposing the right term of Eq. 7 by the partition function Z_a inside the base-pairs a , i.e.

$$Z_a = \sum_{S_a \text{ closed by } a} \exp(-E(S_a)/RT), \quad (8)$$

we get:

$$\begin{aligned}
\prod_a P(\text{loop}(a, \text{chi}(a)) | a) &= \prod_a \frac{\left(\prod_{a' \in \text{chi}(a)} Z_{a'} \right) \cdot \exp(-E^{\text{loop}}(a, \text{chi}(a))/RT)}{Z_a} \\
&= \frac{\prod_a \prod_{a' \in \text{chi}(a)} Z_{a'}}{\prod_a Z_a} \cdot \prod_a \exp(-E^{\text{loop}}(a, \text{chi}(a))/RT) \\
&= *_* \frac{\prod_{a' \neq a_0} Z_{a'}}{\prod_a Z_a} \cdot \prod_a \exp(-E^{\text{loop}}(a, \text{chi}(a))/RT) \\
&= ** \frac{1}{Z} \prod_a \exp(-E^{\text{loop}}(a, \text{chi}(a))/RT) =_{(\text{Eq.2})} P(S). \tag{9}
\end{aligned}$$

(=*_): Every arc but a_0 occurs exactly once as a child in the numerator product.

(=**): a_0 encloses all possible structures, thus $Z_{a_0} = Z$. ◀

Eventually, this work extends and generalizes the approach for canonical helices (only stackings) from [1] to arbitrary secondary structures.

2.4 Pankov structure scoring model

Here, we want to score structures for simultaneous alignment and folding based on the derived Eq. 7, i.e. based on pre-computed conditional loop probabilities, to better reflect the structures' energy within the overall alignment score. Due to the exponential number of possible multi-loop branchings, a polynomial pre-computation and storing of respective multi-loop probabilities $P(\text{loop}(a, \text{chi}(a)))$ is not feasible. Thus, we propose an approximation of such terms based on pair-in-loop probabilities introduced next.

Approximate loop probabilities using pair-in-loop probabilities

To handle arbitrary multi-loops (closed by a) with any number and composition of children base pairs $\text{chi}(a)$, we restrict computation and storage to all parent-child pairs $a \times a' \in \text{chi}(a)$, i.e. we define the *pair-in-loop probability* $P(a' || a)$, in the following abbreviated as *in-loop probabilities*, as

$$P(a' || a) = P(a' \in \text{chi}(a) | a) = \frac{P(a, a' \in \text{chi}(a))}{P(a)} = \frac{\sum_{S \supset \{a, a'\} \wedge a' \in \text{chi}(a)} P(S)}{P(a)}, \tag{10}$$

where we calculate the joint probabilities of the form $P(a, a' \in \text{chi}(a))$ by an extension of McCaskill's algorithm introduced in [17], which can be performed in $O(n^3)$ time. To this end, the pair-in-loop information is stored in additional matrices during the partition function computation without increasing the computational complexity. The probability of a base-pair being external, i.e. being enclosed by the pseudo-arc a_0 , is also computed and stored. Within the following, we discuss how to approximate loop probabilities from in-loop probabilities.

Pair-in-loop approximation of non-branching-loop probabilities. When using pair-in-loop probabilities to approximate non-branching loop probabilities, i.e. loops with exactly one child base pair, the latter is overestimated since Eq. 10 does not distinguish the loop context of the pair. Therefore, also multi-loops contribute to in-loop probabilities such that it follows

$$P(a' || a) \geq P(\text{loop}(a, \{a'\} = \text{chi}(a)) | a). \tag{11}$$

Scoring based on least stable multi-loop branch. An alternative approximation would be to assign the least probable branch to the whole multi-loop. This can be intuited in the energy space as that least stable branching of the multi-loop dominates the formation of the multi-loop.

$$P_{\text{ML-min}}(\text{loop}(a, \text{chi}(a)) \mid a) = \min_{a' \in \text{chi}(a)} P(a' \mid a). \quad (12)$$

Due to the same reasons as for non-branching loops, the following relation holds, i.e.

$$P_{\text{ML-min}}(\text{loop}(a, \text{chi}(a)) \mid a) \geq P(\text{loop}(a, \text{chi}(a)) \mid a). \quad (13)$$

Assuming multi-loop-branch independence. As a straightforward approach we assume an independence between the multi-loop branches that are conditioned to be closed under the same base-pair, i.e.

$$P_{\text{ML-prod}}(\text{loop}(a, \text{chi}(a)) \mid a) = \prod_{a' \in \text{chi}(a)} P(a' \mid a). \quad (14)$$

Weighted overall structure scores

For scoring the structure in the implementation of the Pankov alignment algorithm, we assign the ultimate scoring score^{Pankov} based on the product of multi-loop contributions following Eq. 14, that we designate as the Pankov's probability of structure.

$$P^{\text{Pankov}}(S) = \prod_{a \in S} \prod_{a' \in \text{chi}(a)} P(a' \mid a) \quad (15)$$

Similar to PMcomp, the probabilities are incorporated on the logarithmic scale via the $\Phi(a', a)$ function, which also includes a normalization via the bonus term β . The latter term subsequently balances structure and sequence contributions within the alignment scoring.

$$\Phi(a', a) = \log(P(a' \mid a)) + \beta, \quad (16)$$

$$\begin{aligned} \text{score}^{\text{Pankov}}(S) &= \sum_{a \in S} \sum_{a' \in \text{chi}(a)} \Phi(a', a) \\ &= \log(P^{\text{Pankov}}(S)) + |S \setminus \{a_0\}| \cdot \beta \end{aligned} \quad (17)$$

2.5 Pankov alignment approach

The Pankov algorithm keeps track of closing-loop base-pairs in an efficient manner during dynamic programming computation of the score matrices. The matrices are defined similar to the dynamic programming recursions of the SPARSE algorithm [21], which achieve a quadratic time complexity for the alignment by exploiting the sparsity of the in-loop probabilities. Thus, Pankov uses the following matrices:

- $D(a, b)$ for the scores of matching the two base pairs $a = (a^L, a^R)$ and $b = (b^L, b^R)$ and aligning the two enclosed subsequences;
- $M^{ab}(i, k)$ for storing the maximum score of all possible alignments and foldings of the subsequences $A[a^L + 1..i]$ and $B[b^L + 1..k]$ that are under the loops enclosed by a and b ; and
- I_A and I_B for supporting variability in the helix size via deletion and insertion of base-pairs under the loops closed by a and b respectively.

The matrix entries can be calculated recursively:

$$\begin{aligned}
D(a, b) &= \max \begin{cases} M^{ab}(a^R - 1, b^R - 1) \\ I_A^{ab}(a^R - 1) \\ I_B^{ab}(b^R - 1) \end{cases} \\
M^{ab}(i, k) &= \max \begin{cases} M^{ab}(i - 1, k - 1) + \sigma(i, k) \\ M^{ab}(i, k - 1) + \gamma \\ M^{ab}(i - 1, k) + \gamma \\ \max_{\substack{P(a_1) \geq \theta, P(b_1) \geq \theta \\ a_1^R = i, b_1^R = k \\ P(a_1, a) \geq \theta', P(b_1, b) \geq \theta'}} \left(\begin{aligned} &M^{ab}(a_1^L - 1, b_1^L - 1) \\ &+ D(a_1, b_1) + \sigma(a^L, b^L) + \sigma(a^R, b^R) \\ &+ \Phi(a, a_1) + \Phi(b, b_1) \end{aligned} \right) \end{cases} \\
I_A^{ab}(i) &= \max \begin{cases} I_A^{ab}(i - 1) + \gamma \\ \max_{P(a_1) \geq \theta, P(a_1, a) \geq \theta', a_1^R = i} (a_1^L - a^L + 1) \cdot \gamma + D(a_1, b) + \Phi(a, a_1) \end{cases} \\
I_B^{ab}(k) &= \max \begin{cases} I_B^{ab}(k - 1) + \gamma \\ \max_{P(b_1) \geq \theta, P(b_1, b) \geq \theta', b_1^R = k} (b_1^L - b^L + 1) \cdot \gamma + D(a, b_1) + \Phi(b, b_1) \end{cases}
\end{aligned}$$

Here θ is the minimum base-pair probability threshold and θ' is the corresponding threshold for the joint in-loop probabilities. Following PMcomp's energy model, SPARSE calculates $\Phi(a, a_1)$ as Ψa_1 (independent of the enclosing base pair a) – in Pankov we make use of the full flexibility.

The asymptotic time complexity of the Pankov alignment algorithm is $O(n^2)$. Similar to SPARSE, the base pairing and joint in-loop probabilities need to be computed only once for each sequence. This is important for computing multiple alignments, using a pairwise aligner like Pankov in a progressive scheme (e.g. via `mlocarna` tool from the LocARNA software package). The Pankov implementation of the recursions keeps track of *both* ends of the base-pairs, while in SPARSE the tracking is relaxed and M matrices with common left-ends are combined for further speed-up, although the complexity is not affected. So in practice, compared to SPARSE, Pankov's run-time is slightly increased.

3 Results and Discussion

3.1 Evaluation of the probabilistic energy models

The evaluation procedure

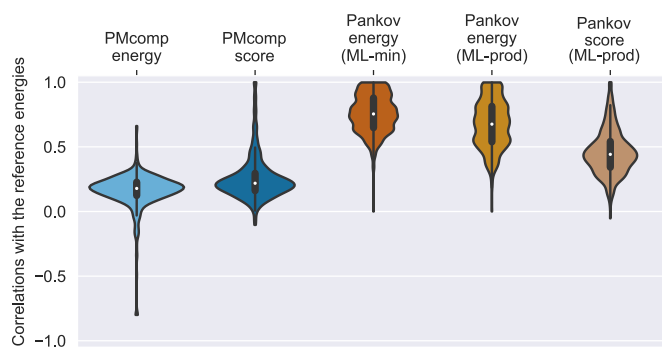
We evaluated the agreement between the reference and the probabilistic free energy models. Having the Turner [14] nearest-neighbor full energy model as the reference, we compared the performance of PMcomp's base-pair independence model and Pankov's loop-based conditional probability model. The Sankoff-like algorithms maximize (or minimize) the sum of structure and sequence alignment scores over the space of possible formations of alignments and structure. Hence, a higher correlation between the calculated energies of a model with the reference free energy values indicates better modeling of the structure score, that is expected to perform better for the task of RNA simultaneous alignment and folding.

We developed an evaluation procedure to measure the level of agreement between the probabilistic models and the reference energy model with correlation coefficients. The procedure performs these steps for an input RNA sequence: (i) suboptimal secondary

structures are generated using RNAsubopt method [24], for the range of the minimum free energy structure up to 5kcal/mol ($-e=5$ -s) and 500 suboptimal structures. (ii) for each suboptimal structure, the probability or free energy is calculated according to the described energy models and structure scores (iii) Over the set of suboptimal structures, the Spearman's rank correlation coefficient between the reference RNAsubopt's free energies and the free energies/scores of the models are computed.

The five approximation variants of the probabilistic models are evaluated which compute the probability or score of a structure. More precisely, these variants are computed according to the equations P^{PMcomp} (Eq. 6), $\text{score}^{\text{PMcomp}}$ (Eq. 5), P^{Pankov} using $P_{\text{ML-min}}$ (Eq. 12), P^{Pankov} using $P_{\text{ML-prod}}$ (Eq. 14) and $\text{score}^{\text{Pankov}}$ using $P_{\text{ML-prod}}$ and β terms (Eq. 17). The structure probabilities are transferred to the energy dimension according to Eq. 3. For consistency in the representation, the scores were also scaled with a similar scheme. As a monotonic function, the energy scale transformation does not affect the absolute rank correlations in neither of the cases.

The minimal significance probability P_{min} in Eq. 5 was set to $1/(2 \cdot \text{sequence-length})$, this is used for the PMcomp score scheme of LocARNA. The bonus balance term β from Eq. 17 was set to 1.5, a bonus of zero makes the rank correlation of $\text{score}^{\text{Pankov}}$ same as the rank correlations of P^{Pankov} using $P_{\text{ML-prod}}$. However, a non-zero value is needed to balance the total alignment score by appropriately shifting the energy. The base-pair and conditional in-loop probabilities were computed using the extended implementation of McCaskill's algorithm (see methods). The run-time complexity for calculating these values for a structure, using the precomputed matrices, are linear to the number of base-pairs in structure ($O(|S|)$) as well as the length of the sequence.

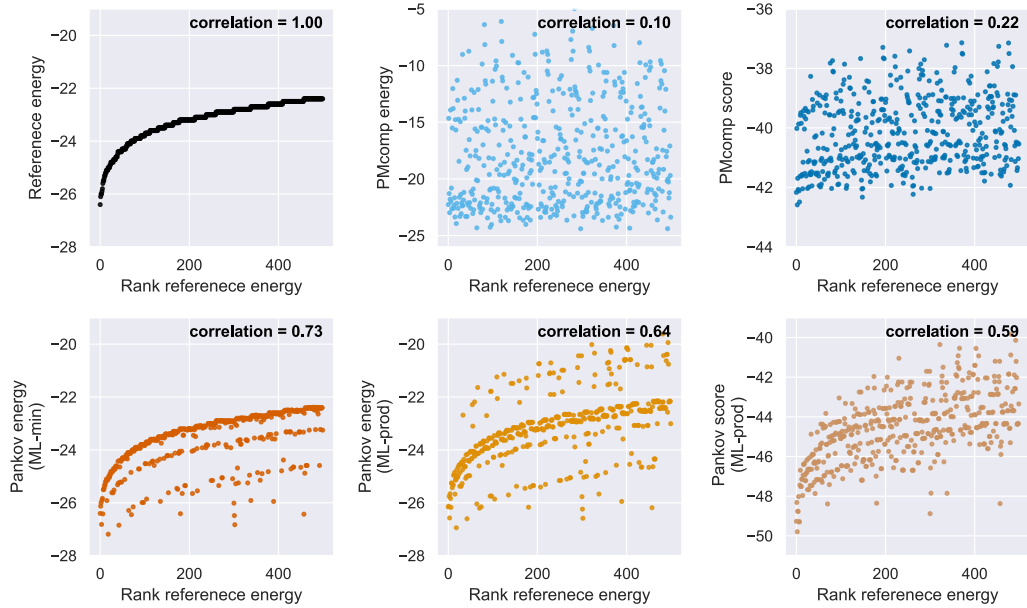


■ **Figure 2** The distribution of rank correlation for evaluating the energy models over around 500 RNA sequences from RNAstrand database. Spearman's rank correlations are computed for each sequence, between the reference and the models' approximations of the suboptimal energies.

Evaluation of real ncRNAs

The described evaluation procedure was repeated for the set of RNA sequences obtained from the RNAstrand database (sequence length [30-200] nucleotides, entries without ambiguous or spurious characters). Figure 2 shows the distribution of the rank correlation evaluation of the sequences.

To inspect the model agreements in details, we visualized the output on a sample tRNA transcript of RNAstrand (ID: SPR_00633) in Figure 3. An evaluation for the top 500 suboptimal structures of lowest free energies is shown. As can be seen in Figure 3,



■ **Figure 3** Evaluation of the agreement between the structure probabilistic and scoring models with the reference energies for a tRNA transcript. Each dot corresponds to a suboptimal structure of the tRNA. The structures are sorted according to reference free energy in the x-axis, computed by Vienna package RNAsubopt tool. The probabilities and scores are scaled to the energy dimension (Eq. 3). Annotated correlations of each panel are the Spearman’s correlation coefficients between the x and y axes. Each correlation corresponds to an individual entry within the corresponding distributions that are shown as violin plots in Figure 2.

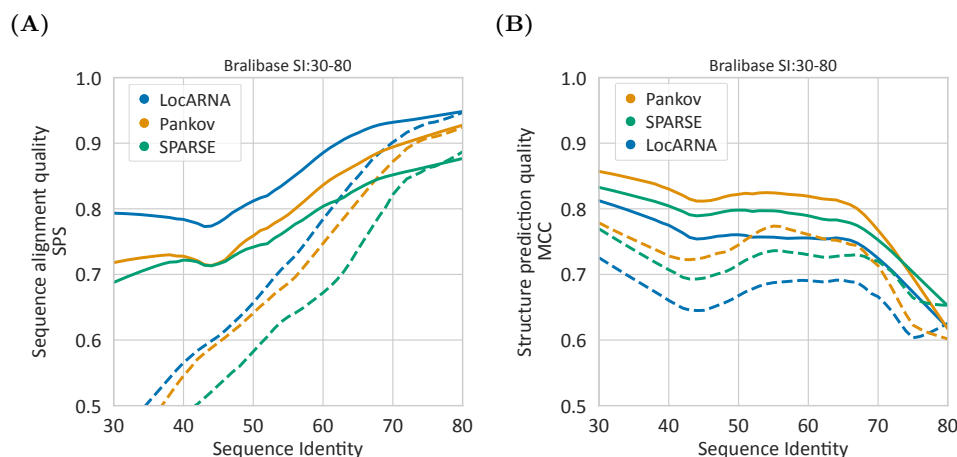
the Pankov’s in-loop-probability-based models (bottom row of Figure 3) perform best in preserving the reference free energy ranks. It is also notable that the energy scaled values are precisely scaling back to the range of reference free energies.

The scatter plot for Pankov energy (ML-min) in Figure 3 confirms the relation for the probability of multiloops that was presented in the methods section (Eq. 13), P^{Pankov} with a ML-min approximation is bounded by the exact probability of the structure such that the approximated energies are always less than or equal to the reference energies.

3.2 Alignment performance evaluation

We evaluated our implementation of Pankov alignment algorithm on the pairwise alignment benchmark set, Bralibase 2.1 [23]. For the evaluated methods, the sparsification probability thresholds were set similar. Namely, LocARNA, SPARSE and Pankov with the minimum base-pair probability θ to 0.001 (option -p). For SPARSE and Pankov the in-loop probability threshold θ' was set to 0.0001 (option -prob-basepair-in-loop). The sequence-structure balance term β of Pankov’s score (Eq. 17) was set to 1.5, the chosen among the values 1, 1.5, 2 and 2.5 posing a fair balance for the average of sequence and structure scores. The Matthews Correlation Coefficient (MCC) performance was stable for β of 1.5 and larger values.

Figure 4(A,B) show the performance comparison in term of sequence alignment quality sum-of-pairs-score (SPS) and structure prediction quality by Matthews Correlation Coefficient (MCC)[7]. To mediate the Bralibase curve “dent” effect [12], the visualization was done for sequence pairs of sequence identity (SI) between 30 to 80% to avoid a curve dent around SI-80



■ **Figure 4** Comparison of the alignment performance on the Bralibase 2.1 pairwise benchmark set K2. **(A)** The sequence-level quality is measured as sum-of-pairs-score (SPS) by comparing the alignment edges of the reference and predicted alignments. **(B)** The structure prediction quality is measured as Matthews Correlation Coefficient (MCC) by comparing the base-pairs of the reference structure with the predicted structures. The solid lines depict all families and the dashed lines depict the subset without tRNA and the two rRNA families.

that seems to be mainly caused by enforcing a continuous curve over a quasi-heterogeneous distribution. Entries with a higher SI are not of particular interest, as they mostly perform fine also using the structure-unaware alignment algorithms. Furthermore, the dashed curves in Figure 4 correspond to the subset of the benchmark by excluding the three ribosomal and tRNAs families. These families are shown to be moderately overrepresented in the Bralibase and could overweight in the overall performance, especially on lower sequence identity range [12].

In the aspect of execution time, Pankov is overall faster than LocARNA and slower than SPARSE since Pankov implements the exact loop-closing track of the alignment recursions (see methods). Our implementation of Pankov had an average run time of 1.8 seconds on Bralibase K2 instances, when running on a AMD Opteron 2.1 GHz processor; this compares to respective average run times of 3.9 and 0.7 seconds of LocARNA and SPARSE. As can be seen in Figure 4, Pankov considerably improves structure prediction over both SPARSE and LocARNA. Compared to its predecessor SPARSE, it even improves the sequence alignment quality.

4 Conclusion

Sankoff's algorithm is the reference standard for simultaneous alignment and folding (SA&F) of RNAs. While the theoretical work integrates the full loop-based nearest-neighbor energy model, the derived algorithms mostly implement a simplified or limited structure energy models or restrict on the alignment and structure formation possibilities, to reduce the high computational complexity. PMcomp proposed a probabilistic light-weight energy model. This empowers the PMcomp-like methods to strongly reduce the computational overhead of the exact thermodynamic folding details and allows further algorithmic optimizations and sparsification based on the equilibrium probability of the base-pairs.

Here we showed that PMcomp’s energy model assumes a level of independence between the base-pairing events, which violates the underlying nearest neighbor energy model. To solve this issue, we demonstrated an exact way to compute the probability of an RNA secondary structure from the decomposed loop-probabilities. To circumvent the computational complexity of multi-loop cases, we introduced an energy model to accurately approximate this loop decomposition in an efficient way using the precomputed in-loop probabilities. Our proposed energy model takes care of the nearest-neighbor thermodynamic rule. It was further empirically validated that the novel model has a much closer agreement with the full-loop energy model, based on the dataset of real non-coding RNAs. Using this energy model, we proposed the Pankov algorithm for pairwise simultaneous alignment and folding of RNAs. Benchmark results show that the implementation of Pankov outperforms its predecessors on predicting the secondary structure from the pairs of homologous RNAs.

The concept of conditional and joint in-loop probabilities has some parallels to the production rules of Stochastic Context-Free Grammars (SCFG) that can encode base-pair relations differently [5], they have also been used to solve the SA&F problem [4]. The overhead of treating various nucleotides separately during the alignment procedure is similar to the invocation of the full loop-based energy model, which restrains the implementation towards using simplified grammars that may not benefit from the power of thermodynamic rules. In our proposed model, the probabilistic terms are obtained from the thermodynamic partition function, so the probabilistic transition rules are straightforward and do not need to deal with individual types and combinations of nucleotides separately.

Pankov, to the best of authors’ knowledge, is the first SA&F method that dissociates the loop computation details from the alignment and prediction step to efficiently solve the target problem without substantially compromising the power of underlying thermodynamic rules.

References

- 1 Athanasius F Bompfinewerer, Rolf Backofen, Stephan H Bernhart, Jana Hertel, Ivo L Hofacker, Peter F Stadler, and Sebastian Will. Variations on RNA folding and alignment: lessons from Benasque. *Journal of mathematical biology*, 56(1-2):129–144, 2008.
- 2 Thomas R Cech and Joan A Steitz. The noncoding RNA revolution—trashing old rules to forge new ones. *Cell*, 157(1):77–94, 2014.
- 3 Padideh Danaee, Mason Rouches, Michelle Wiley, Dezhong Deng, Liang Huang, and David Hendrix. bpRNA: large-scale automated annotation and analysis of RNA secondary structure. *Nucleic acids research*, 46(11):5381–5394, 2018.
- 4 R. D. Dowell and S. R. Eddy. Efficient Pairwise RNA Structure Prediction and Alignment Using Sequence Alignment Constraints. *BMC Bioinformatics*, 7:400, 2006.
- 5 Robin D Dowell and Sean R Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC bioinformatics*, 5(1):71, 2004.
- 6 Paul P. Gardner, Andreas Wilm, and Stefan Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res*, 33(8):2433–9, 2005.
- 7 Jan Gorodkin, Shawn L Stricklin, and Gary D Stormo. Discovering common stem-loop motifs in unaligned RNA sequences. *Nucleic Acids Research*, 29(10):2135–2144, 2001.
- 8 Arif Ozgun Harmanci, Gaurav Sharma, and David H. Mathews. Efficient pairwise RNA structure prediction using probabilistic alignment constraints in Dynalign. *BMC Bioinformatics*, 8:130, 2007.
- 9 Jakob Hull Havgaard, Rune B Lyngsø, Gary D Stormo, and Jan Gorodkin. Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. *Bioinformatics*, 21(9):1815–1824, 2005.
- 10 I. L. Hofacker, S. H. Bernhart, and P. F. Stadler. Alignment of RNA base pairing probability matrices. *Bioinformatics*, 20(14):2222–7, 2004.

- 11 Hisanori Kiryu, Yasuo Tabei, Taishin Kin, and Kiyoshi Asai. Murlet: a practical multiple alignment tool for structural RNA sequences. *Bioinformatics*, 23(13):1588–1598, 2007.
- 12 Benedikt Löwes, Cedric Chauve, Yann Ponty, and Robert Giegerich. The BRaliBase dent-a tale of benchmark design and interpretation. *Briefings in bioinformatics*, 18(2):306–311, 2016.
- 13 David H Mathews and Douglas H Turner. Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. *Journal of molecular biology*, 317(2):191–203, 2002.
- 14 DH Mathews, J Sabina, M Zuker, and DH Turner. Expanded Sequence Dependence of Thermodynamic Parameters Improves Prediction of RNA Secondary Structure. *J Mol Biol*, 288(5):911–40, 1999.
- 15 J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–19, 1990.
- 16 Milad Miladi, Junge Alexander, Costa Fabrizio, Seemann Stefan E., Havgaard Jakob Hull, Jan Gorodkin, and Rolf Backofen. RNAscClust: clustering rna sequences using structure conservation and graph based motifs. *Bioinformatics*, 33(14):2089–2096, 2017.
- 17 Christina Otto, Mathias Mohl, Steffen Heyne, Mika Amit, Gad M. Landau, Rolf Backofen, and Sebastian Will. ExpaRNA-P: simultaneous exact pattern matching and folding of RNAs. *BMC Bioinformatics*, 15(1):6602, 2014.
- 18 David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J. Appl. Math.*, 45(5):810–825, 1985.
- 19 Matthew G Seetin and David H Mathews. RNA structure prediction: an overview of methods. In *Bacterial Regulatory RNA*, pages 99–122. Springer, 2012.
- 20 Elfar Torarinsson, Jakob H. Havgaard, and Jan Gorodkin. Multiple structural alignment and clustering of RNA sequences. *Bioinformatics*, 23(8):926–32, 2007.
- 21 Sebastian Will, Christina Otto, Milad Miladi, Mathias Möhl, and Rolf Backofen. SPARSE: quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics. *Bioinformatics*, 31(15):2489–2496, 2015.
- 22 Sebastian Will, Kristin Reiche, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. Inferring Non-Coding RNA Families and Classes by Means of Genome-Scale Structure-Based Clustering. *PLoS Comput Biol*, 3(4):e65, 2007.
- 23 Andreas Wilm, Indra Mainz, and Gerhard Steger. An enhanced RNA alignment benchmark for sequence alignment programs. *Algorithms Mol Biol*, 1:19, 2006.
- 24 Stefan Wuchty, Walter Fontana, Ivo L Hofacker, and Peter Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers: Original Research on Biomolecules*, 49(2):145–165, 1999.

Context-Aware Seeds for Read Mapping

Hongyi Xin

Computer Science Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA

Mingfu Shao

Department of Computer Science and Engineering,
The Pennsylvania State University, University Park, PA, USA

Carl Kingsford¹

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Motivation: Most modern seed-and-extend NGS read mappers employ a seeding scheme that requires extracting t non-overlapping seeds in each read in order to find all valid mappings under an edit distance threshold of t . As t grows (such as in long reads with high error rate), this seeding scheme forces mappers to use more and shorter seeds, which increases the seed hits (seed frequencies) and therefore reduces the efficiency of mappers.

Results: We propose a novel seeding framework, context-aware seeds (CAS). CAS guarantees finding all valid mapping but uses fewer (and longer) seeds, which reduces seed frequencies and increases efficiency of mappers. CAS achieves this improvement by attaching a confidence radius to each seed in the reference. We prove that all valid mappings can be found if the sum of confidence radii of seeds are greater than t . CAS generalizes the existing pigeonhole-principle-based seeding scheme in which this confidence radius is implicitly always 1. Moreover, we design an efficient algorithm that constructs the confidence radius database in linear time. We experiment CAS with *E. coli* genome and show that CAS reduces seed frequencies by up to 20.3% when compared with the state-of-the-art pigeonhole-principle-based seeding algorithm, the Optimal Seed Solver.

Availability: https://github.com/Kingsford-Group/CAS_code

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Read Mapping, Seed and Extend, Edit Distance, Suffix Trie

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.15

Supplement Material https://github.com/Kingsford-Group/CAS_code

Funding This research is funded in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through grant GBMF4554 to CK, by the U.S. National Science Foundation (CCF-1319998) and by the U.S. National Institutes of Health (R01GM122935). This work was partially funded by the Shurl and Kay Curci Foundation. This project is funded, in part, by a grant (4100070287) from the Pennsylvania Department of Health. The department specifically disclaims responsibility for any analyses, interpretations, or conclusions.

1 Introduction

Read mapping is used ubiquitously in bioinformatics. Commonly, it is defined as follows:

► **Problem 1 (Read Mapping).** *Given read R and reference T (usually with $|T| \gg |R|$), an edit distance measurement $D(\cdot, \cdot)$, and an error tolerance threshold t , we say a substring of T at location $[l_1, l_2]$, i.e., $T[l_1, l_2]$, is a valid mapping of R if we have $D(R, T[l_1, l_2]) < t$.*

¹ corresponding author, email: carlk@cs.cmu.edu



To efficiently map reads, modern mappers usually employ the *seed-and-extend* mapping strategy [8, 9, 1, 14]: a mapper extracts a substring of R as a *seed*, s ; iterates through all seed locations of s in T ; at each seed location, performs sequence alignment of R against the surrounding text in T ; reports alignments that have edit distances below t as valid mappings.

For mappers that use non-overlapping seeds, the number of seeds to extract from a read R is governed by the pigeonhole principle: to find all valid mappings of R , the mapper must divide R into at least t non-overlapping seeds. Otherwise, the mapper will not be able to consistently find all valid mappings of R in T . As t grows, the length of seeds is reduced. Using short seeds significantly increases the workload of a mapper [6, 11]. Shorter seeds appear more frequently in T , hence increasing the number of alignments while mapping a read. To improve the performance of mappers, it is desirable to use fewer non-overlapping seeds under a fixed t , which lets a mapper not only use fewer seeds, but also use longer seeds.

In this paper, we focus on improving seed-and-extend mappers that use non-overlapping seeds. We propose a novel seeding scheme, called *context-aware* seeds (CAS). CAS enables a mapper to use fewer than t seeds without missing any valid mappings. CAS attaches each seed s with a *confidence radius* score, c_s , with $c_s \geq 1$. Let S be a set of non-overlapping seeds from R . CAS ensures that as long as $\sum_{s \in S} c_s \geq t$, then S is sufficient to find all valid mappings of R under an error tolerance threshold of t . When S includes any seed s with $c_s > 1$, then $|S| < t$ and all valid mappings are secured with fewer-than- t seeds ($|S|$ denotes the number of seeds in S). In the worst case where $c_s = 1$ for all $s \in S$, CAS degenerates into the case governed by pigeonhole principle with $|S| = t$.

Figure 1 compares CAS and the pigeonhole-principle-based seeds. Assume that we have verified that the two CAS seeds AACCTGG and TTGG have confidence radii of $c_s = 2$. Therefore CAS can be guaranteed to find all valid mappings with just these two seeds, as $\sum_s c_s = 4 \geq t$. Using the pigeonhole principle, however, a mapper needs to select $t = 4$ non-overlapping seeds. It forces the mapper to pick short and repetitive seeds, making the mapper perform more local alignments.

read	AACCTGG
reference	GGAATTAAGGAACCGTTGGTTAATTCCGG
ordinary seeds	AACCTGG
reference	GGAATTAAGGAACCGTTGGTTAATTCCGG
context-aware seeds	AACCTGG
reference	GGAATTAAGGAACCGTTGGTTAATTCCGG

■ **Figure 1** Illustration of CAS. The upper part shows a read and a reference. Suppose that $t = 4$, i.e., we want to find all alignments of the read in the reference with fewer than 4 edits. There is only one such locally optimal alignment (marked as red). The middle part shows the seed extraction result with the pigeonhole principle, which splits the read into $t = 4$ seeds. This gives many seed locations and thus many alignments. With CAS (in the lower part), we can split the read into 2 long seeds while still guarantee to find all valid mappings. The two long seeds together have a total seed frequency of 2, drastically reducing the number of alignments.

We establish the theoretical foundation of CAS and demonstrate that with CAS future mappers can map reads more efficiently using fewer, longer and less frequent seeds without losing valid mappings. We also propose a suffix-trie-based CAS database construction algorithm that builds a CAS database from T in linear time, based on which we design a greedy CAS seeding algorithm that extracts CAS from reads. We test the greedy CAS seeding algorithm against a state-of-the-art pigeonhole-principle-based seeding algorithm, Optimal Seed Solver (OSS), on an *E. coli* dataset.

2 Context-Aware Seeds

CAS reduces seed usage in read mapping by introducing a novel metric for seeds in T , the confidence radius. A seed s in T has a confidence radius c_s if c_s is a smallest value (a lower bound), such that all substrings in T whose edit distance is smaller than c_s must occur in T within a small window where s occurs. The window equals to extending s by $c_s - 1$ letter(s) at both ends. For example, under $t = 2$, seed **AACC** in T from Figure 1 has a confidence radius of 2. Any substring in T whose edit distance to **AACC** equals 1 (e.g., **AAC**, **ACC**, **GAACC**, **AACCG**) locates within the 1-letter extended window of **AACC** (**GAACCG**). The confidence radius of each possible seed in T can be computed by profiling T . CAS guarantees that all valid mappings of a read R can be located, as long as the seeds s extracted from R collectively have a confidence radius of $\sum_s c_s > t$. Below, we give the formal definition of CAS and prove the correctness of CAS.

Let s be a string in T and $[l_1, l_2]$ be a pair of locations. We say string $T[l_1, l_2]$ is in the *vicinity* of s under an integer c , if $\exists [l_{s1}, l_{s2}]$, where $l_1 - c < l_{s1} < l_{s2} < l_2 + c$ and $T[l_{s1}, l_{s2}] = s$. Furthermore, let seed s be a substring of R at $[l_{r1}, l_{r2}]$ ($s = R[l_{r1}, l_{r2}]$) and let $T[l_1, l_2]$ be a valid mapping of R . We say $T[l_1, l_2]$ is in the *vicinity of s with regard to R* under c , if string $T[l_1 + l_{r1}, l_1 + l_{r2}]$ is in the vicinity of s under c . If a valid mapping $T[l_1, l_2]$ is in the vicinity of s with respect to R under t , then $T[l_1, l_2]$ can be discovered by locally aligning R against the surrounding text in T at each seed location of s .

The pigeonhole principle states that by dividing R into a set of t non-overlapping seeds, denoted by S , then $\forall [l_1, l_2]$ where $T[l_1, l_2]$ is a valid mapping of R , there must be $s \in S$ where $T[l_1, l_2]$ is in the vicinity of s with regard to R .

CAS seeks to retain the seed vicinity guarantee of the pigeonhole principle, where all valid mappings of a read R are in the vicinity of its seeds with regard to R under t , with **fewer than t seeds**. Given two substrings s and s' of T and a edit-distance threshold t , we say s' is a neighbor of s if $D(s, s') < t$. Assume that s' is a neighbor of s under t , CAS defines s' as a *trivial neighbor* of s , if and only if $\forall [l_1, l_2]$ where $T[l_1, l_2] = s'$, $T[l_1, l_2]$ is in the vicinity of s under $D(s, s')$. Otherwise CAS defines s' as a *nontrivial neighbor* of s . Finally, CAS defines *the confidence radius* c_s of s as the minimum of 1) t and 2) the minimum edit-distance between s and all nontrivial neighbors of s . Since a seed is trivial to itself and is at least 1-edit-distance away from any other string, we have $t \geq c_s \geq 1$ for any seed s .

We now give the central theorem of CAS, the theoretical foundation that enables seed-and-extend mappers to find all valid mappings using fewer than t seeds.

► **Theorem 1.** *Let S be a set of non-overlapping seeds of a read R , if $\sum_{s \in S} c_s \geq t$, then $\forall [l_1, l_2]$ where $D(R, T[l_1, l_2]) < t$, $\exists s \in S$ where $T[l_1, l_2]$ is in the vicinity of s with regard to R under t .*

Proof. Assume that $T[l'_1, l'_2]$ is a valid mapping of R , where $D(R, T[l'_1, l'_2]) < t$. Further assume that $T[l'_1, l'_2]$ is not in the vicinity, with regard to R under t , of any $s \in S$. In the minimum-edit-distance alignment between R and $T[l'_1, l'_2]$, assume that the non-overlapping seeds s_1, s_2, \dots, s_n of R are aligned to the non-overlapping segments $s_{T1}, s_{T2}, \dots, s_{Tn}$ of $T[l'_1, l'_2]$, with $n = |S|$. Since $T[l'_1, l'_2]$ is not in the vicinity, with regard to R under t , of any $s \in S$; and also because $c_{s_i} \leq t$ for all i ; there does not exist i where s_{Ti} is in the vicinity of s_i , under c_{s_i} . Therefore, s_{Ti} is a nontrivial neighbor of s_i for all $i \in [1, n]$. Because c_{s_i} is the minimum edit-distance between s_i and any of its nontrivial neighbors, we have $D(R, T[l'_1, l'_2]) \geq \sum_i D(s_i, s_{Ti}) \geq \sum_s c_s \geq t$. $D(R, T[l'_1, l'_2]) \geq t$ contradicts the assumption that $T[l'_1, l'_2]$ is a valid mapping of R . Therefore such $T[l'_1, l'_2]$ does not exist. ◀

3 Construction of Confidence Radius Database

The confidence radius c_s of each seed s is stored in a table, called *the confidence radius database*. The confidence radius database only needs to be constructed once offline for a reference T .

Computing c_s of seed s involves finding the minimum edit distance to its nontrivial neighbors. Below we propose an algorithm that constructs the confidence radius database in $O(|\Sigma|^2 \cdot M)$ time, where Σ is the alphabet set of T and M is the total number of neighbors of all strings in T (up to length P and under the edit distance threshold t).

The confidence radius database is constructed in two steps: first, we construct a neighbor database, which stores all neighbors of all seeds (up to length P) under the edit distance threshold t ; then, we find the confidence radius of each from its neighbors. We prove that both steps can be done in $O(|\Sigma|^2 \cdot M)$ time.

3.1 Construction of the Neighbor Database

To find all neighbors of all substrings in T (up to a maximum length P), we first build a P -level suffix trie of T , then find all neighbors of each seed in $Trie$ by systematically traversing the suffix trie in a top-down manner. Formally, let $Trie = (V, E)$ be a suffix trie of T of a maximum depth of $P + t$. Let $r \in V$ be the root of $Trie$. Each node represents a substring in T , i.e., the string obtained by concatenating the letters on edges along the path from r to v . We denote the edit distance between these two substrings corresponding to u and v as $D(u, v)$. We aim to solve the following problem:

► **Problem 2.** *Given a suffix trie $Trie = (V, E)$ and an integer t , to compute all pairs of nodes $u, v \in V$ such that $D(u, v) \leq t$.*

For any $v \in V$, $p(u)$ denotes the parent node of v in $Trie$. $\sigma(p(v), v)$ denotes the letter on the edge between v and $p(v)$, i.e., $(p(v), v) \in E$. We have the following lemmas.

► **Lemma 2.** *Let $u, v \in V$. Then $D(u, v) \leq t$ only if $D(p(u), p(v)) \leq t$.*

Proof. Proved in Landau and Vishkin [7] by enumerating and validating all possible scenarios. ◀

► **Lemma 3.** *Let $u, v \in V$. We have*

$$D(u, v) = \min \begin{cases} D(p(u), p(v)) + \delta_{uv} \\ D(p(u), v) + 1 \\ D(u, p(v)) + 1 \end{cases}$$

where $\delta_{uv} = 1$ if $\sigma(p(u), u) \neq \sigma(p(v), v)$ and $\delta_{uv} = 0$ if $\sigma(p(u), u) = \sigma(p(v), v)$.

Proof. This follows the dynamic programming algorithm for the edit distance problem. ◀

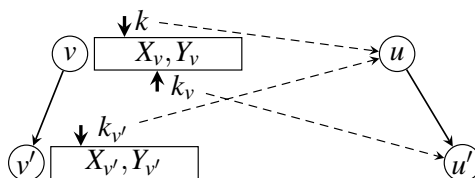
Lemma 2 shows that nodes are neighbors only if their parents are neighbors. Hence the neighbors of a child node must be the children of the neighbors of its parent node. Lemma 3 further shows that the edit distance between two children nodes can be computed in constant time, given the edit distances between one child and the parent node of the other child, as well as the edit distance between the two parent nodes.

We construct the neighbor database by traversing $Trie$ as follows: First, assign each node in V an integral *rank* from $\{1, 2, \dots, |V|\}$ following a top-down, left-to-right order. The root r of $Trie$ has rank of 1, and then the children of children of r have ranks of 2, 3, \dots , from the

leftmost child to the rightmost child, and so on. Nodes that are deeper in *Trie* rank higher. Among nodes of the same depth, children of a higher ranking parent node rank higher. A breadth-first-search traversal of *Trie* ranks all nodes.

For any $v \in V$, we define $X_v := \{u \in V \mid D(u, v) \leq t\}$ as the set of neighbors of v , including v itself, and define $Y_v := \{D(u, v) \mid u \in X_v\}$ as the accompanying edit-distance set of X_v . For every neighbor node u in X_v , Y_v provides the edit distance between u and v . We compute X_v and Y_v for each node $v \in V$ from low ranking nodes to high ranking nodes. Both X_v and Y_v are implemented as arrays.

The algorithm for constructing the neighbor database is summarized in Algorithm 1. We iterate through all nodes by rank from low to high. For each node $v \in V$, we iterate through all children of v . For each children node v' of v , we compute $X_{v'}$ and $Y_{v'}$ of v' based on the previously computed X_v and Y_v of v . Figure 2 illustrates the process of validating a candidate neighbor u' of another node v' , based on the information of its parent node v and the neighbor u of v , where u is also the parent of u' (lines 4–17 in Algorithm 1). We prove that this algorithm maintains the following three invariants:



■ **Figure 2** Illustration of processing a single node v (i.e., lines 4–17 of Algorithm 1).

■ **Algorithm 1** Linear Time Algorithm for Problem 2.

Input: Suffix trie $Trie = (V, E)$ and the edit-distance threshold t

Output: X_v and Y_v for each $v \in V$

0. Initialize X_r and Y_r for root $r \in V$.
1. **FOR** each node $v \in V$ in ascending order:
 2. Initialize pointer $k_v = 0$ for arrays X_v and Y_v .
 3. Initialize arrays $X_{v'}$ and $Y_{v'}$ for each child v' of v as empty arrays.
 4. Initialize pointer $k_{v'} = -1$ for arrays $X_{v'}$ and $Y_{v'}$ for each child v' of v .
 5. **FOR** $k = 0 \rightarrow |X_v|$:
 6. **LET** $u := X_v[k]$.
 7. **FOR** each child u' of u :
 8. **FOR** each child v' of v :
 9. **LET** $\delta = 1$ if $\sigma(u, u') \neq \sigma(v, v')$ and $\delta = 0$ if $\sigma(u, u') = \sigma(v, v')$. Compute $D_1 = Y_v[k] + \delta$, i.e., $D_1 = D(v, u) + \delta$.
 10. Increase k_v until $X_v[k_v] \geq u'$. **IF** we have $X_v[k_v] = u'$, i.e., $u' \in X_v$, **THEN** compute $D_2 = Y_v[k_v] + 1$, i.e., $D_2 = D(v, u') + 1$; otherwise set $D_2 = \infty$.
 11. Increase $k_{v'}$ until $X_{v'}[k_{v'}] \geq u$. **IF** we have $X_{v'}[k_{v'}] = u$, i.e., $u \in X_{v'}$, **THEN** compute $D_3 = Y_{v'}[k_{v'}] + 1$, i.e., $D_3 = D(v', u) + 1$; otherwise set $D_3 = \infty$.
 12. Compute $D(v', u') = \min\{D_1, D_2, D_3\}$. **IF** $D(v', u') < t$, **THEN** add u' to the end of $X_{v'}$ and add $D(u', v')$ to the end of $Y_{v'}$.
 13. **END FOR**
 14. **END FOR**
 15. **END FOR**
 16. **END FOR**

1. For any node $v \in V$, array X_v is always sorted according to their ranks, i.e., nodes that are added to X_v are always in ascending order *w.r.t.* their ranks.
2. Right before processing node v (i.e., before line 4 of Algorithm 1), X_v and Y_v are already computed and sorted *w.r.t.* their ranks.
3. Right after processing node v (i.e., after line 17 of Algorithm 1), $X_{v'}$ and $Y_{v'}$ are computed and sorted *w.r.t.* their ranks for each child v' of v .

The initialization step Algorithm 1 (line 2) computes X_r and Y_r for root node r . Its neighbors include all nodes whose depth in *Trie* is no greater than t . The edit distance between r to a neighbor node u is simply the depth of u minus 1 (we assume that root r is at depth 1). Root r is also in X_r with $D(r, r) = 0$. Clearly, the first and the second invariant hold for root r .

In the main loop (lines 3–18), for a node $v \in V$, Algorithm 1 iterates through all of its children. For a child v' of v , lines 4–17 compute $X_{v'}$ and $Y_{v'}$ of v' . Line 4–6 initialize the pointers that will be used to fetch the edit distances $D(v, u')$ and $D(v', u)$, which are stored in Y_v and the partially computed $Y_{v'}$, respectively. Because u ranks higher than u' , by the time of computing $D(v', u')$, $D(v', u)$ is already computed and stored in $X_{v'}$. $D(v', u')$ is then computed according to Lemma 3. Specifically, pointer k tracks the position of u in array X_v (i.e., the index of u in array X_v); pointer k_v tracks the position of u' in array X_v ; and pointer $k_{v'}$ tracks the position of u in array $X_{v'}$. Line 11 computes $D_1 := D(v, u) + \delta$, in which $D(v, u)$ is fetched from Y_v indexed by k . Line 12 computes $D_2 := D(v, u') + 1$, in which $D(v, u')$ is fetched from Y_v indexed by k_v . Line 13 computes $D_3 := D(v', u) + 1$, in which $D(v', u)$ is fetched from $Y_{v'}$ indexed by $k_{v'}$. Line 14 computes $D(v', u') := \min\{D_1, D_2, D_3\}$; adds u' to $X_{v'}$ and adds $D(v', u')$ to $Y_{v'}$ if $D(v', u') < t$.

Algorithm 1 maintains the first invariant. For each child v' of v , assuming X_v is sorted, then neighbors are also added to $X_{v'}$ in a sorted manner, as Algorithm 1 iterates through neighbors ordered by X_v . Since X_r is sorted for root r , given the inductive nature of Algorithm 1, we conclude that X_v must be sorted for any $v \in \text{Trie}$.

Algorithm 1 maintains the third invariant. According to Lemma 2, a node $u' \in X_{v'}$ requires $u \in X_v$ for their parents u and v . Any node \bar{u}' whose parent $\bar{u} \notin X_v$ results in $\bar{u}' \notin X_{v'}$. Algorithm 1 iterates through all u in X_v . Therefore, after line 17, all neighbors of child v' must have been found, assuming the second invariant holds. The second invariant holds because all neighbors of r are correctly defined during initialization. As the algorithm propagates, because of the inductive nature of Algorithm 1, the second invariant holds.

Let M denote the member size of set $\{(u, v) \mid D(u, v) \leq t\}$. The complexity of Algorithm 1 is $O(|\Sigma|^2 \cdot M)$.

► **Theorem 4.** *Algorithm 1 computes X_v and Y_v for each $v \in V$ in $O(|\Sigma|^2 \cdot M)$ time.*

Proof. For each $v \in V$, lines 4–17 compute $X_{v'}$ and $Y_{v'}$ for each child v' of v in $O(|X_v| \cdot |\Sigma|^2 + \sum_{v': p(v')=v} |X_{v'}|)$ time. Since pointers of k_v and $k_{v'}$ can only move forward, lines 12–13 cost $|X_v| + \sum_{v': p(v')=v} |X_{v'}|$ operations. Operations in lines 11–14 cost constant time. Hence, lines 7–17 cost $O(|X_v| \cdot |\Sigma|^2)$ operations, as the number of children of each node is bounded by $|\Sigma|$. The overall run time of Algorithm 1 is thus bounded by $\sum_{v \in V} O(|X_v| \cdot |\Sigma|^2 + \sum_{v': p(v')=v} |X_{v'}|) = O(|\Sigma|^2 \cdot M)$. ◀

With $|\Sigma|$ being a small constant (for example $\Sigma = \{A, C, G, T\}$ for DNA analysis), Algorithm 1 finds all M neighbor pairs in *Trie* in $O(M)$ time.

3.2 Computing the Confidence Radius Among Nontrivial Neighbors

The neighbor database stores both the trivial and nontrivial neighbors of each seed. However, CAS only requires the minimum edit distance to the nontrivial neighbors of each seed. In order to derive the confidence radius of each seed, we propose an augmentation to Algorithm 1, such that it computes the minimum edit distance to nontrivial neighbors while constructing the neighbor database. We prove that the augmentation does not increase the time complexity of Algorithm 1.

Within the neighbor array X_v of a node v , let the sub-array X_v^0 store all trivial neighbors and X_v^1 store all nontrivial neighbors, where $X_v = X_v^0 \cup X_v^1$. By definition, the confidence radius of v is computed as $c_v := \min_{u \in X_v^1} D(u, v)$. To compute c_v , instead of finding all nontrivial neighbors, X_v^1 , we compute a subset $X_v^2 \subset X_v^1$, where $\min_{u \in X_v^2} D(u, v) = \min_{u \in X_v^1} D(u, v)$.

Let u be a neighbor of v ; we say u is an *immediate* neighbor of v if u is a substring, or a superstring, or an overlapping string of v ; otherwise we say u is a *non-immediate* neighbor of v (see Figure 3 for examples). Immediate neighbors are not necessarily trivial neighbors. If u is a trivial neighbor of v , by definition, then u must be an immediate neighbor of v . However, the opposite is not necessarily true, i.e., u could be a substring of v (an immediate neighbor) yet u is nontrivial to v . Substring u may appear at more locations in T than v does. It is easier to determine whether u is an immediate neighbor to v than whether u is a trivial neighbor to v .

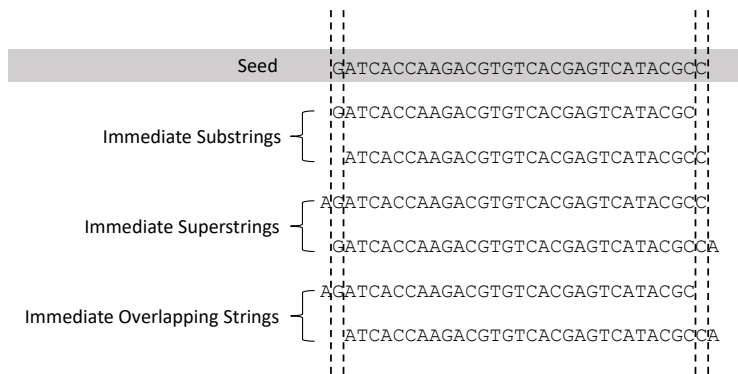


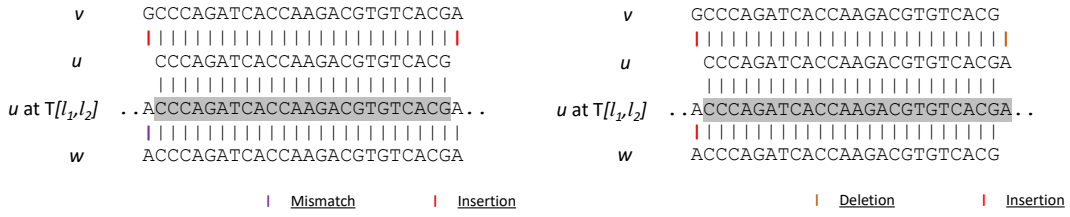
Figure 3 Examples of trivial neighbors of a seed, including substrings, superstrings, and overlapping strings of this seed.

Let X_v^2 be the set of non-immediate neighbors of a node v . The minimum edit distance from v to nontrivial neighbors of v equals to the minimum edit distance between v to neighbors in X_v^2 . We prove this in Theorem 7. To prove Theorem 7, we first prepare the following two lemmas.

► **Lemma 5.** *If u is a superstring of v , then u is a trivial neighbor of v .*

Proof. Since u is a superstring of v , for any location $[l_1, l_2]$ of u , $\exists [l_1, l_2]$ where $T[l_1, l_2] = v$ and $l_1 - D(u, v) \leq l_1 < l_2 \leq l_2 + D(u, v)$. By definition, u is a trivial neighbor of v . ◀

► **Lemma 6.** *If u is a substring or an overlapping string of v and u is a nontrivial neighbor of v , then $\exists w \in Trie$, where w is neither an immediate neighbor nor a trivial neighbor of v , with $|w| = |v|$ and $D(v, w) \leq D(v, u)$.*



■ **Figure 4** Illustration of Lemma 6. The figure to the left shows an example where u is a substring of v , while the figure to the right shows an example where u is an overlapping string of v . Notice that in both figures, w is always optimally aligned to v .

Proof. Since u is a nontrivial neighbor of v , $\exists [l_1, l_2]$, where $T[l_1, l_2] = u$ but $T[l_1, l_2]$ is not in the $D(v, u)$ -edit vicinity of v . We extract a substring w within $T[l_1 - D(u, v), l_2 + D(u, v)]$, where w locally and optimally aligns to v in $T[l_1 - D(u, v), l_2 + D(u, v)]$, with $|w| = |v|$, as shown in Figure 4. Then w must be a nontrivial neighbor of v since $T[l_1, l_2]$ is not in the $D(u, v)$ -edit vicinity of v . Because w is optimally aligned to v within $[l_1 - D(u, v), l_2 + D(u, v)]$, we have $D(w, v) \leq D(u, v)$. ◀

By combining Lemmas 5 and 6 we prove the following theorem.

► **Theorem 7.** $c_v = \min_{u \in X_v^2} D(u, v)$, where X_v^2 is the set of non-immediate neighbors of v .

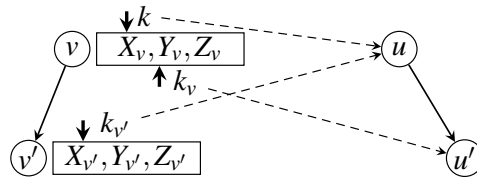
Proof. Lemmas 5 and 6 state that for any nontrivial immediate neighbor u of seed v , there must exist a nontrivial and non-immediate neighbor w of v where $D(w, v) \leq D(u, v)$. Therefore, by definition, we have $c_v = \min_{u \in X_v^2} D(u, v)$. ◀

We find the immediate neighbors, X_v^3 , of each node $v \in Trie$, by checking if a neighbor $u \in X_v$ is a immediate substring, superstring or overlapping string of v . We associate with each node v a new vector $Z_v := \{F(v, u) \mid u \in X_v\}$, where $F(v, u)$ stores the information of whether $u \in X_v^3$. With $X_v^2 = X_v \setminus X_v^3$, the updated workflow is illustrated in Figure 5.

Computation of $F(v, u)$ can be piggybacked on top of computing $D(v, u)$ in Algorithm 1. Given u and v , $F(v, u)$ stores whether v and u possess any of the below *immediate conditions*: (1) v is a prefix of u . (2) v is a suffix of u . (3) u is a prefix of v . (4) u is a suffix of v . (5) v is neither a prefix nor a suffix but a substring of u . (6) u is neither a prefix nor a suffix but a substring of v . (7) A prefix of v is a suffix of u . (8) A suffix of v is a prefix of u .

From above immediate conditions, we deduce the immediate relationship between v and u . With conditions 1–6, we can infer the superstring-substring relationship. With Condition 7–8, we can infer the overlapping relationship. If v and u qualifies none of the above immediate conditions, then they must be non-immediate neighbors.

For simplicity, we initialize each node as satisfying immediate conditions 1, 2, 3 and 4 to itself. We initialize the root node r as a prefix to any of its neighbors; and any neighbors of r as a suffix to r . Finally, r is not an overlapping string or a substring of any neighbor.



■ **Figure 5** Illustration of adding $Z_v := \{F(u, v) \mid u \in X_v\}$ to each node.

$F(u, v)$ can be computed in constant time if $F(p(v), p(u))$, $F(p(v), u)$ and $F(v, p(u))$ are known. For example, in Figure 5, $F(v', u')$ satisfies condition 1, only if (a) $v' = u'$ or (b) $F(v', u)$ satisfies condition 1. $F(v', u')$ satisfies condition 2, only if (a) $u' = v'$ or (b) $F(v, u)$ satisfies condition 2 and $\sigma(u, u') = \sigma(v, v')$. Conditions 3 and 4 are mirror cases of conditions 1 and 2, respectively with v and u , v' and u' trading places. $F(v', u')$ satisfies condition 5, only if (a) $F(v', u)$ satisfies condition 5 or (b) $F(v', u)$ satisfies condition 2, while $v' \neq u'$ and v is not root. Condition 6 is a mirror case of condition 5. $F(v', u')$ satisfies condition 7 only if (a) $F(v, u')$ satisfies condition 7 or (b) $F(v, u')$ satisfies condition 2, while $v \neq u'$ and v is not root. Condition 8 is a mirror case of condition 7.

The computation of $F(\cdot, \cdot)$ is piggybacked on top of the computation of $D(\cdot, \cdot)$, as both methods use dynamic programming. Both methods require prior knowledge between the child-parent and parent-parent node pairs; and from prior results, both methods compute the new result of the child-child node pair in constant time. As a result, piggybacking the computation of immediateness does not increase the complexity of Algorithm 1.

Finally, the confidence radius of node v equals $\min D(v, u)$ where $u \in X_v^2$, where $F(v, u)$ does not satisfy any of the immediate conditions. The confidence radius of a node can be found by simply scanning its neighbor array, which finishes in linear time. The overall complexity of constructing the confidence radius database is still $O(|\Sigma|^2 \cdot M)$.

The confidence radius database is stored in a $|T|$ -by- P table, where P is user-provided. The $[x, y]$ entry of the table stores the c_s of seed $T[x, x + y]$. In practice, $|T| \gg P$ and when needed, we can condense the confidence radius database into bit-vectors to reduce the table size. If necessary, when $|T|$ is large, we can sub-sample seeds only at fixed-length intervals to further reduce the storage footprint.

4 A Seeding Scheme with Context-Aware Seeds

While the major goal of this paper is to establish the theoretical framework of CAS, to test the effectiveness of CAS, we propose a greedy seed selection method, referred to as greedy CAS seeding. Greedy CAS seeding selects consecutive Maximum Exact Matching substrings (MEMs, which are seeds that cannot be further extended without bumping into errors) from a read as seeds. At the end of each MEM, greedy CAS seeding heuristically skips the next two base pairs, in an effort to skip potential errors. Greedy CAS seeding sorts seeds by their frequency from low to high, into S_{raw} . Then selects the minimum number of seeds S from S_{raw} in sequential order such that $\sum_{s \in S} c_s \geq t$. In the rare cases where there is insufficient number of CAS seeds such that $\nexists S$ with $\sum_{s \in S} c_s \geq t$, greedy CAS seeding reverts back to using the pigeonhole principle, by dividing the read into t non-overlapping seeds.

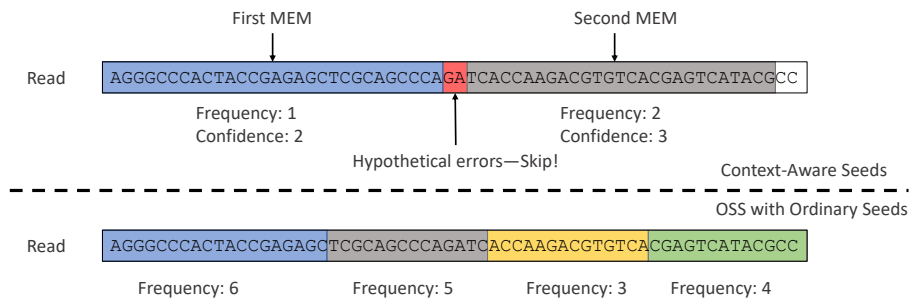


Figure 6 An example of drawing context-aware seeds from a read.

Figure 6 compares the seed extraction results of greedy CAS seeding against the state-of-the-art, pigeonhole-principle-based seeding method, the Optimal Seed Solver (OSS) [15]. OSS has been previously shown that it generates the least frequent seeds, when compared to other pigeonhole-principle-based seeding methods, such as flexible-placement k-mers or spaced seeds. Figure 6 demonstrates both seeding methods in action under $t = 4$. Greedy CAS seeding is shown in the upper half while OSS is shown in the lower half. Compared to OSS, which uses a total of $t = 4$ seeds, greedy CAS seeding uses only two seeds. As a result, greedy CAS seeding can afford longer and less frequent seeds.

Greedy CAS seeding has a maximum complexity of $O(|R| + |S| \log(|S|))$ ($|R|$ denotes the length of R while $|S|$ denotes the cardinality of set S). We use Burrows-Wheeler Transformation (BWT) array to index seeds. With BWT array, it takes $O(|s|)$ operations to access the seed database for seed s and locate all seed locations of s . Given that $\sum_{s \in S} |s| \leq |R|$, and $|S| \leq t \ll |R|$, we conclude that the maximum complexity of greedy CAS seeding is $O(|R| + t \log(t))$.

5 Experiments

We benchmark greedy CAS seeding against OSS on the *E. coli* genome. We benchmark both seeding schemes on a 22-million, 100-bp *E. coli* read set from EMBL-EBI, ERX008638-1. We build a confidence radius database for *E. coli* genome with a maximum edit distance threshold $t = 5$ and a max seed length $P = 60$. We measure the effectiveness of both approaches by comparing the average total seed frequency of selected seeds under different edit distance thresholds $t = \{1, 2, 3, 4, 5\}$. The average total frequency is the sum of seed frequencies extracted from each read, averaged over all reads in the read set.

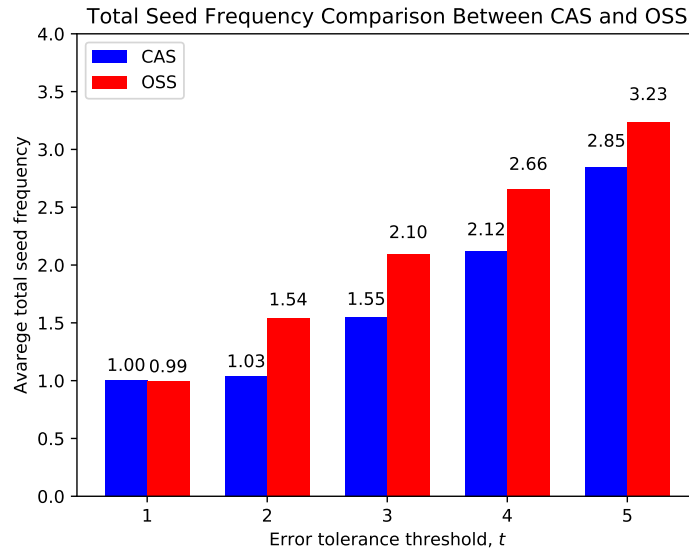
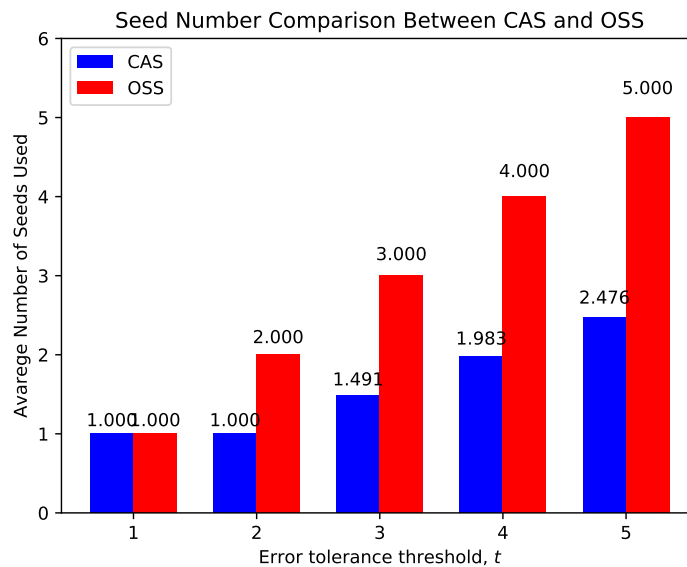


Figure 7 Comparison between CAS and OSS in terms of total seed frequency, with various edit distance thresholds t .

Figure 7 shows the average total seed frequency comparison between the two approaches. OSS has slightly smaller total seed frequency (averaged over all reads) under $t = 1$, but it quickly increases, exceeding CAS at $t > 1$. OSS outperforms CAS under $t = 1$ because greedy CAS seeding extracts seeds sequentially; while OSS scans through all possible MEM placements in a read and picks the least frequent placement. When t gets larger, OSS is



■ **Figure 8** Comparison between CAS and OSS in terms of average number of seeds used, with various edit distance thresholds t .

pressured to use more seeds, which leads to using shorter and more frequent seeds. To the contrary, greedy CAS seeding often uses fewer than t seeds, as shown in Figure 8, which let it use longer and less frequent seeds. At $t = 4$, greedy CAS seeding outperforms OSS by 20.3%.

CAS is expected to perform better on larger genomes. The *E. coli* genome is a small genome, which has only around 4.6 million base pairs. In comparison, the human genome has more than 3 billion base pairs. For small genomes, seeds become less frequent by nature. Therefore short seeds become acceptable as they are not as frequent as they are in larger genomes. We therefore expect CAS to perform better in larger genomes. However, due to practical (not theoretical) limitations in scaling up the construction of the confidence radius database on larger genomes (further elaborated in the Discussion section), we only demonstrate CAS on the *E. coli* genome.

While the focus of this paper is to establish the theoretical foundation of CAS, instead of providing a complete read mapping solution, it is worth mentioning that greedy CAS seeding (only the seeding mechanism) is more practical than OSS. OSS requires scanning through all substrings of R , which has a total size of $O(|R|^2)$, for seed frequencies. Combined with BWT, it takes at least $O(|R|^2)$ operations to collect all seed frequencies with OSS. Greedy CAS seeding, to the contrary, finishes in $O(|R| + t \log(t))$ time with $t \ll |R|$.

6 Discussion

Although Algorithm 1 finishes in $O(|\Sigma|^2 \cdot M)$ time, in practice, M could be on the scale of trillions or more, for large and complex genomes. This is because for large genomes, the suffix trie is close to full in the first ten to twenty levels, where almost every permutation of letters exists. Nodes in these levels have large numbers of neighbors: the number of neighbors of a node v , equals to the number of unique strings formed by editing the string of v with up to t edits. After each edit, the resulting string is guaranteed to appear in *Trie*. This is further amplified by the exponentially-growing number of nodes in each level. In human genomes, there are more than one billion unique 15-base-pair suffixes. This means that for

human genomes, under $t = 4$, there could be more than 1 trillion total neighbors just for 15-base-pair suffixes. Maintaining metadata at such scale vastly exceeds the capacity of our currently available computational power. From our experiment, it takes around 300 CPU hours to compute the confidence radius database for the *E. coli* genome under $t = 5$ and $P = 60$ on a multi-cpu, mechanical hard drive system. However, it is worth noting that as a theoretical study, the database construction program is not fully optimized for speed and is currently I/O-bound due to frequently reading and writing neighbor information into neighbor arrays of nodes in *Trie*.

While there are many nodes (long suffixes) with fewer neighbors, given that Algorithm 1 traverses *Trie* in a top-down manner, it is unavoidable to track the massive number of neighbors for short suffixes. This is an interesting algorithmic problem for future work.

CAS may be applied to situations other than NGS read mapping. For example, the idea of context-aware seeds may improve long-read mapping. Long reads suffer from high error rates [13, 3, 4]. Finding error-free seeds for long reads is very challenging [5]. CAS can serve as a metric measuring the likelihood of seeds having errors: if there exists a seed, s , with high confidence radius, it is highly likely that s is free of errors. The likelihood of obtaining a reference-matching seed through many accidental errors is small.

Finally, CAS can be applied to develop probes for DNA and RNA identification. When designing probe sequences, it is important to make certain that the target sequence is unique in the genome [12, 2, 10]. It prevents probes from accidentally annealing to a similar sequences. CAS checks the existence of similar sequences by consulting the confidence radius database.

7 Conclusion

In this work, we proposed a new seeding framework, context-aware seeds (CAS). CAS extends the pigeonhole principle and guarantees finding all valid mappings with fewer seeds. CAS associates each seed s with a confidence radius c_s , defined as a lower bound of edit distances towards nontrivial neighbors of s . We proved that the CAS can find all valid mappings of any read R , as long as its seeds s satisfy $\sum c_s \geq t$.

We proposed a linear-time algorithm for constructing the confidence radius database. It computes the confidence radii of seeds by traversing the suffix trie of a reference. We experimented CAS on *E. coli* genome and compared it against the state-of-the-art pigeonhole-principle-based seeding scheme, OSS, and showed that CAS outperforms OSS by reducing the sum of seed frequencies by up to 20.3%.

This paper focuses on the theoretical aspects of CAS, especially how it extends the pigeonhole principle into using fewer seeds. Composing a practical solution of Algorithm 1 on larger genomes is an interesting-yet-separate problem for future work.

Financial disclosure. C. K. is co-founder of Ocean Genomics, Inc.

References

- 1 Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.
- 2 Eric Dugat-Bony, Eric Peyretailade, Nicolas Parisot, Corinne Biderre-Petit, Faouzi Jaziri, David Hill, Sébastien Rimour, and Pierre Peyret. Detecting unknown sequences with DNA microarrays: explorative probe design strategies. *Environmental Microbiology*, 14(2):356–371, 2012.

- 3 Ehsan Haghshenas, Faraz Hach, S Cenk Sahinalp, and Cedric Chauve. Colormap: correcting long reads by mapping short reads. *Bioinformatics*, 32(17):i545–i551, 2016.
- 4 Ehsan Haghshenas, S Cenk Sahinalp, and Faraz Hach. lordFAST: sensitive and fast alignment search tool for long noisy read sequencing data. *Bioinformatics*, 35(1):20–27, 2018.
- 5 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 *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer, 2017.
- 6 Szymon M Kielbasa, Raymond Wan, Kengo Sato, Paul Horton, and Martin C Frith. Adaptive seeds tame genomic sequence comparison. *Genome Research*, 21(3):487–493, 2011.
- 7 Gad M Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.
- 8 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357, 2012.
- 9 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, 2013. [arXiv:1303.3997](https://arxiv.org/abs/1303.3997).
- 10 Qingge Li, Guoyan Luan, Qiuping Guo, and Jixuan Liang. A new class of homogeneous nucleic acid probes based on specific displacement hybridization. *Nucleic Acids Research*, 30(2):e5–e5, 2002.
- 11 Ngoc Hieu Tran and Xin Chen. AMAS: optimizing the partition and filtration of adaptive seeds to speed up read mapping. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 13(4):623–633, 2016.
- 12 Juexiao Sherry Wang and David Yu Zhang. Simulation-guided DNA probe design for consistently ultraspecific hybridization. *Nature Chemistry*, 7(7):545, 2015.
- 13 Jason L Weirather, Mariateresa de Cesare, Yunhao Wang, Paolo Piazza, Vittorio Sebastiano, Xiu-Jie Wang, David Buck, and Kin Fai Au. Comprehensive comparison of Pacific Biosciences and Oxford Nanopore Technologies and their applications to transcriptome analysis. *F1000Research*, 6, 2017.
- 14 Hongyi Xin, Donghyuk Lee, Farhad Hormozdiari, Samihan Yedkar, Onur Mutlu, and Can Alkan. Accelerating read mapping with FastHASH. *BMC Genomics*, 14(1):S13, 2013.
- 15 Hongyi Xin, Sunny Nahar, Richard Zhu, John Emmons, Gennady Pekhimenko, Carl Kingsford, Can Alkan, and Onur Mutlu. Optimal seed solver: optimizing seed selection in read mapping. *Bioinformatics*, 32(11):1632–1642, 2015.

Bounded-Length Smith–Waterman Alignment

Alexander Tiskin 

Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom
<http://warwick.ac.uk/alexiskin>

Abstract

Given a fixed alignment scoring scheme, the bounded length (respectively, bounded total length) Smith–Waterman alignment problem on a pair of strings of lengths m , n , asks for the maximum alignment score across all substring pairs, such that the first substring’s length (respectively, the sum of the two substrings’ lengths) is above the given threshold w . The latter problem was introduced by Arslan and Egecioğlu under the name “local alignment with length threshold”. They proposed a dynamic programming algorithm solving the problem in time $O(mn^2)$, and also an approximation algorithm running in time $O(rmn)$, where r is a parameter controlling the accuracy of approximation. We show that both these problems can be solved exactly in time $O(mn)$, assuming a rational scoring scheme; furthermore, this solution can be used to obtain an exact algorithm for the normalised bounded total length Smith–Waterman alignment problem, running in time $O(mn \log n)$. Our algorithms rely on the techniques of fast window-substring alignment and implicit unit-Monge matrix searching, developed previously by the author and others.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Divide and conquer; Theory of computation → Dynamic programming; Applied computing → Molecular sequence analysis; Applied computing → Bioinformatics

Keywords and phrases sequence alignment, local alignment, Smith–Waterman alignment, matrix searching

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.16

1 The framework

In this section, we introduce briefly the framework developed by the author in [23, 22, 21, 24]. Definitions and results of this section have either appeared in these publications, or are their straightforward generalisations.

1.1 Preliminaries

For matrix and vector indices, we will use either integers, or half-integers (sometimes called odd half-integers):

$$\{\dots, -2, -1, 0, 1, 2, \dots\} \quad \left\{ \dots, -\frac{5}{2}, -\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \dots \right\}$$

For ease of reading, half-integer variables will be indicated by hats (e.g. \hat{i} , \hat{j}). Ordinary variable names (e.g. i , j , with possible subscripts or superscripts), will normally denote integer variables, but can sometimes denote a variable that may be either integer, or half-integer.

It will be convenient to denote

$$i^- = i - \frac{1}{2} \quad i^+ = i + \frac{1}{2}$$

for any integer or half-integer i . The set of all half-integers can now be written as

$$\{\dots, (-3)^+, (-2)^+, (-1)^+, 0^+, 1^+, 2^+, \dots\}$$

We denote integer and half-integer *intervals* by

$$[i : j] = \{i, i + 1, \dots, j - 1, j\} \quad \langle i : j \rangle = \left\{ i^+, i + \frac{3}{2}, \dots, j - \frac{3}{2}, j^- \right\}$$



© Alexander Tiskin;

licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 16; pp. 16:1–16:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

16:2 Bounded-Length Smith–Waterman Alignment

In both cases, the interval is defined by a pair of integer *endpoints*. We will use round parentheses $(i : j)$ to indicate that in the given context, either square $[i : j]$ or angle $\langle i : j \rangle$ brackets may be used.

For finite intervals $[i : j]$ and $\langle i : j \rangle$, we call the difference $j - i$ interval *length*. Note that for a given length n , an integer interval consists of $n + 1$ elements, and a half-integer interval of n elements.

We use the following notation for the Cartesian product of two integer or two half-integer intervals:

$$(i : i' \mid j : j') = (i : i') \times (j : j')$$

We will denote singleton Cartesian products by $(i, j) = (i \mid j)$.

We indicate the index range of a matrix by juxtaposition, e.g. matrix $A(i : i' \mid j : j')$ over an integer or half-integer index range. The same notation will be used for selecting subvectors and submatrices: for example, given matrix $A(0 : n \mid 0 : n)$, we denote by $A(i : i' \mid j : j')$ the submatrix defined by the given sub-intervals. When any of the indices i, i', j, j' coincide with the range boundary, they can be omitted, e.g. $A(i : \mid j : j') = A(i : n \mid 0 : j')$, and $A(\mid j : j') = A(0 : n \mid j : j')$. In particular, we write $A(i, j)$ for a single matrix element, $A(i \mid \mid)$ for a row, and $A(\mid \mid j)$ for a column. We will denote by $\sum A$ the sum of all elements in matrix A .

By default, vectors and matrices will be indexed by integers based at 0, or by half-integers based at $0^+ = \frac{1}{2}$. Where necessary, all our definitions and statements can easily be generalised to indexing over arbitrary integer or half-integer intervals.

Given a string, we distinguish between its contiguous *substrings*, and not necessarily contiguous *subsequences*. Special cases of a substring are *a prefix* and *a suffix* of a string. Unless indicated otherwise, an algorithm's input is a string a of length m , and a string b of length n ; we assume $m \leq n$. Normally, we will say that two characters α, β *match*, if $\alpha = \beta$, and *mismatch* otherwise. In addition to this, we introduce the *wildcard character* '?', which matches itself and all other characters.

► **Definition 1.** Let $D(0 : n_1 \mid 0 : n_2)$ be a matrix. Its dominance-sum matrix (also called distribution matrix) $D^\nearrow[0 : n_1 \mid 0 : n_2]$ is defined by

$$D^\nearrow[i, j] = \sum D\langle i : \mid j \rangle$$

for all $i \in [0 : n_1], j \in [0 : n_2]$.

► **Definition 2.** Let $A[0 : n_1 \mid 0 : n_2]$ be a matrix. Its cross-difference matrix $A^\square\langle 0 : n_1 \mid 0 : n_2 \rangle$ (also called density matrix) is defined by

$$A^\square\langle \hat{i}, \hat{j} \rangle = A[\hat{i}^+, \hat{j}^-] - A[\hat{i}^-, \hat{j}^-] - A[\hat{i}^+, \hat{j}^+] + A[\hat{i}^-, \hat{j}^+]$$

for all $\hat{i} \in \langle 0 : n_1 \rangle, \hat{j} \in \langle 0 : n_2 \rangle$.

► **Definition 3.** Matrix A is a Monge matrix, if its cross-difference matrix A^\square is nonnegative. Matrix A is an anti-Monge matrix, if its negative $-A$ is Monge.

The structure of Monge matrices has been described by Burdyuk and Trofimov [6] and Bein and Pathak [4] (see also [8, 7]), and can be used to represent a Monge matrix implicitly by its cross-difference matrix and a pair of vectors.

► **Theorem 4** (Monge matrix canonical decomposition). *Let $A[0 : n_1 \mid 0 : n_2]$ be a Monge matrix. Let $D = A^\square$, $b = A[n_1 \mid :]$, $c = A[: 0]$. We have*

$$A[i, j] = D^\nearrow[i, j] + b[j] + c[i] - b[0] \quad (1)$$

for all $i, j \in [0 : n_1 \mid 0 : n_2]$.

► **Definition 5.** *A permutation matrix is a zero-one matrix containing exactly one nonzero in every row and every column.*

► **Definition 6.** *Matrix A is called unit-Monge, if its cross-difference matrix A^\square is a permutation matrix. Matrix A is called unit-anti-Monge, if its negative $-A$ is unit-Monge.*

An efficient row minima searching algorithm on a canonically represented unit-Monge matrix was given by Gawrychowski [10].

► **Theorem 7.** *Let A be a unit-Monge matrix, canonically represented by the permutation matrix $P = A^\square$ with n nonzeros, and by implicit vectors b, c , where each vector element can be queried in time $O(1)$. Given P, b, c , a data structure can be computed in time $O(n \log \log n)$ in the pointer machine model, and in time $O(n)$ in the unit-cost RAM model, such that the index of the leftmost minimum element in a row of A can be queried in time $O(1)$.*

1.2 LCS and semi-local LCS

► **Definition 8.** *Let a, b be strings. The longest common subsequence (LCS) score $lcs(a, b)$ is the length of the longest string that is a subsequence of both a and b . Given strings a, b , the LCS problem asks for the LCS score $lcs(a, b)$.*

The classical dynamic programming algorithm for the LCS problem [17, 25] runs in time $O(mn)$. The best known algorithms speed it up by a (model-dependent) polylogarithmic factor [16, 9, 5]. Similar speedups are possible for the algorithms presented in this paper; however, we will consider them outside the paper’s scope, and will not discuss them any further.

Although global comparison (full string against full string) and fully-local comparison (all substrings against all substrings) are the two most common approaches to comparing strings, another important type of string comparison lies “in between”.

► **Definition 9.** *Given strings a, b , the semi-local LCS problem asks for the LCS scores as follows:*

- *the whole a against every substring of b (string-substring LCS);*
- *every prefix of a against every suffix of b (prefix-suffix LCS);*
- *every suffix of a against every prefix of b (suffix-prefix LCS);*
- *every substring of a against the whole b (substring-string LCS).*

Note that this definition is symmetric with respect to exchanging the two strings. In many ways, our approach consists in exploiting to the full this and other symmetries of the LCS problem, such as the symmetry between the left and the right within each of the strings.

Some alternative terms for semi-local comparison, used especially in biological texts, are “end-free alignment” [14], [13, Subsection 11.6.4] or “semi-global alignment” [14], [15, Problem 6.24]. The string-substring (and its symmetric substring-string) component of semi-local string comparison is also called “fitting alignment” [15, Problem 6.23]. String-substring LCS is an important problem in its own right, closely related to approximate pattern matching, where a short fixed pattern string is compared to various substrings of a long text string.

Let $b^{pad} \langle -m : m+n \rangle = ?^m \underline{b} ?^m$.

► **Definition 10.** The semi-local LCS matrix is defined as $H_{a,b}[i, j] = \text{lcs}(a, b^{\text{pad}}\langle i, j \rangle)$, where $i \in [-m : n]$, $j \in [0 : m + n]$. The string-substring LCS matrix is defined as $H_{a,b}^{\text{s.sub}}[i, j] = H_{a,b}[i, j] = \text{lcs}(a, b\langle i : j \rangle)$, where $i, j \in [0 : n]$.

In [23], we show that matrix $H_{a,b}$ is unit-anti-Monge. By Theorem 4, it can be represented implicitly in compact form by a permutation matrix $P_{a,b}$, that we call here *semi-local LCS kernel*, from which every element of $H_{a,b}$ (and therefore also of its submatrix $H_{a,b}^{\text{s.sub}}$) can be queried efficiently. In [23], we also give an algorithm computing this implicit representation in time $O(mn)$.

1.3 Alignment

The concept of LCS score is generalised by that of *alignment score* (see e.g. [14]). An *alignment* of strings a, b is obtained by putting a subsequence of a into one-to-one correspondence with a subsequence of b , respecting the index order. In contrast with an LCS, the two subsequences need not be identical. A pair of corresponding characters, one from a and the other from b , are said to be *aligned*. A character in one string that is not aligned against a character of the other string is said to be aligned against a *gap* in that string. An aligned character-character or character-gap pair is given a real-valued *score*:

- a pair of matching characters scores $w_+ \geq 0$;
- a pair of mismatching characters scores $w_0 < w_+$;
- a gap-character or character-gap pair scores $w_- \leq \frac{1}{2}w_0$; normally, also $w_- \leq 0$.

Negative scores are also called *penalties*. Any particular triple of character alignment scores (w_+, w_0, w_-) will be called a *scoring scheme*. A scoring scheme will be called *rational*, if all its components are rational numbers.

The intuition behind the score inequalities is as follows: aligning a matching pair of characters is always better than aligning a mismatching pair of characters, while the latter is at least as good as aligning each of the two characters against a gap.

► **Definition 11.** Let a, b be strings. Assuming a fixed scoring scheme, the alignment score $\text{align}(a, b)$ is the maximum total score across all possible alignments of a, b . Given strings a, b , the alignment problem asks for their alignment score.

In notation related to alignment under a general scoring scheme, we will use script typeface (e.g. \mathcal{H}), in order to distinguish it from the LCS scoring scheme, for which we keep using sans-serif typeface (e.g. H).

► **Definition 12.** The semi-local alignment problem and its component subproblems (string-substring alignment problem, etc.) are defined analogously to Definition 9, replacing the LCS score by the alignment score. The semi-local alignment matrix $\mathcal{H}_{a,b}$ and the string-substring alignment matrix $\mathcal{H}_{a,b}^{\text{s.sub}}$ are defined analogously to Definition 10.

The semi-local alignment problem can be reduced to the semi-local LCS problem by a couple of simple techniques, *regularisation* and *blow-up*, described (using slightly different terminology) in [22].

► **Definition 13.** A scoring scheme $(1, w_0, 0)$, where $0 \leq w_0 < 1$, will be called *regular*.

An arbitrary (finite) scoring scheme can be reduced to a regular one as follows (a similar method is used by Rice et al. [18]; see also [12, 15]). Let us consider first the global alignment of strings a, b , with an arbitrary scoring scheme (w_+, w_0, w_-) . This scheme can be replaced by a scheme $(w_+ + 2x, w_0 + 2x, w_- + x)$ for any real x . Indeed, such a transformation increases

the score of every global alignment by $(m+n)x$; therefore, the relative scores of different global alignments do not change. The desired regular scoring scheme can be obtained by taking $x = -w_-$, and then dividing each of the resulting scores by $w_+ - 2w_- > 0$:

$$(w_+, w_0, w_-) \mapsto (1, w_0^* = \frac{w_0 - 2w_-}{w_+ - 2w_-}, 0) \quad (2)$$

The resulting regular string alignment score h^* and the original alignment score h are related to each other by the following identities, defining regularisation and its reverse:

$$h^* = \frac{h - (m+n)w_-}{w_+ - 2w_-} \quad h = h^* \cdot (w_+ - 2w_-) + (m+n) \cdot w_- \quad (3)$$

► **Definition 14.** A bistochastic matrix is a matrix of nonnegative real elements, in which the sum of elements is exactly one in every row and every column.

► **Definition 15.** A Monge matrix A is called regular, if its cross-difference matrix A^\square is bistochastic. An anti-Monge matrix A is called regular, if its negative $-A$ is regular.

► **Theorem 16** (Alignment matrix canonical decomposition). Let a, b be strings. Assuming a fixed scoring scheme, the semi-local alignment matrix $\mathcal{H}_{a,b}$ is anti-Monge. Furthermore, for a regular scoring scheme, $\mathcal{H}_{a,b}$ is regular anti-Monge:

$$\mathcal{H}_{a,b}[i, j] = j - i - \mathcal{S}_{a,b}^\triangleright[i, j]$$

for all $i \in [-m : n]$, $j \in [0 : m+n]$, where $\mathcal{S}_{a,b} = -\mathcal{H}_{a,b}^\square$ is a bistochastic matrix.

Proof. The anti-Monge property of $\mathcal{H}_{a,b}$ is a straightforward generalisation of a similar property for semi-local LCS matrices.

Assume a regular scoring scheme $(1, w_0, 0)$. First, let us also assume it to be rational: $w_0 = \frac{\nu}{\nu}$. Then, the semi-local alignment problem on strings a, b can be reduced to the semi-local LCS problem by the following *blow-up* technique. We transform input strings a, b of lengths m, n into *blown-up* strings \mathbf{a}, \mathbf{b} of lengths $\mathbf{m} = \nu m, \mathbf{n} = \nu n$. The transformation replaces every character γ by a substring $\mathcal{S}^\mu \gamma^{\nu-\mu}$ of length ν (here, \mathcal{S} is a special guard character, not present in the original strings). We have

$$\mathcal{H}_{a,b}[i, j] = \frac{1}{\nu} \cdot \mathbf{H}_{\mathbf{a},\mathbf{b}}[\nu i, \nu j] \quad (4)$$

for all $i \in [-m : n]$, $j \in [0 : m+n]$, where the alignment matrix $\mathcal{H}_{a,b}$ is defined by the given scoring scheme on the original strings a, b , and the LCS matrix $\mathbf{H}_{\mathbf{a},\mathbf{b}}$ by the LCS scoring scheme on the blown-up strings \mathbf{a}, \mathbf{b} .

Every element of $\mathcal{S}_{a,b}$ can now be obtained as a scaled sum of a $\nu \times \nu$ block of $\mathbf{P}_{\mathbf{a},\mathbf{b}}$:

$$\mathcal{S}_{a,b}[\hat{i}, \hat{j}] = \frac{1}{\nu} \sum \mathbf{P}_{\mathbf{a},\mathbf{b}}[\nu \hat{i}^- : \nu \hat{i}^+ | \nu \hat{j}^- : \nu \hat{j}^+] \quad (5)$$

for all $\hat{i} \in \langle -m : n \rangle$, $\hat{j} \in \langle 0 : m+n \rangle$. By construction, matrix $\mathcal{S}_{a,b}$ is bistochastic.

A general regular scoring scheme can be approximated by a rational regular scheme to arbitrary precision. The property of matrix $\mathcal{S}_{a,b}$ to be bistochastic is preserved in the limit, therefore the theorem statement follows by compactness. ◀

► **Definition 17.** Assuming a regular scoring scheme, the semi-local alignment kernel for strings a, b is the bistochastic matrix $\mathcal{S}_{a,b} \langle -m : n | 0 : m+n \rangle$, determined by Theorem 16. The string-substring alignment kernel is the submatrix $\mathcal{S}_{a,b}^{s,sub} = \mathcal{S}_{a,b} \langle 0 : n | 0 : n \rangle$

The alignment matrix $\mathcal{H}_{a,b}$ can be represented implicitly by (the nonzeros of) the alignment kernel $\mathcal{S}_{a,b}$. However, in contrast with the LCS kernel, the alignment kernel may not be a permutation matrix. Sparsity of the alignment kernel is partially preserved for a rational scoring scheme.

► **Definition 18.** Let $\nu > 0$ be a natural number. A bistochastic matrix will be called ν -bistochastic, if every its element is a multiple of $1/\nu$. A regular Monge matrix A will be called ν -regular, if A^\square is ν -bistochastic. A regular anti-Monge matrix will be called ν -regular, if $-A$ is ν -regular.

A permutation matrix is 1-bistochastic, and a unit-Monge matrix is 1-regular Monge. Note that in a ν -bistochastic matrix, there can be at most ν nonzeros in every row and every column.

► **Theorem 19.** Let a, b be strings. Assuming a regular rational scoring scheme with denominator ν , the alignment kernel $\mathcal{S}_{a,b}$ is ν -bistochastic, and the alignment matrix $\mathcal{H}_{a,b}$ is ν -regular anti-Monge.

Proof. Every element of $\mathcal{S}_{a,b}$ is the sum of a $\nu \times \nu$ block of the blown-up matrix $\frac{1}{\nu}P_{a,b}$ by (5), and is therefore a multiple of $1/\nu$. ◀

Theorem 19 implies that for $\nu = O(1)$, the alignment kernel $\mathcal{S}_{a,b}$ has $O(m+n)$ nonzeros. The described reduction also preserves the $O(mn)$ running time for computing the alignment kernel given a pair of strings, same as for the LCS kernel.

1.4 Window-substring alignment

Given a string and a fixed integer $w \geq 0$, we call any substring of length w a w -window.

► **Definition 20.** Given strings a, b , and a window length w , the window-substring alignment problem asks for the alignment score of every w -window in string a against every substring in string b .

In [23], we gave an efficient algorithm for the window-substring alignment problem.

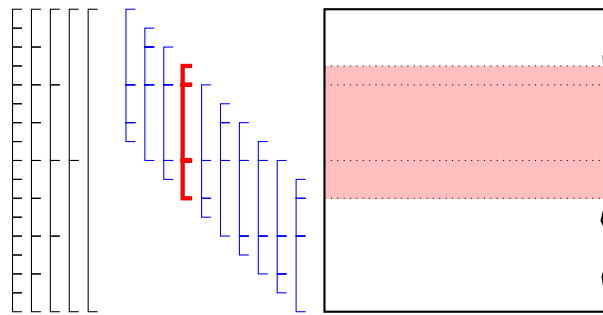
► **Theorem 21.** The window-substring alignment problem can be solved implicitly in time $O(mn)$, providing the window-substring alignment kernel $\mathcal{S}_{a^{(k:l)},b}^{s,sub}$ for every $k, l \in [0 : m]$, $l - k = w$.

Proof. See [23]. Briefly, we compute a string-substring alignment kernel for a binary tree of specially defined *canonical substrings* of a against whole b . We then assemble these kernels into window-substring alignment kernels by a special procedure of implicit matrix multiplication, fully described in [24]. ◀

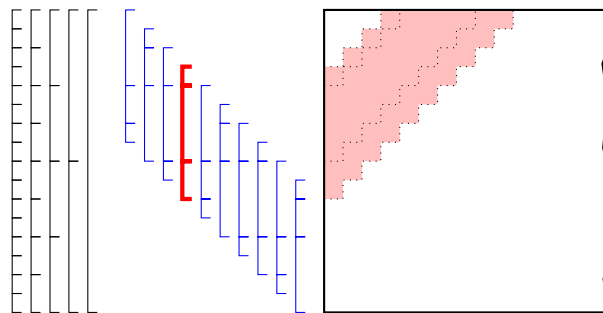
Definition 20 can be extended to other types of *length-constrained comparison*. In particular, we can consider a related symmetric problem, where the total length of strings a, b is fixed.

► **Definition 22.** Given strings a, b , and a length parameter w , the fixed total length alignment problem asks for the alignment score of every substring in string a against every substring in string b , such that the sum of the two substrings' lengths is w .

The fixed total length alignment problem can be solved by a technique similar to to the algorithm of Theorem 21. Observe that a window-substring alignment kernel corresponds to a horizontal rectangular strip of width w in the alignment grid, defined by the window in string a and the whole string b . To obtain an algorithm for the fixed total length alignment



■ **Figure 1** Window-substring alignment.



■ **Figure 2** Fixed total length alignment.

problem with a rational scoring scheme, we replace these with an alternative type of kernel, corresponding to an anti-diagonal strip of rectilinear width w . Otherwise, the algorithm's structure is identical to the one of Theorem 21, and the running time remains $O(mn)$.

► **Example 23.** Figure 1 shows the computation of window-substring alignment by the algorithm of Theorem 21 on string a of length $m = 16$ with window size $w = 7$, against string b of arbitrary length n . The canonical substrings of a of lengths 1, 2, 4, 8, 16 are shown in black, and 7-windows in blue. For each 7-window, the figure shows its decomposition into canonical substrings. For one of the 7-windows, highlighted in thick red, the corresponding area is outlined in the alignment dag.

Figure 2 shows the computation of fixed total length alignment on string a of length $m = 16$ with length parameter $w = 7$, against string b of arbitrary length n . Conventions are the same as in Figure 1.

1.5 Approximate pattern matching

Approximate pattern matching is a natural generalisation of classical (exact) pattern matching, allowing for some character differences between the pattern and a matching substring of the text. Given a *pattern string* p of length m and a *text string* t of length $n \geq m$, approximate pattern matching asks for all the substrings of the text that are close to the pattern, i.e. those that have sufficiently high alignment score (or, equivalently, sufficiently low edit distance) against the pattern. Such substrings of the text will be called *matching substrings*.

► **Definition 24.** *Given a pattern p and a text t , and assuming a fixed scoring scheme, the complete approximate matching problem is defined as follows. For every prefix $t\langle 0 : j \rangle$, the problem asks for the maximum alignment score of p against all possible choices of a suffix*

16:8 Bounded-Length Smith–Waterman Alignment

$t\langle i : j \rangle$ from this prefix:

$$h[j] = \max_{i \in [0:j]} \text{align}(p, t\langle i : j \rangle)$$

where $j \in [0 : n]$.

The complete approximate matching problem corresponds to searching for column maxima in the string-substring alignment matrix $\mathcal{H}_{p,t}^{s,sub}$.

The complete approximate pattern matching problem can be solved by a classical dynamic programming algorithm due to Sellers [19] (see also [14]), running in time $O(mn)$.

Assuming a rational scoring scheme with constant denominator, an algorithm for complete approximate matching can also be obtained using alignment kernels.

► **Theorem 25.** *Let $A[0 : n_1 \mid 0 : n_2]$ be a ν -regular Monge matrix, where $\nu = O(1)$, canonically represented by the ν -bistochastic matrix $S = A^\square$ with $n \leq \nu \cdot \min(n_1, n_2)$ nonzeros, and implicit vectors $b = A[n_1 \mid :]$, $c = A[: \mid 0]$, where random access to an element can be performed in time $O(1)$. Given S , b , c , the index of the leftmost minimum element in every row of A can be obtained in time $O(n \log \log n)$ in the pointer machine model, and in time $O(n)$ in the unit-cost RAM model.*

Proof. Straightforward generalisation of Theorem 7. ◀

► **Theorem 26.** *Assuming a rational scoring scheme with denominator $\nu = O(1)$, the complete approximate matching problem can be solved in time $O(mn)$.*

Proof. The algorithm runs in two phases.

Phase 1. We first regularise the scoring scheme by Equation (3). The alignment kernel $\mathcal{S}_{p,t}^*$ for the regular scheme is then obtained in time $O(mn)$.

Phase 2. Since complete approximate matching compares alignment scores across different string-substring pairs, we now reverse score regularisation via Equation (3). We then obtain column maxima of the (suitably scaled) string-substring submatrix $\mathcal{H}_{p,t}^{s,sub}$ by Theorem 25.

The total running time is dominated by Phase 1, and is therefore as claimed. ◀

2 Classical Smith–Waterman alignment

A classical method for variable-length local string comparison was given by Smith and Waterman [20]. Given a scoring scheme and a pair of input strings a , b , this method allows one to obtain a pair of substrings with the highest alignment score across all substring pairs in a , b . More generally, for each pair of prefixes of strings a , b , the method obtains their highest-scoring pair of suffixes.

► **Definition 27.** *Given a scoring scheme, the Smith–Waterman (SW) alignment problem on strings a , b is defined as follows. For every prefix $a\langle 0 : l \rangle$ and every prefix $b\langle 0 : j \rangle$, the problem asks for the maximum alignment score over all possible choices of a suffix $a\langle k : l \rangle$ and a suffix $b\langle i : j \rangle$ of the respective prefixes:*

$$h[l, j] = \max_{k \in [0:l], i \in [0:j]} \text{align}(a\langle k : l \rangle, b\langle i : j \rangle)$$

where $l \in [0 : m]$, $j \in [0 : n]$.

An efficient dynamic programming algorithm for SW-alignment was given by Smith and Waterman [20] and by Gotoh [11].

► **Algorithm 28** (SW-alignment).

Parameters: scoring scheme (w_+, w_0, w_-) .

Input: strings a, b of length m, n , respectively.

Output: SW-alignment scores for a against b .

Description. We initialise

$$h[0, j] \leftarrow 0 \quad h[l, 0] \leftarrow 0$$

for all $l \in [0 : m], j \in [0 : n]$. We then iterate through all indices $l \in [1 : m]$ and $j \in [1 : n]$. In each iteration, we assign

$$h[l, j] \leftarrow \max \begin{cases} h[l-1, j-1] + \begin{cases} w_+ & \text{if } a\langle l^- \rangle \text{ matches } b\langle j^- \rangle \\ w_0 & \text{otherwise} \end{cases} \\ h[l-1, j] + w_-; h[l, j-1] + w_-; 0 \end{cases}$$

► **Theorem 29.** *The SW-alignment problem can be solved in time $O(mn)$.*

Proof. In Algorithm 28, each iteration runs in constant time. The overall running time is $mn \cdot O(1) = O(mn)$. ◀

3 Bounded-length Smith–Waterman alignment

A commonly observed drawback in biological applications of SW-alignment is that it seeks to maximise the scores of substring pairs regardless of their lengths. However, if the substrings are too short, then their alignment may be less significant biologically than a slightly lower-scoring alignment of much longer substrings. To address this issue, various alternative definitions of local string comparison have been proposed, taking substring lengths into account; see [2] for a survey. Algorithms for such length-sensitive alignment problems are typically more computationally expensive than Algorithm 28 (SW-alignment); for this reason, approximation versions of such problems have also been considered [2].

The most straightforward approach to defining length-sensitive local alignment is to require that the aligned substrings satisfy a specific lower bound on their lengths, thus filtering out substrings that are too short.

► **Definition 30.** *Given strings a, b , an length threshold $w \geq 0$, and assuming a fixed scoring scheme, the bounded length (respectively, the bounded total length) SW-alignment problem is defined as follows. For every prefix $a\langle 0 : l \rangle$ and every prefix $b\langle 0 : j \rangle$, the problem asks for the maximum alignment score over all possible choices of a suffix $a\langle k : l \rangle$ and a suffix $b\langle i : j \rangle$ of the respective prefixes, such that $l - k \geq w$ (respectively, $l - k + j - i \geq w$):*

$$h_{len \geq w}[l, j] = \max_{\substack{k \in [0:l], i \in [0:j] \\ l-k \geq w}} \text{align}(a\langle k : l \rangle, b\langle i : j \rangle)$$

$$h_{tlen \geq w}[l, j] = \max_{\substack{k \in [0:l], i \in [0:j] \\ l-k+j-i \geq w}} \text{align}(a\langle k : l \rangle, b\langle i : j \rangle)$$

where $l \in [0 : m], j \in [0 : n]$.

The bounded total length SW-alignment problem was introduced by Arslan and Egecioğlu [1] (see also [2]) under the name *local alignment with length threshold (LAt)*. They proposed a dynamic programming algorithm solving the problem in time $O(mn^2)$. They also gave an

16:10 Bounded-Length Smith–Waterman Alignment

approximation algorithm running in time $O(rmn)$, and returning a pair of substrings of total length at least $(1 - \frac{1}{r})w$, scoring no less than the exact solution to the original problem.

We now show that, assuming a rational scoring scheme, the bounded (total) length SW-alignment problem can be solved exactly by an efficient algorithm. In fact, our exact algorithm is asymptotically faster not only than the exact algorithm of [1], but also than their approximation algorithm. We first describe an algorithm for the bounded length alignment problem, and then adjust it to the bounded total length alignment problem.

► **Algorithm 31** (Bounded-length SW-alignment).

Parameters: a rational scoring scheme (w_+, w_0, w_-) , where $w_- < 0$.

Input: strings a, b of length m, n , respectively; length threshold $w \geq 0$.

Output: Bounded length SW-alignment scores for a against b .

Description. The algorithm runs in three phases:

1. solving the window-substring alignment problem for every w -window in a against every substring in b ;
2. solving the complete approximate matching problem for every w -window in a against b : for every w -window $a\langle k : l \rangle$ in a and every prefix $b\langle 0 : j \rangle$ of b , we find the maximum alignment score of $a\langle k : l \rangle$ against all possible choices of a suffix $b\langle i : j \rangle$ from this prefix;
3. extending the complete approximate matching scores to optimal bounded-length SW-alignment scores by a dynamic programming procedure that generalises Algorithm 28 (SW-alignment).

We now describe each of these phases in more detail.

Phase 1. By Theorem 21, we obtain the window-substring alignment kernel $\mathcal{S}_{a\langle k:l \rangle, b}^{s.sub}$ for every $k, l, l - k = w$.

Phase 2. We now consider the window-substring alignment matrices $\mathcal{H}_{a\langle k:l \rangle, b}^{s.sub}$ for every $k, l, l - k = w$. Every such matrix is represented implicitly by the window-substring alignment kernel $\mathcal{S}_{a\langle k:l \rangle, b}^{s.sub}$, obtained in the first phase. By Theorem 26, for every w -window $a\langle k : l \rangle$ in a and every prefix $b\langle 0 : j \rangle$ of b , we find a suffix $b\langle i : j \rangle$ of that prefix, that has the highest alignment score against $a\langle k : l \rangle$:

$$h_{len=w}[l, j] = \max_{i \in [0:j]} \mathcal{H}_{a\langle k:l \rangle, b}^{s.sub}[i, j]$$

Phase 3. We initialise

$$h_{len \geq w}[w, j] \leftarrow h_{len=w}[w, j] \quad h_{len \geq w}[l, 0] \leftarrow 0$$

for all $l \in [w : m], j \in [0 : n]$. We then iterate through all indices $l \in [w + 1 : m]$ and $j \in [1 : n]$. In each iteration, we assign

$$h_{len \geq w}[l, j] \leftarrow \max \begin{cases} h_{len \geq w}[l - 1, j - 1] + \begin{cases} w_+ & \text{if } a\langle l^- \rangle \text{ matches } b\langle j^- \rangle \\ w_0 & \text{otherwise} \end{cases} \\ h_{len \geq w}[l - 1, j] + w_-; \quad h_{len \geq w}[l, j - 1] + w_-; \quad h_{len=w}[l, j] \end{cases}$$

► **Theorem 32.** *The bounded length SW-alignment problem can be solved in time $O(mn)$.*

Proof. In Algorithm 31, the running time of each of the three phases, and therefore the overall running time, is $O(mn)$. ◀

The bounded total length alignment problem can be solved by an algorithm structured similarly to Algorithm 31. In this algorithm Phase 1, instead of the window-substring alignment problem, solves the fixed total length alignment problem. Phase 2 obtains column maxima in the resulting implicit alignment matrices. Phase 3 remains unchanged. The algorithm's running time remains $O(mn)$.

4 Normalised bounded-length Smith–Waterman alignment

Arslan et al. [3] introduced another, more sophisticated approach to length-sensitive local alignment, by incorporating the substrings' lengths into the scoring function.

► **Definition 33.** *Let a, b be strings. Assuming a fixed scoring scheme, the strings' normalised alignment score is defined as their alignment score divided by their total length:*

$$\mathit{nalign}(a, b) = \frac{\mathit{align}(a, b)}{m + n}$$

Analogously to Definition 30, the normalised bounded total length SW-alignment problem was introduced by Arslan and Egecioglu [1] (see also [2]) under the name *normalised local alignment with length threshold (NLAt)*. They gave an approximation algorithm running in time $O(rmn \log n)$, assuming a rational scoring scheme, and returning a pair of substrings of total length at least $(1 - \frac{1}{r})w$, scoring no less than the exact solution to the original problem. The algorithm uses the authors' approximation algorithm for ordinary (unnormalised) bounded total length SW-alignment as a subroutine; the only dependence on parameter r is within that subroutine. By replacing this subroutine with Algorithm 31, we obtain an exact algorithm for normalised bounded total length SW-alignment running in time $O(mn \log n)$, assuming a rational scoring scheme.

5 Conclusion and open problems

We have described an efficient algorithm for bounded-length alignment and bounded total length alignment; our algorithm can also be used as a subroutine for efficient normalised bounded total length alignment. The algorithm relies on a rather complex framework of semi-local string comparison, window-substring alignment and implicit matrix searching, developed previously by the author and others. It remains a challenge to simplify and streamline our algorithm to a point where it can be easily and efficiently implemented.

The algorithm relies on the scoring scheme to be rational, with the least common denominator of the character alignment scores considered to be a constant factor in the running time. It remains an open question whether the algorithm can be generalised to an arbitrary real-weighted scoring scheme.

References

- 1 Abdullah N. Arslan and Ömer Egecioglu. Dynamic Programming Based Approximation Algorithms for Sequence Alignment with Constraints. *INFORMS Journal on Computing*, 16(4):441–458, 2004.
- 2 Abdullah N. Arslan and Ömer Egecioglu. Dynamic and fractional programming-based approximation algorithms for sequence alignment with constraints. In *Handbook of Approximation Algorithms and Metaheuristics*, Chapman and Hall/CRC Computer and Information Science Series, chapter 76, pages 76–1–76–16. Chapman and Hall/CRC, 2007.
- 3 Abdullah N. Arslan, Ömer Egecioglu, and Pavel A. Pevzner. A new approach to sequence comparison: Normalized sequence alignment. *Bioinformatics*, 17(4):327–337, 2001.
- 4 Wolfgang W. Bein and Pramod K. Pathak. A characterization of the Monge property and its connection to statistics. *Demonstratio Mathematica*, 29(2):451–457, 1996.
- 5 Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008.
- 6 V. Y. Burdyuk and V. N. Trofimov. Generalization of the results of Gilmore and Gomory on the solution of the traveling salesman problem. *Engineering Cybernetics*, 14:12–18, 1976.

16:12 Bounded-Length Smith–Waterman Alignment

- 7 Rainer E. Burkard. Monge properties, discrete convexity and applications. *European Journal of Operational Research*, 176(1):1–14, 2007.
- 8 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.
- 9 Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. A Subquadratic Sequence Alignment Algorithm for Unrestricted Scoring Matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003.
- 10 Paweł Gawrychowski. Faster algorithm for computing the edit distance between SLP-compressed strings. In *Lecture Notes in Computer Science*, volume 7608 of *Lecture Notes in Computer Science*, pages 229–236, 2012.
- 11 Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.
- 12 D. Gusfield, K. Balasubramanian, and D. Naor. Parametric optimization of sequence alignment. *Algorithmica*, 12(4-5):312–326, 1994.
- 13 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge, 1997.
- 14 Benjamin Jackson and Srinivas Aluru. Pairwise Sequence Alignment. In *Handbook of Computational Molecular Biology*, Chapman and Hall/CRC Computer and Information Science Series, chapter 1, pages 1–1–1–32. Chapman and Hall/CRC, 2010.
- 15 N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms*. Computational Molecular Biology. The MIT Press, 2004.
- 16 William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.
- 17 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.
- 18 S. V. Rice, H. Bunke, and T. A. Nartker. Classes of Cost Functions for String Edit Distance. *Algorithmica*, 18(2):271–280, 1997.
- 19 Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980.
- 20 T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.
- 21 A. Tiskin. Faster subsequence recognition in compressed strings. *Journal of Mathematical Sciences*, 158(5):759–769, 2009.
- 22 Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008.
- 23 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008.
- 24 Alexander Tiskin. Fast Distance Multiplication of Unit-Monge Matrices. *Algorithmica*, 71(4):859–888, 2015.
- 25 Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *Journal of the ACM*, 21(1):168–173, 1974.

Validating Paired-End Read Alignments in Sequence Graphs

Chirag Jain

School of Computational Science and Engineering, Georgia Institute of Technology, USA
cjain@gatech.edu

Haowen Zhang

School of Computational Science and Engineering, Georgia Institute of Technology, USA
hwzhang@gatech.edu

Alexander Diltthey

Institute of Medical Microbiology, University Hospital of Düsseldorf, Germany
Alexander.Diltthey@med.uni-duesseldorf.de

Srinivas Aluru

School of Computational Science and Engineering, Georgia Institute of Technology, USA
aluru@cc.gatech.edu

Abstract

Graph based non-linear reference structures such as variation graphs and colored de Bruijn graphs enable incorporation of full genomic diversity within a population. However, transitioning from a simple string-based reference to graphs requires addressing many computational challenges, one of which concerns accurately mapping sequencing read sets to graphs. Paired-end Illumina sequencing is a commonly used sequencing platform in genomics, where the paired-end distance constraints allow disambiguation of repeats. Many recent works have explored provably good index-based and alignment-based strategies for mapping individual reads to graphs. However, validating distance constraints efficiently over graphs is not trivial, and existing sequence to graph mappers rely on heuristics. We introduce a mathematical formulation of the problem, and provide a new algorithm to solve it exactly. We take advantage of the high sparsity of reference graphs, and use sparse matrix-matrix multiplications (SpGEMM) to build an index which can be queried efficiently by a mapping algorithm for validating the distance constraints. Effectiveness of the algorithm is demonstrated using real reference graphs, including a human MHC variation graph, and a pan-genome de-Bruijn graph built using genomes of 20 *B. anthracis* strains. While the one-time indexing time can vary from a few minutes to a few hours using our algorithm, answering a million distance queries takes less than a second.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Applied computing → Computational genomics

Keywords and phrases Sequence graphs, read mapping, index, sparse matrix-matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.17

Funding This work is supported in part by the US National Science Foundation under CCF-1816027.

Acknowledgements The authors thank Abdurrahman Yasar, Siva Rajamanickam and Srinivas Eswar for sharing their insights on sparse matrix manipulations.

1 Introduction

Owing to continuous technological and algorithmic advancements in genomics during the past four decades, whole-genome sequencing has now become ubiquitous, leading to an explosive growth in genome databases. Despite this progress, the current human genome reference (GRCh38) is primarily derived from a single individual [14, 38]. Many recent studies have demonstrated improved variant analysis using a graph-based reference while accounting



© Chirag Jain, Haowen Zhang, Alexander Diltthey, and Srinivas Aluru;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 17; pp. 17:1–17:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for population diversity. Accounting for this diversity is especially critical in polymorphic regions of genomes [10, 17]. Realizing this paradigm shift from a simple linear reference to a graph-based reference, however, requires addressing several open computational challenges [5], one of which concerns designing robust algorithms for mapping reads to graph-based references. Accuracy of read mapping is critical for downstream biological analyses.

Designing provably good algorithms for approximate sequence matching to graphs, using both index-based and alignment-based approaches, remains an active research area. A few recent works have investigated extending Burrows-Wheeler-Transform-based indexing to sequence DAGs [41] and de-Bruijn graphs [2, 29, 40]. Similarly, there exist studies that have explored extension of the classic sequence-to-sequence alignment routines to graphs [19, 20, 30, 36]. In our recent work [19], we presented new complexity results and algorithms for the alignment problem using general sequence-labeled graphs. The results show that a sequence (of length m) can be aligned to a labeled directed graph $G(V, E)$ in $O(|V| + m|E|)$ time, using commonly used scoring functions, while allowing edits in the query but not graph labels. However, a general string to graph pattern matching formulation is only good for mapping single-end reads or single-molecule sequencing reads, and does not account for pairing information.

Paired-end sequencing provides information about the relative orientation and genomic distance between the two reads in a pair. When reads originate from repetitive regions, this information is valuable for pruning large number of false candidates [4]. Popular short read mapping tools for linear references, e.g., BWA-mem [24] and Bowtie2 [22], therefore, enforce these constraints in a read pair to guide the selection of the true mapping locus. Using a linear reference, calculating gap between two mapping locations is just a simple subtraction operation. However, it still remains unclear how to efficiently validate the constraints using large non-linear graph-based references and read sets.

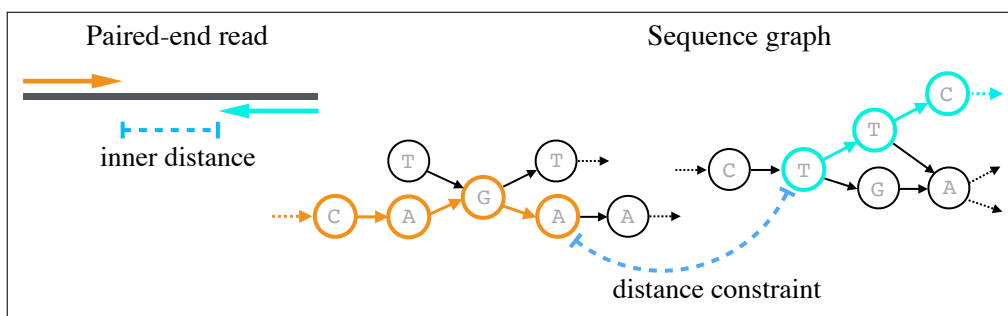
Several sequence to graph aligners have been developed in recent years to map reads to variation graphs [11, 12, 18, 21, 28, 35, 37], de-Bruijn graphs [16, 25, 26] and splicing graphs [1, 8]. Readers are referred to review articles, e.g., [5, 34] for an expanded list of the tools. Among these tools, Graph-Aligner [35], vg [12], deBGA [26], HISAT2 [21] and HLA-PRG [11] support paired-end read mapping. However, all of these use heuristics to measure the observed insert size between the two reads in a pair, mainly due to lack of associated provably-good graph-based algorithms. A popular heuristic adopted by the tools is to do the computation while assuming a linear ordering of vertices (e.g., topological order). However, it can produce misleading results in complex variation-rich graph regions.

In this work, we provide the first mathematical formulation of the problem of validating paired-end distance constraints in sequence graphs, and propose an exact algorithm to solve it that is also practical. The proposed algorithm exploits sparsity in sequence graphs to build an index, which can be queried quickly using a simple lookup during the read mapping process. On the performance side, we provide formal arguments to shed light on why our indexing procedure is efficient with regards to the computation time and storage requirements. We show the practical significance of our algorithm using LRC_KIR and MHC variation graphs derived from the human genome, as well as pan-genomic de Bruijn graphs of *Bacillus anthracis* strains.

2 Problem Formulation

► **Definition 1.** *Sequence Graph:* A sequence graph $G(V, E)$ is a directed graph with vertices V and edges E , where each vertex $v \in V$ is labeled with a character from alphabet Σ .

Sequence graph is typically defined as a directed graph with either string or character labeled vertices because converting one form into the other is straightforward. In addition, commonly used graph formats, such as de-Bruijn graphs, bi-directional de-Bruijn graphs, overlap graphs or variation graphs can be converted into sequence graphs with at most a constant factor increase in vertex or edge set sizes. While single-end read alignments can be judged by their alignment scores alone, a valid paired-end read alignment over a sequence graph should satisfy the expected paired-end distance constraints and orientation. As insert size can vary within a range, let d_1 and d_2 denote the minimum and maximum allowed values of the inner distance between the reads within a pair (see Figure 1).



■ **Figure 1** Visualizing distance constraints while mapping paired-end reads to sequence graphs.

► **Definition 2.** *Paired-end Validation Problem:* Suppose two reads r_1, r_2 in a pair are independently mapped to a sequence graph $G(V, E)$ using their positive and negative strands respectively. Let v_1 be the vertex where a path to which r_1 (+ve strand) is mapped ends, and v_2 be the vertex where a path to which r_2 (-ve strand) is mapped starts; then we refer to this pair of paths as a valid paired-end read mapping if and only if there exists a path from v_1 to v_2 of length $d \in [d_1, d_2]$.

The above problem definition is based on the assumption that a fragment, from which a read pair is sequenced, can align to any (cyclic or acyclic) path in the input sequence graph. In this work, we focus on designing an efficient algorithm that can quickly answer the above path queries for any two given vertices in the graph. Typically, there are multiple mapping candidates to evaluate for each read pair, especially if a read is sequenced from repetitive regions of a genome. In addition, a typical read set in a genomic study may contain millions or billions of reads. Therefore, validating the distance queries quickly using an appropriate indexing scheme is desirable.

3 Related Problems in Graph Theory

Computing all-pairs shortest paths in $G(V, E)$ may help to identify true-positives or true-negatives, but only for those vertex pairs whose shortest distance $\geq d_1$. No conclusion can be drawn when the shortest distance between two vertices is $< d_1$, as a valid path need not be the shortest path. In addition, computing all-pairs shortest paths is expensive, and may not provide the desired scalability [7]. If $d_1 = d_2$, the formulated problem becomes a special case

of the exact-path length problem [33], with all edge weights set to 1. The exact-path length problem determines if a path of a specified distance exists between two vertices in a weighted graph. An extension of this problem, referred to as the gap-filling problem [39], has been explored in the context of genome assembly using paired-end or mate pair read sets. Although the exact-path length problem has been shown to be \mathcal{NP} -complete [33], we will demonstrate a simple and practical polynomial-time algorithm for our problem with unweighted edges. Finally, if $d_1 = 0$ and $d_2 = |V|$, then our problem is equivalent to determining transitive closure of a graph [32]. In our case, however, we expect $d_2 \ll |V|$.

Our approach is based on an indexing strategy where we pre-compute a boolean index matrix, which has a 1 for each vertex pair that satisfies the distance constraints (Section 4). Computing the index requires polynomial operations, and paired-end distance queries can be computed quickly using index lookups during the read mapping process. Before describing the algorithm, we first discuss a trivial pseudo-polynomial time algorithm to solve the paired-end distance validation problem. It is based on a well-known algorithm used to solve the intractable subset-sum problem.

A Pseudo-polynomial Time Algorithm

The problem of validating distance constraints between two vertices can be solved using dynamic programming. Assume $s \in V$ is the source vertex from where we need to query paths of length $d \in [d_1, d_2]$. For a vertex $v \in V$, let $a(v, l)$ be a boolean value which is true if and only if there is a path of length l from source s to v . Then, the following recurrence solves the problem:

$$\begin{aligned} a(v, 0) &= 1 \text{ if } v = s \text{ and } 0 \text{ otherwise,} \\ a(v, l) &= \bigvee_{(u,v) \in E} a(u, l-1) \end{aligned}$$

Solving the above recurrence requires filling a $|V| \times d_2$ table in column-wise order. The distance constraint from the source vertex s to $t \in V$ is satisfied if and only if $a(t, l) = 1$ for any $l \in [d_1, d_2]$. Note that it is sufficient to store two columns in memory to fill the table, and an additional column to track the final result. The algorithm is summarized as a lemma below.

► **Lemma 3.** *There exists an $O(d_2|E|)$ time and $O(|V|)$ space algorithm that decides existence of a path of length $d \in [d_1, d_2]$ from one vertex to another in $G(V, E)$.*

The time complexity of the above algorithm is significantly high, as it requires $O(d_2|E|)$ time to validate distance constraints from a fixed source vertex. With some optimizations however, the above algorithm can be accelerated. As observed by Salmela *et al.* [39] in the context of gap-filling problem, we expect $d_2 \ll |V|$, therefore, it should be possible to compute a sub-graph containing vertices within $\leq d_2/2$ distance from v_1 or v_2 , before solving the recurrence. While this strategy was shown to be effective for gap-filling between assembled contigs, the count of vertex pairs to evaluate during read mapping process is expected to be significantly higher for large read sets. Reference genomes (e.g., GRCh38 for human genome) or graphs are static, or evolve slowly, in genomic analyses. As such, it is desirable to use an index-based strategy, where we pay a one-time cost to build an index, and validate the paired-end distance constraints quickly.

4 An Index-based Polynomial-time Algorithm

In the following, we describe our index construction and querying algorithm. Given a sequence graph in the form of a boolean adjacency matrix, the index construction procedure uses boolean matrix additions and multiplications. As we will note later, the worst-case time of building our index is polynomial in the input size, but still computationally prohibitive to handle real data instances. Subsequently, we will show how to exploit sparsity in graphs to accelerate the computation. The construction algorithm relies on the following boolean matrix operations.

► **Definition 4.** *Boolean matrix operations: Let A and B be two boolean $n \times n$ matrices. The standard boolean matrix operations are evaluated in the following way:*

$$\begin{array}{lll}
 \text{Addition} & C = A \vee B & C_{ij} = A_{ij} \vee B_{ij} \\
 \text{Multiplication} & C = A \cdot B & C_{ij} = \bigvee_{k=1}^n A_{ik} \wedge B_{kj} \\
 \text{Power} & C = A^k & C = \underbrace{A \cdot A \cdot \dots \cdot A}_{k \text{ times}}
 \end{array}$$

The boolean matrix addition and multiplication can also be performed using the standard matrix addition and multiplication, respectively. This is done by adjusting the non-zero values in output matrix to 1. Next, we define index matrix \mathcal{T} , built using the adjacency matrix of the input graph and the distance parameters d_1 and d_2 . Lemma 6 and 7 include its correctness proof and worst-case construction time complexity.

► **Definition 5.** *Let Adj be the $|V| \times |V|$ -sized boolean adjacency matrix associated with graph $G(V, E)$. Define index matrix $\mathcal{T} = Adj^{d_1} \cdot (Adj \vee I)^{d_2 - d_1}$, where I is an identity matrix.*

► **Lemma 6.** $\mathcal{T}[i, j] = 1$ if and only if there exists a path of length $d \in [d_1, d_2]$ from vertex v_i to vertex v_j .

Proof. Note that $Adj^k[i, j] = 1$ if and only if there is a path of length k from vertex v_i to v_j . To validate the paired-end distance constraints, we require $Adj^{d_1} \vee Adj^{d_1+1} \vee \dots \vee Adj^{d_2}$.

$$\begin{aligned}
 \bigvee_{i=d_1}^{d_2} Adj^i &= Adj^{d_1} \cdot \left(\bigvee_{i=0}^{d_2-d_1} Adj^i \right) \\
 &= Adj^{d_1} \cdot (Adj \vee I)^{d_2-d_1} \quad \blacktriangleleft
 \end{aligned}$$

► **Lemma 7.** *If multiplying two square matrices of dimension $|V| \times |V|$ requires $O(|V|^\omega)$ time, $\omega \in \mathbb{R}$, then computing the index matrix requires $O(|V|^\omega \cdot \log(d_2))$ time and $O(|V|^2)$ space.*

Proof. Matrix addition uses $O(|V|^2)$ operations. Computing A^k requires $O(|V|^\omega \log k)$ operations. Therefore, computing the index matrix requires $O(|V|^2 + |V|^\omega(1 + \log d_1 + \log(d_2 - d_1)))$ operations. As $\omega \geq 2$ and $d_2 \geq d_1$, this simplifies to $O(|V|^\omega \cdot \log(d_2))$ time. ◀

The current best algorithm to compute general matrix multiplication requires $O(|V|^{2.37})$ time [23]. Using the general matrix storage format, querying for the distance constraints between a vertex pair requires a simple $O(1)$ lookup. However, general matrix multiplication solvers require at least quadratic time and space (in terms of $|V|$), which does not scale to real graph instances. We next propose an alternate approach to build the index matrix that exploits sparsity in sequence graphs.

4.1 Exploiting Sparsity in Sequence Graphs

Typically, sequence graphs representing variation or assembly graphs have large diameter and high sparsity, with edge to vertex ratio close to 1 [31]. As $d_2 \ll |V|$ in practice, we also expect our final index matrix to be sparse. As a result, we propose using SpGEMM (sparse matrix-matrix multiplication) operations to build the index. Below, we briefly recall the algorithm and matrix storage format used for SpGEMM. Subsequently, we shed light on the construction time and size of the index using this approach. We borrow standard notations typically used to discuss SpGEMM algorithms. Let $nnz(A)$ denote number of non-zero values in matrix A . During boolean matrix multiplication $A \cdot B$, let $bitops(AB)$ indicate the count of non-zero bitwise-AND operations (i.e., $1 \wedge 1$), assuming Definition 4.

4.1.1 Working with Sparse Matrices

Storage. During SpGEMM, the input and output matrices are stored in a sparse format, such that the space is primarily used for non-zeros. Compressed Sparse Row (CSR) is a classic data structure for this purpose [3]. In CSR format, a boolean matrix $A_{n \times n}$ can be represented by using two arrays: the first array ptr of size $n + 1$ contains row pointers, and the second array $cols$ of size $O(nnz(A))$ contains column indices of each non-zero entry in A , starting from the first row to the last. The row pointers are essentially offsets within the second array, such that the range $[cols[ptr[i]], cols[ptr[i + 1]])$ lists column indices in row i . By default, CSR format does not require the indices of a row to be sorted. However, the sorted order will be useful in our index storage to enable fast querying. Therefore, we use “sorted-CSR” format in our application.

► **Remark 8.** Storing a matrix $A_{n \times n}$ in sorted-CSR format requires $\Theta(n + nnz(A))$ space.

► **Remark 9.** Given a sequence graph $G(V, E)$ as an array of edge tuples, transforming its adjacency information into sorted-CSR format takes $\Theta(|V| + |E|)$ time using count sort [15].

Multiplication (SpGEMM). SpGEMM algorithms limit their operation count to just non-zero multiplications and additions required to compute the product, as the remaining entries are guaranteed to be 0. Most of the sequential and high-performance parallel algorithms for SpGEMM, including in MATLAB [13], are based on Gustavson’s algorithm [15]. The algorithm can take input matrices A and B in sorted-CSR format and produce the output matrix $C = A \cdot B$ in the same format. In this algorithm, a row of matrix C , i.e., $C[i, :]$ is computed as a linear combination of the rows κ of B for which $A[i, \kappa] \neq 0$. The complexity result from Gustavson’s work is listed as the following lemma.

► **Lemma 10.** *The time complexity to multiply two sparse matrices $A_{n \times n}$ and $B_{n \times n}$ using Gustavson’s algorithm is $\Theta(n + nnz(A) + bitops(AB))$.*

4.1.2 Indexing Time and Storage Complexity

Computing the index (Definition 5) requires several SpGEMM operations. As such, it is hard to derive a tight bound on the complexity, as runtime and index size depend on non-zero structure of the input sequence graph. However, it is important to get an insight into how the different parameters, e.g., $|V|, d_1, d_2$ may affect them. To address this, we derive a practically useful lower-bound on the complexity.

Consider the chain graph $G'(V', E')$ associated with a longest path in $G(V, E)$, $V' \subset V, E' \subset E$. We claim that the time needed to index $G(V, E)$ is either the same or worse than indexing its chain $G'(V', E')$ (Lemma 11). Subsequently, we compute the time complexity

for indexing the chain using our SpGEMM-based algorithm. The rationale for analyzing the chain graph is (a) non-zero structure of a chain is simple and well-defined for computing time complexity, and (b) sequence graphs are expected to have “near-linear” topology in practice, therefore the derived lower-bounds will be a useful indication of the true costs.

► **Lemma 11.** *The time requirement for indexing a graph $G(V, E)$ using SpGEMM is either the same or higher than indexing the chain graph $G'(V', E')$ associated with its longest path.*

Proof. Note that G' is a sub-graph of G , which will also reflect in their adjacency matrices. The above lemma is based on the following simple observation. Suppose $A, B, A', B', \delta_1, \delta_2$ are boolean square matrices, such that, $A = A' \vee \delta_1$ and $B = B' \vee \delta_2$. Then, using Gustavson’s algorithm, multiplying A and B requires at least as much time as required for multiplying A' and B' . In addition, the product $(A \cdot B)$ is of the form $(A' \cdot B') \vee \delta_3$, where δ_3 is a boolean matrix. For each SpGEMM executed while computing the index, we can use this argument to support the claim. ◀

► **Lemma 12.** *Computing the index for $G(V, E)$ using SpGEMM requires $\Omega(|V'|((d_2 - d_1)^2 + \log d_1))$ time.*

Proof. Let Adj' be the adjacency matrix associated with $G'(V', E')$. To prove the above claim, it is useful to visualize the structure of Adj' (Figure 2). Define a constant $k \ll |V'|$. For simplicity, assume d_1 and $d_2 - d_1$ are powers of 2. Throughout the index computation, the time required by Gustavson’s SpGEMM algorithm is dictated by *bitops*. Following Lemma 10, multiplying Adj'^k to Adj'^k requires $\Theta(|V'|)$ time. In addition, multiplying $(Adj' \vee I)^k$ with $(Adj' \vee I)^k$ requires $\Theta(|V'|k^2)$ time. Therefore, we need $\Theta(|V'| \underbrace{(1 + 1 + \dots + 1)}_{\log d_1 \text{ times}})$ time to

compute Adj'^{d_1} , and $\Theta(|V'| (1 + 2^2 + 4^2 + \dots + (d_2 - d_1)^2))$ time to compute $(Adj' \vee I)^{d_2 - d_1}$. The final multiplication between Adj'^{d_1} and $(Adj' \vee I)^{d_2 - d_1}$ uses $\Theta(|V'| (d_2 - d_1))$ time. All these operations add up to $\Theta(|V'| ((d_2 - d_1)^2 + \log d_1))$ time. This argument, and Lemma 11 suffice to support the claim. ◀

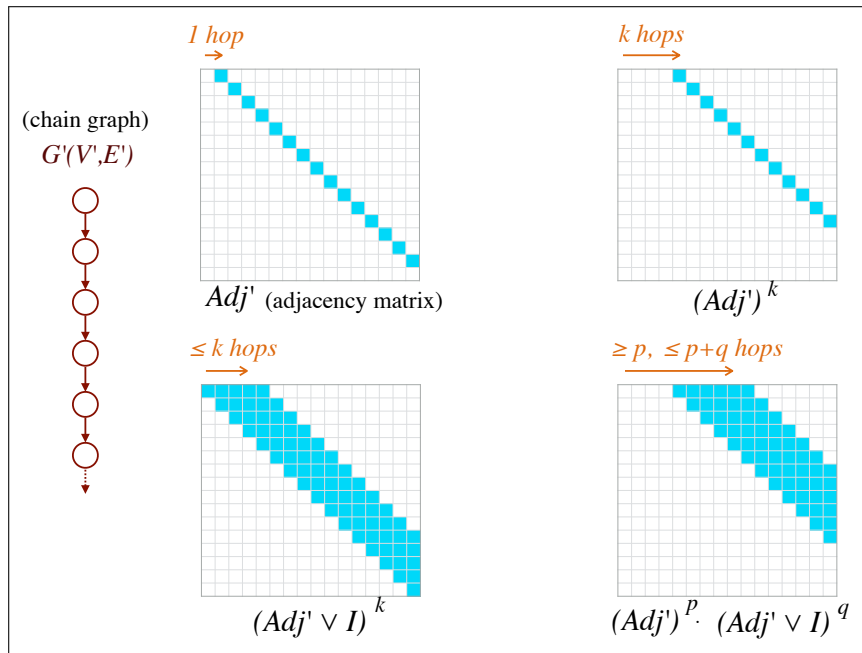
► **Remark 13.** The index size is dictated by count of non-zeros in the final output matrix. Using similar arguments as above, it can be shown that the index storage for graph $G(V, E)$ requires $\Omega(|V'| (d_2 - d_1 + 1))$ space.

4.1.3 Querying the Index

Querying for a value in a sorted-CSR formatted index is trivial. The lookup procedure for $\mathcal{T}[i, j]$ requires a binary search among the non-zeros of row i . Let $\maxRownnz(\mathcal{T})$ be the maximum row size, i.e., maximum number of non-zero entries in a row of index matrix \mathcal{T} . After computing the index, deciding the existence of a path of length $d \in [d_1, d_2]$ between two vertices in $G(V, E)$ requires $O(\log \maxRownnz(\mathcal{T}))$ time.

5 Results

We implemented our algorithm, referred to here as PairG, in C++. The source code is available at <https://github.com/ParBLISS/PairG>. We conducted our evaluation using an Intel Xeon CPU E5-2680 v4, equipped with 28 physical cores and 256 GB main memory. In our implementation, we utilize KokkosKernels [9], an open-source parallel library for basic linear algebra (BLAS) routines. This library does not provide explicit support for boolean matrix operations, so we used integer matrix operations instead, while rounding the non-zero



■ **Figure 2** Visualizing non-zero structure of adjacency matrix of a chain graph. We also show how the structure changes after exponentiation. This is useful to count *bitops* during SpGEMM.

output values to one. We leveraged multi-threading support in KokkosKernels, and allowed it to use 28 threads during execution. Our benchmark data sets, summarized below, consist of cyclic and acyclic graphs built using publicly available real data. We tested indexing and querying performance using PairG for various values of distance constraints. These choices were motivated by the typical insert sizes used for Illumina paired-end sequencing. We demonstrate that PairG can index graphs with more than a million vertices in a reasonable time. Once the index is built, it can answer a million distance constraint queries in a fraction of a second.

5.1 Datasets

We generated seven sequence graphs, four acyclic (G1-G4) and three cyclic (G5-G7) (see Table 1). The first four sequence graphs are variation graphs built using human genome segments (GRCh37) and variant files from the 1000 Genomes Project (Phase 3) [6]. We used vg [12] for building these graphs. The human genomic regions considered in our evaluation are mitochondrial DNA (mtDNA), BRCA1 gene, the killer cell immunoglobulin-like receptors (LRC_KIR), and the major histocompatibility complex (MHC). The sizes of these regions range from 16.6 kilobases (mtDNA) to 5.0 megabases (MHC) in the human genome. De Bruijn Graph (DBG) is another popular format to represent “pan-genome” of a species. DBG is also a good candidate structure to test our algorithm on more complex graphs. We built DBGs with k -mer length 25 using whole-genome sequences of one (G5), five (G6), and twenty (G7) *B. anthracis* strains, using SplitMEM [27]. Strain ids and size of these genomes are listed in the Appendix.

We tested PairG using three different ranges of distance constraints, associated with three insert-size configurations. Assuming a typical sequencing scenario of insert-size configuration as 300 bp and read length 100 bp, the inner distance between paired-end reads should equal

■ **Table 1** Directed sequence graphs used for evaluation. In these graphs, each vertex is labeled with a DNA nucleotide. Four acyclic graphs are derived from segments of human genome and variant files from the 1000 Genomes Project (Phase 3). Three cyclic graphs are de Bruijn graphs built using whole-genome sequences of *Bacillus anthracis* strains, with k -mer length 25.

Id	Graph	# vertices	# edges	Type
G1	mitochondrial-DNA	21,038	26,842	acyclic
G2	BRCA1	83,268	85,422	
G3	LRC_KIR	1,108,769	1,154,046	
G4	MHC	5,138,913	5,318,639	
G5	<i>B. anthracis</i> (1 strain)	5,174,432	5,175,000	cyclic
G6	<i>B. anthracis</i> (5 strains)	10,360,296	10,363,334	
G7	<i>B. anthracis</i> (20 strains)	11,237,067	11,253,437	

100 bp (i.e., insert size minus twice the read length). To allow for sufficient variability, we tested PairG using $d_1 = 0, d_2 = 250$. Similarly, for insert-size configurations of 500 bp and 700 bp, we tested PairG using inner distance limits ($d_1 = 150, d_2 = 450$) and ($d_1 = 350, d_2 = 650$), respectively. There may be insert size configurations where allowing read overlaps may also be necessary; this can be handled trivially by computing a second matrix with desirable limit on the overlap length.

5.2 Index construction

We report wall-clock time, memory-usage, and index size (nnz) using various graphs and distance constraints in Table 2. We note large variation in performance for various graph sizes and complexity. Time, memory, and index size increase almost linearly with increasing graph size. This is also expected based on our theoretical analysis (Section 4.1.2). The specified range of distance constraints $[d_1, d_2]$ can also affect performance. The index size for graphs G1-G5 remains almost uniform for the three different constraints. This should be because the first five graphs have linear chain-like topology, where the performance should be dictated by the gap ($d_2 - d_1$) according to our analysis. The graphs G1-G4 were built by augmenting the variations (substitutions, indels) on the reference sequence, and the fifth graph uses a single strain. On the other hand, index size varies with distance constraints for the last two graphs, especially G7. We expect G7 to have significantly more branching due to complex structural variations and repeats. Another important contributing factor is that DBGs collapse repetitive k -mers into a single vertex, causing large deviation from the linear topology. As a result, it takes time ranging from 1.3 hours to 17 hours for graph G7.

For larger graphs, we expect the index size (nnz) to increase with graph size. In sorted-CSR storage format, we use 4 bytes for each non-zero, which can become prohibitively large at the scale of complete human genome. However, we expect substantial room for compressing the final index, and plan to explore it in the future. Memory-usage of a matrix operation (addition or multiplication) is dictated by the size of the associated input and output matrices. As a result, memory required for the index construction appears to increase proportionally with the index size (Table 2).

5.3 Querying Performance

While indexing is a one-time routine for a sequence graph, we would need to query the index numerous times during the mapping process. Querying for a vertex pair requires a simple and fast lookup in the index (Section 4.1.3). As read mapping locations are expected to

17:10 Validating Paired-End Read Alignments in Sequence Graphs

■ **Table 2** Performance measured in terms of wall-clock time and memory-usage for building index matrix using all input graphs and different distance constraints. *nnz* represents number of non-zero elements in the index matrix, to indicate its size. Our implementation uses 4 bytes to store each non-zero of a matrix in memory.

Id	Distance constraints								
	[0 - 250]			[150 - 450]			[350 - 650]		
	Time	Mem	<i>nnz</i>	Time	Mem	<i>nnz</i>	Time	Mem	<i>nnz</i>
G1	0.2s	0.1G	6.8M	0.4s	0.2G	7.8M	0.4s	0.2G	7.6M
G2	0.4s	0.3G	21M	0.9s	0.5G	26M	0.9s	0.5G	26M
G3	5.6s	3.8G	0.3B	12s	6.2G	0.3B	12s	6.2G	0.3B
G4	25s	17G	1.3B	53s	28G	1.6B	53s	28G	1.6B
G5	25s	17G	1.3B	54s	28G	1.6B	54s	28G	1.7B
G6	52s	35G	2.8B	2m	60G	3.6B	2m	60G	4.2B
G7	1.3h	56G	4.6B	7.2h	118G	8.9B	17.2h	129G	14B

■ **Table 3** Time to execute a million queries using all the graphs and distance constraints. Each query is a random pair of vertices in the graph.

Id	Distance constraints		
	[0 - 250]	[150 - 450]	[350 - 650]
	Time (sec)		
G1	0.1	0.1	0.1
G2	0.2	0.2	0.2
G3	0.4	0.4	0.5
G4	0.5	0.5	0.5
G5	0.4	0.5	0.5
G6	0.4	0.5	0.5
G7	0.5	0.5	0.6

be uniformly distributed over the graph, we tested the querying performance by generating a million random vertex pairs (u, v) , $u, v \in [1, |V|]$. For all the seven graphs, querying a million vertex pairs finished in less than a second (Table 3). Even though the majority of randomly generated queries result in a “no” answer, this aspect has insignificant effect on the query performance. Our results implicate that distance constraints can be validated exactly without additional overhead on the mapping time, which is similar to the case of mapping reads to a reference sequence.

We also compared our index-based algorithm using a breadth first search (BFS)-based heuristic. Using this heuristic, we answer a query of a pair of vertices (u, v) as true if and only if v is reachable within d_2 distance from u . Accordingly, BFS initiated from vertex u is terminated if we find vertex v , or after we have explored all vertices up to depth d_2 . This heuristic does not require a pre-computed index. However, we find that querying time using our index-based algorithm is faster by two to three orders of magnitude. For the randomly generated query set, fraction of results that agree between the two approaches varied from 98% to 100%. The above heuristic yields incorrect result for a vertex pair (u, v) when all the possible paths that connect u to v have length $< d_1$.

6 Conclusions and Future Work

In this work, we formulated the Paired-end Validation Problem, required for paired-end read mapping on sequence graphs. We proposed the first provably good and practically useful exact algorithm for solving this problem. The proposed algorithm builds on top of existing SpGEMM algorithms, to exploit the sparsity and large diameter characteristics of sequence graphs. Our experiments indicate that index construction time is affected by the size and topology of the sequence graph, as well as the desired distance constraints. The querying time is less than a second for answering a million distance queries using all test cases.

There are two immediate avenues for future work. First, we expect huge room for lossless compression in the final index matrix. By computing an appropriate vertex ordering, we can potentially benefit from differential coding of column indices. Thus, alternative formats than sorted-CSR might be more efficient for index storage. We anticipate that good compression ratio can be achieved without affecting querying performance. Second, we plan to integrate this work with an exact sequence to graph aligner as a paired-end read mapping tool, and evaluate the advantages of a provably-good approach relative to existing heuristics-based methods.

References

- 1 Stefano Beretta, Paola Bonizzoni, Luca Denti, Marco Previtali, and Raffaella Rizzi. Mapping RNA-seq data to a transcript graph via approximate pattern matching to a hypertext. In *International Conference on Algorithms for Computational Biology*, pages 49–61. Springer, 2017.
- 2 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 225–235. Springer, 2012.
- 3 Aydın Buluç, John Gilbert, and Viral B Shah. Implementing sparse matrices for graph algorithms. In *Graph Algorithms in the Language of Linear Algebra*, pages 287–313. SIAM, 2011.
- 4 Stefan Canzar and Steven L Salzberg. Short read mapping: An algorithmic tour. *Proceedings of the IEEE*, 105(3):436–458, 2015.
- 5 Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in bioinformatics*, 19(1):118–135, 2016.
- 6 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.
- 7 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
- 8 Luca Denti, Raffaella Rizzi, Stefano Beretta, Gianluca Della Vedova, Marco Previtali, and Paola Bonizzoni. ASGAL: aligning RNA-Seq data to a splicing graph to detect novel alternative splicing events. *BMC bioinformatics*, 19(1):444, 2018.
- 9 Mehmet Deveci, Christian Trott, and Sivasankaran Rajamanickam. Performance-portable sparse matrix-matrix multiplication for many-core architectures. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 693–702. IEEE, 2017.
- 10 Alexander Dilthey, Charles Cox, Zamin Iqbal, Matthew R Nelson, and Gil McVean. Improved genome inference in the MHC using a population reference graph. *Nature genetics*, 47(6):682, 2015.
- 11 Alexander Dilthey, Pierre-Antoine Gourraud, Alexander J Mentzer, Nezh Cereb, Zamin Iqbal, and Gil McVean. High-accuracy HLA type inference from whole-genome sequencing data using population reference graphs. *PLoS computational biology*, 12(10):e1005151, 2016.

- 12 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*, 2018.
- 13 John R Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, 1992.
- 14 Richard E Green, Johannes Krause, Adrian W Briggs, Tomislav Maricic, Udo Stenzel, Martin Kircher, Nick Patterson, Heng Li, Weiwei Zhai, Markus Hsi-Yang Fritz, et al. A draft sequence of the Neandertal genome. *science*, 328(5979):710–722, 2010.
- 15 Fred G Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software (TOMS)*, 4(3):250–269, 1978.
- 16 Mahdi Heydari, Giles Miclotte, Yves Van de Peer, and Jan Fostier. BrownieAligner: accurate alignment of Illumina sequencing data to de Bruijn graphs. *BMC bioinformatics*, 19(1):311, 2018.
- 17 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, 2012.
- 18 Chirag Jain, Sanchit Misra, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru. Accelerating sequence alignment to graphs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019 (to appear).
- 19 Chirag Jain, Haowen Zhang, Yu Gao, and Srinivas Aluru. On the Complexity of Sequence to Graph Alignment. In *Research in Computational Molecular Biology*, pages 85–100, Cham, 2019. Springer International Publishing.
- 20 Vaddadi Naga Sai Kavya, Kshitij Tayal, Rajgopal Srinivasan, and Naveen Sivadasan. Sequence Alignment on Directed Graphs. *Journal of Computational Biology*, 26(1):53–67, 2019.
- 21 Daehwan Kim, Joseph M Paggi, and Steven Salzberg. HISAT-genotype: Next Generation Genomic Analysis Platform on a Personal Computer. *BioRxiv*, page 266197, 2018.
- 22 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357, 2012.
- 23 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014.
- 24 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint*, 2013. [arXiv:1303.3997](https://arxiv.org/abs/1303.3997).
- 25 Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de Bruijn graphs. *BMC bioinformatics*, 17(1):237, 2016.
- 26 Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. deBGA: read alignment with de Bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.
- 27 Shoshana Marcus, Hayan Lee, and Michael C Schatz. SplitMEM: a graphical algorithm for pan-genome analysis with suffix skips. *Bioinformatics*, 30(24):3476–3483, 2014.
- 28 Tom O Mokveld, Jasper Linthorst, Zaid Al-Ars, and Marcel Reinders. CHOP: Haplotype-aware path indexing in population graphs. *bioRxiv*, 2018.
- 29 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.
- 30 Gonzalo Navarro. Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237(1-2):455–463, 2000.
- 31 Adam M Novak, Glenn Hickey, Erik Garrison, Sean Blum, Abram Connelly, Alexander Dilthey, Jordan Eizenga, MA Saleh Elmohamed, Sally Guthrie, André Kahles, et al. Genome graphs. *bioRxiv*, page 101378, 2017.
- 32 Esko Nuutila. *Efficient transitive closure computation in large digraphs*. Finnish Academy of Technology, 1998.
- 33 Matti Nykänen and Esko Ukkonen. The exact path length problem. *Journal of Algorithms*, 42(1):41–53, 2002.

- 34 Benedict Paten, Adam M Novak, Jordan M Eizenga, and Erik Garrison. Genome graphs and the evolution of genome inference. *Genome research*, 27(5):665–676, 2017.
- 35 Goran Rakocevic, Vladimir Semenyuk, Wan-Ping Lee, James Spencer, John Browning, Ivan J Johnson, Vladan Arsenijevic, Jelena Nadj, Kaushik Ghose, Maria C Suci, et al. Fast and accurate genomic analyses using genome graphs. Technical report, Nature Publishing Group, 2019.
- 36 Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in $O(V + mE)$ time. *bioRxiv*, 2017. URL: <https://www.biorxiv.org/content/early/2017/11/08/216127>.
- 37 Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, March 2019. doi:10.1093/bioinformatics/btz162.
- 38 David Reich, Michael A Nalls, WH Linda Kao, Ermeg L Akyzbekova, Arti Tandon, Nick Patterson, James Mullikin, Wen-Chi Hsueh, Ching-Yu Cheng, Josef Coresh, et al. Reduced neutrophil count in people of African descent is due to a regulatory variant in the Duffy antigen receptor for chemokines gene. *PLoS genetics*, 5(1):e1000360, 2009.
- 39 Leena Salmela, Kristoffer Sahlin, Veli Mäkinen, and Alexandru I Tomescu. Gap filling as exact path length problem. *Journal of Computational Biology*, 23(5):347–361, 2016.
- 40 Jouni Sirén. Indexing variation graphs. In *2017 Proceedings of the nineteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 13–27. SIAM, 2017.
- 41 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 (TCBB)*, 11(2):375–388, 2014.

A Appendix: Data Availability

■ **Table 4** List of 20 *Bacillus anthracis* strains used to build the sequence graphs G5-G7. We used the first strain in G5, the first five strains in G6, and all the 20 strains in G7.

Strain id	Assembly id	Size (Mbp)
Ames	GCA_000007845.1	5.23
delta Sterne	GCA_000742695.1	5.23
Cvac02	GCA_000747335.1	5.23
Parent1	GCA_001683095.1	5.23
PR09-1	GCA_001683255.1	5.23
Sterne	GCA_000008165.1	5.23
CNEVA-9066	GCA_000167235.1	5.49
A1055	GCA_000167255.1	5.37
A0174	GCA_000182055.1	5.29
Sen2Col2	GCA_000359425.1	5.17
SVA11	GCA_000583105.1	5.49
95014	GCA_000585275.1	5.34
Smith 1013	GCA_000742315.1	5.29
2000031021	GCA_000742655.1	5.33
PAK-1	GCA_000832425.1	5.40
Pasteur	GCA_000832585.1	5.23
PR06	GCA_001683195.1	5.23
55-VNIIVViM	GCA_001835485.1	5.35
Sterne 34F2	GCA_002005265.1	5.39
UT308	GCA_003345105.1	5.37

Detecting Transcriptomic Structural Variants in Heterogeneous Contexts via the Multiple Compatible Arrangements Problem

Yutong Qiu¹

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA
yutongq@andrew.cmu.edu

Cong Ma¹

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA
congm1@andrew.cmu.edu

Han Xie

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA
hanx@andrew.cmu.edu

Carl Kingsford

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, PA, USA
carlk@cs.cmu.edu

Abstract

Transcriptomic structural variants (TSVs) – large-scale transcriptome sequence change due to structural variation – are common, especially in cancer. Detecting TSVs is a challenging computational problem. Sample heterogeneity (including differences between alleles in diploid organisms) is a critical confounding factor when identifying TSVs. To improve TSV detection in heterogeneous RNA-seq samples, we introduce the MULTIPLE COMPATIBLE ARRANGEMENT PROBLEM (MCAP), which seeks k genome rearrangements to maximize the number of reads that are concordant with at least one rearrangement. This directly models the situation of a heterogeneous or diploid sample. We prove that MCAP is NP-hard and provide a $\frac{1}{4}$ -approximation algorithm for $k = 1$ and a $\frac{3}{4}$ -approximation algorithm for the diploid case ($k = 2$) assuming an oracle for $k = 1$. Combining these, we obtain a $\frac{3}{16}$ -approximation algorithm for MCAP when $k = 2$ (without an oracle). We also present an integer linear programming formulation for general k . We characterize the graph structures that require $k > 1$ to satisfy all edges and show such structures are prevalent in cancer samples. We evaluate our algorithms on 381 TCGA samples and 2 cancer cell lines and show improved performance compared to the state-of-the-art TSV-calling tool, SQUID.

2012 ACM Subject Classification Applied computing → Computational transcriptomics

Keywords and phrases transcriptomic structural variation, integer linear programming, heterogeneity

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.18

Funding This work was supported in part by the Gordon and Betty Moore Foundation’s Data-Driven Discovery Initiative [GBMF4554 to C.K.]; the US National Institutes of Health [R01GM122935]; and The Shurl and Kay Curci Foundation. This project is funded, in part, by a grant (4100070287) from the Pennsylvania Department of Health. The department specifically disclaims responsibility for any analyses, interpretations, or conclusions.

¹ These authors contributed equally to this work.



© Yutong Qiu, Cong Ma, Han Xie, and Carl Kingsford;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 18; pp. 18:1–18:5

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements The results shown here are in part based upon data generated by the TCGA Research Network: <https://www.cancer.gov/tcga>. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. Specifically, it used the Bridges system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC) [17]. C.K. is co-founder of Ocean Genomics, Inc.

1 Introduction

Transcriptomic structural variations (TSVs) are transcriptome sequence alterations due to genomic structural variants (SVs). TSVs may cause the joining of parts from different genes, which are fusion-gene events. Fusion genes are known for their association with various types of cancer. For example, the joint protein products of *BCR-ABL1* genes are prevalently found in leukemia [4]. In addition to fusion genes, the joining of intergenic and genic regions, called non-fusion-gene events, are also related to cancer [22].

TSV events are best studied with RNA-seq data. Although SVs are more often studied with whole genome sequencing (WGS) [2, 12, 18, 9, 5, 20], the models built on WGS data lack the flexibility to describe alternative splicing and differences in expression levels of transcripts affected by TSVs. In addition, RNA-seq data is far more common [14] than WGS data, for example, in The Cancer Genome Atlas (TCGA, <https://cancergenome.nih.gov>).

Many methods have been proposed that identify fusion genes with RNA-seq data. Generally, these tools identify candidates of TSV events through investigation into read alignments that are discordant with the reference genome (e.g. [10, 15, 3, 16, 21, 11]). A read alignment is *concordant* with a reference sequence if the alignment to the sequence agrees with the read library preparation. For example in paired-end Illumina sequencing, the orientation of the forward read should be 5'-to-3' and the reverse for the mate read. Otherwise the alignment is *discordant* with the reference. A series of filtering or scoring functions are applied on each TSV candidate to eliminate the errors in alignment or data preparation. The performance of filters often relies heavily on a large set of method parameters and requires prior annotation [13]. Furthermore, most of the fusion-gene detection methods limit the scope to the joining of protein-coding regions and ignore the joining of intergenic regions that could also affect the transcriptome. An approach that correctly models both fusion-gene and non-fusion-gene events without a large number of ad hoc assumptions is desired.

An intuitive TSV model is the one that describes directly the rearrangement of the genome. For example, when an inversion happens, two double-strand breaks (DSB) are introduced to the genome and the segment between the DSBs is flipped. After a series of TSVs are applied to a genome, a rearranged genome is produced. In order to identify the TSVs, we can attempt to infer the rearranged genome from the original genome and keep track of the arrangements of genome segments. Since a model of the complete genome is produced, both fusion-gene and non-fusion-gene events can be detected. A recently published TSV detection tool, SQUID [14], models TSV events in this way by determining a single rearrangement of a reference genome that can explain the maximum number of observed sequencing reads. SQUID finds one arrangement of genome segments such that a maximum number of reads are able to be sequenced from it. Novel transcriptomic adjacencies appearing in the arrangement are predicted as TSVs while the ones not appearing are regarded as sequencing or alignment errors.

Despite the generally good performance of SQUID, it relies on the assumption that the sample is homogeneous, i.e. the original genome contains only one allele that can be

represented by a single rearranged string. This assumption is unrealistic in diploid (or high ploidy) organisms. When TSV events occur within the same regions on different alleles, read alignments may suggest multiple conflicting ways of placing a segment. Under the homogeneous assumption, conflicting TSV candidates are regarded as errors. Therefore, this assumption leads to discarding the conflicting TSV candidates that would be compatible on separate alleles and therefore limits the discovery of true TSVs. Conflicting SV candidates are addressed in a few SV detection tools such as VariationHunter-CR [9]. However, VariationHunter-CR assumes a diploid genome, and its model is built for WGS data that lacks ability to handle RNA-seq data.

We present an improved model of TSV events in heterogeneous contexts. We address the limitation of the homogeneous assumption by extending the assumption to k alleles. We introduce the MULTIPLE COMPATIBLE ARRANGEMENT PROBLEM (MCAP), which seeks, assuming the number of alleles k is known, an optimal set of k arrangements of segments from GSG such that the number of concordant sequencing reads with any of the rearrangement sequences is maximized. Each arrangement is a permutation and reorientation of all segments from the reference genome, representing the altered sequence of one allele. The originally discordant edges that are concordant in any of the k arrangements are predicted as TSVs, and those edges are regarded as errors otherwise. We show that MCAP is NP-hard. To address NP-hardness, we propose a $\frac{1}{4}$ -approximation algorithm for the $k = 1$ case and a $\frac{3}{4}$ -approximation solution to the $k = 2$ case using an oracle for $k = 1$. Combining these, we obtain a $\frac{3}{16}$ -approximation algorithm for MCAP when $k = 2$ (without an oracle). We also present an integer linear programming (ILP) formulation that gives an optimal solution for general k .

We characterize the patterns of reads that result in conflicting TSV candidates under a single-allele assumption. We show that these patterns are prevalent in both cancer cell lines and TCGA samples, thereby further motivating the importance of SV detection approaches that directly model heterogeneity.

We apply our algorithms to 381 TCGA samples from 4 cancer types and show that many more TSVs can be identified under a diploid assumption compared to a haploid assumption. We also evaluate an exact ILP formulation under a diploid assumption (D-SQUID) on previously annotated cancer cell lines HCC1395 and HCC1954, identifying several previously known and novel TSVs. We also show that, in most of the TCGA samples, the performance of the approximation algorithm is very close to optimal and the worst case of $\frac{3}{16}$ -approximation is rare.

2 The Genome Segment Graph (GSG)

A Genome Segment Graph, similar to a splice graph [8], encodes relationships between genomic segments and a set of reads. A *segmentation* S of the genome is a partition of the genome into disjoint intervals according to concordant and discordant paired-end alignments with respect to the reference genome. The genome partitioning, edge construction and edge filtering is done in the same way as in [14].

► **Definition 1** (Genome Segment Graph). *A genome segment graph is a weighted, undirected graph $G = (V, E, \mathbf{w})$ derived from a segmentation S of the genome and a collection of reads. The vertex set, $V = \{s_h \in S\} \cup \{s_t \in S\}$, includes a vertex for both endpoints, head (h) and tail (t), for each segment $s \in S$. The head of a segment is the end that is closer to the 5' end of the genome. The tail is the end that is close to the 3' end. Pairs of reads that span more than one segment are represented by edges. There are four types of connections: head-head, head-tail, tail-head and tail-tail. Each edge $e = (u_i, v_j) \in E$, where $i, j \in \{h, t\}$, is undirected*

18:4 Detecting TSVs in Heterogeneous Contexts via MCAP

and connects endpoints of two segments. The weight ($w_e \in \mathbf{w}$) is the number of sequencing reads that support edge e .

We also define the weight of a subset $E' \subseteq E$ of edges $w(E') = \sum_{e \in E'} w_e$. (More details on the GSG provided in Ma et al. [14].)

► **Definition 2** (Permutation, Orientation function and Arrangement). *A permutation is a function where $\pi(u) = i$, where i is the index of segment $u \in S$ in an ordering of a set S of segments. We also define orientation function $f(u) = 1$ if segment u should remain the original orientation, or 0 if it should be inverted. An arrangement is a pair of permutation and orientation functions (π, f) .*

If $\pi(u) < \pi(v)$, we say that segment u is closer to the 5' end of the rearranged genome than segment v . Each arrangement is a concatenation of segments from different chromosomes, which retrieves the sequences affected by inter- and intra-chromosomal TSV events. The arrangement of genome segments imitates the movements of genomic sequences by SVs. One crucial difference between arrangement in GSG and sequence movements by SVs is that an arrangement in GSG only captures the movement that are relevant to transcriptome sequence alterations. Such alterations can either fuse two transcript sequences or incorporate previously non-transcribing sequences into transcripts as long as they are present in RNA-seq reads.

► **Definition 3** (Concordant and Discordant edges). *Let e be an edge connecting segment u on end a and segment v on end b ($a, b \in \{h, t\}$). Given arrangement (π, f) , suppose $\pi(u) < \pi(v)$, edge e is concordant with respect to the arrangement if $f(u) = \mathbb{1}_{a=t}$ and $f(v) = \mathbb{1}_{b=h}$. Denote the concordance as $e \sim (\pi, f)$. Otherwise, e is discordant and denote as $e \not\sim (\pi, f)$.*

Since edges are constructed based on segment connections indicated by read alignments, the concordance and discordance of edges are extensions from read alignments. A discordant edge represents a set of discordant read alignments. Examples of discordant edges with tail-tail and head-head connections are shown in Figure 1a. Concordant edges, when connecting nodes that belong to the same chromosome, represent concordant alignments that are either continuous alignments or split-alignments due to alternative splicing. Due to alternative splicing, a node can be incident to multiple concordant edges given an arrangement. Edges that initially spanned two chromosomes but become concordant in an arrangement represent inter-chromosomal translocation events.

Segments connected by discordant edges can be arranged so that some of the discordant edges become concordant. See Figure 1b,c for examples of arrangements that make tail-tail and head-head connections concordant.

► **Definition 4** (Conflicts among a Set of Edges). *Given GSG $G = (V, E, \mathbf{w})$ and a subset of edges E' , the edges in set E' are in conflict with each other if there is no single arrangement (π, f) such that $e \sim (\pi, f)$ ($\forall e \in E'$). Otherwise, edges in set E' are compatible with each other.*

► **Definition 5** (Transcriptomic Structural Variant (TSV)). *A TSV is a new adjacency in transcript sequences that cannot be explained by alternative splicing.*

In GSG, the adjacency in transcript sequences is represented by edges. New adjacencies that cannot be explained by alternative splicing belong to the set of edges that are either discordant with respect to the reference arrangement or connecting segments belonging to different chromosomes.

3 The Multiple Compatible Arrangements Problem (MCAP)

3.1 Problem Statement

Given an input GSG $G = (V, E, w)$ and a positive integer k , the MULTIPLE COMPATIBLE ARRANGEMENTS PROBLEM seeks a set of k arrangements $A = \{(\pi_i, f_i)\}_{i=1}^k$ that are able to generate the maximum number of sequencing reads:

$$\max_A \sum_{e \in E} w(e) \cdot \mathbf{1}[e \sim A], \quad (1)$$

where $\mathbf{1}[e \sim A]$ is 1 if edge e is concordant in at least one $(\pi_i, f_i) \in A$, and 0 otherwise.

This objective function aims to find an optimal set of k arrangements of segments where the sum of concordant edge weights is maximized in the rearranged alleles, where k is the number of alleles and assumed to be known for sure. The objective seeks to maximize the agreement between rearranged allelic sequences and observed RNA-seq data. Assuming that the majority of RNA-seq reads are sequenced, the concordant edges with respect to the optimal set of arrangements represent the most confident transcriptomic adjacencies. In heterogeneous samples where $k \neq 1$, MCAP separates the conflicting edges onto k alleles as shown in an example in Figure 1.

When $k = 1$, the problem reduces to finding a single rearranged genome to maximize the number of concordant reads, which is the problem that SQUID [14] solves. We refer to the special case when $k = 1$ as SINGLE COMPATIBLE ARRANGEMENT PROBLEM (SCAP).

Predicted TSVs are the concordant edges with respect to any of the arrangements in a solution to MCAP that were either discordant with respect to or spanning multiple chromosomes in the reference genome.

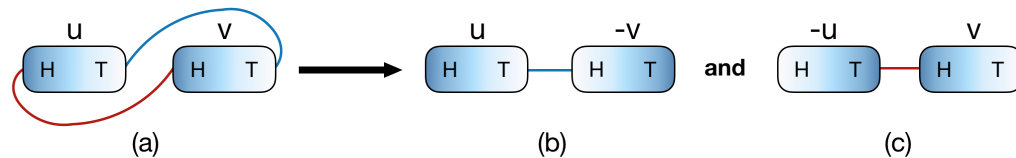


Figure 1 MCAP resolves conflicts. The darker ends of the segments represent head with respect to the original genome. The lighter ends represent tail with respect to the original genome. “H” stands for head and “T” stands for tail. (a) Two conflicting edges connecting two segments u and v . If the sample is known to be homogeneous ($k = 1$), then the conflict is due to errors. If $k = 2$, MCAP seeks to separate two edges into two compatible arrangements as in (b) and (c). (b) In the first arrangement, segment v is flipped, which makes the blue edge concordant. (c) In the second arrangement, u is flipped to make the red edge concordant.

3.2 NP-hardness of SCAP and MCAP

► **Theorem 6.** *SCAP is NP-hard.*

Proof Sketch. We prove the NP-hardness of SCAP by reducing from MAX-2-SAT. While 2-SAT can be solved in polynomial time, MAX-2-SAT, which asks for the maximum number of clauses that can be satisfied, is NP-hard. For boolean variables and clauses in any MAX-2-SAT instance, we create gadget segments in the GSG so that the satisfaction of each clause is determined by the edge concordance and the boolean assignment is determined by segment

inversion. The gadgets force the optimal sum of concordant edge weights to directly represent the number of satisfied clauses. Correspondingly, the optimal orientations of segments represent the assignment of boolean variables. See Appendix A for a complete proof. ◀

► **Corollary 7.** *MCAP is NP-hard.*

Proof. SCAP is a special case of MCAP with $k = 1$, so the NP-hardness of MCAP is immediate. ◀

4 A $\frac{1}{4}$ -approximation Algorithm for SCAP

We provide a greedy algorithm for SCAP that achieves at least $\frac{1}{4}$ approximation ratio and takes $O(|V||E|)$ time. The main idea of the greedy algorithm is to place each segment into the current order one by one by choosing the current “best” position. The current “best” position is determined by the concordant edge weights between the segment to be placed and the segments already in the current order.

■ **Algorithm 1** Greedy algorithm for SCAP.

Data: Segment set S , genome segment graph $G = (V, E, w)$
Result: An arrangement of the segments and the sum of concordant edge weights

```

1  $order = [];$ 
2  $orientation = [];$ 
3 for  $i$  in  $1 : |S|$  do
4    $s^i =$  the  $i^{th}$  segment in  $S$ ;
   // choose from 4 possible order and orientation options
5    $options = [(s^i$  in the beginning of  $order$  in forward strand), ( $s^i$  in the beginning
   of  $order$  in reverse strand), ( $s^i$  in the end of  $order$  in forward strand), ( $s^i$  in the
   end of  $order$  in reverse strand)];
6   for  $j$  in  $1 : 4$  do
7      $weights[j] =$ 
        $w(\{e \in E : e \text{ connects } s^i \text{ with } s^k \text{ and concordant in } options[j], k < i\});$ 
8   end
   // update the current order and orientation
9    $opt = \operatorname{argmax}_{1 \leq i \leq 4, i \in \mathbb{N}} weights[i];$ 
10   $order =$  update segment order as given by  $options[opt];$ 
11   $orientation =$  update segment orientation as given by  $options[opt];$ 
12 end

```

► **Theorem 8.** *Algorithm 1 approximates SCAP with at least $\frac{1}{4}$ approximation ratio.*

Proof. Denote $E' \subset E$ as the concordant edges in the arrangement of Algorithm 1. Let OPT be the optimal value of SCAP. We are to prove $w(E') \geq \frac{1}{4}w(E) \geq \frac{1}{4}OPT$.

For iteration i in the for loop, the edges $E_i = \{e \in E : e \text{ connects } s^i \text{ with } s^j, i < j\}$ are considered when comparing the options. Each of the four options makes a subset of E_i concordant. These subsets are non-overlapping and their union is E_i . Specifically, the concordant edge subset is $\{e = (s_h^i, s_t^j)\}$ for the first option, $\{e = (s_h^i, s_h^j)\}$ for the second, $\{e = (s_t^i, s_h^j)\}$ for the third, and $\{e = (s_t^i, s_t^j)\}$ for the last.

By the selecting the option with the largest sum of concordant edge weights, the concordant edges E'_i in iteration i satisfies $w(E'_i) \geq \frac{1}{4}w(E_i)$. Therefore, the overall concordant edge weights of all iterations in the for loop satisfy

$$\sum_i w(E'_i) \geq \frac{1}{4} \sum_i w(E_i) = \frac{1}{4} w \left(\bigcup_i E_i \right).$$

Each edge $e \in E$ must appear in one and only one of E_i , and thus $\bigcup_i E_i = E$. This implies $\sum_i w(E'_i) \geq \frac{1}{4}w(E) \geq \frac{1}{4}OPT$. ◀

Algorithm 1 can be further improved in practice by considering more order and orientation options when inserting a segment into current order. In the pseudo-code 1, only two possible insertion places are considered: the beginning and the end of the current order. However, a new segment can be inserted in between any pair of adjacent segments in the current order. We provide an extended greedy algorithm to take into account the extra possible inserting positions (Algorithm 2). Algorithm 2 has a time complexity of $O(|V|^2|E|)$, but it may achieve a higher total concordant edge weight in practice.

■ **Algorithm 2** Extended greedy algorithm for SCAP.

Data: Segment set S , genome segment graph $G = (V, E, w)$

Result: An arrangement of the segments and the sum of concordant edge weights

```

1 order = [];
2 orientation = [];
3 for  $i$  in  $1 : |S|$  do
4    $s^i$  = the  $i^{th}$  segment in  $S$ ;
   // choose from  $i + 1$  possible order and orientation options
5   options = [( $s^i$  in the beginning of order in forward strand), ( $s^i$  in the beginning
   of order in reverse strand)];
6   for  $j$  in  $1 : i - 1$  do
7     Append [( $s^i$  right after order[ $j$ ] in forward strand), ( $s^i$  right after order[ $j$ ] in
     reverse strand)] to list of options;
8   end
9   for  $j$  in  $1 : 2i$  do
10    weights[ $j$ ] =
     $w(\{e \in E : e \text{ connects } s^i \text{ with } s^k \text{ and concordant in } options[j], k < i\})$ ;
11  end
   // update the current order and orientation
12  opt =  $\operatorname{argmax}_{1 \leq j \leq i, j \in \mathbb{N}} weights[j]$ ;
13  order = update segment order as given by options[opt];
14  orientation = update segment orientation as given by options[opt];
15 end

```

5 A $\frac{3}{4}$ -approximation of MCAP with $k = 2$ Using a SCAP Oracle

If an optimal SCAP solution can be computed, one way to approximate the MCAP's optimal solution is to solve a series of SCAP instances iteratively to obtain multiple arrangements. Here, we prove the iterative SCAP solution has an approximation ratio of $\frac{3}{4}$ for the special case of MCAP with $k = 2$.

■ **Algorithm 3** $\frac{3}{4}$ -approximation for MCAP with $k = 2$.

Data: A genome segment graph $G = (V, E, w)$

Result: a set of two arrangements, sum of weights of edges that are concordant in either arrangement

- 1 $a_1 =$ optimal SCAP arrangement on G ;
 - 2 $E' = \{e \in E : e \text{ is discordant in } a_1\}$;
 - 3 $G' = (V, E', w)$;
 - 4 $a_2 =$ optimal SCAP arrangement on G' ;
 - 5 $\tilde{E} = \{e \in E : e \sim A, A = \{a_1, a_2\}\}$;
 - 6 $W = \sum_{e \in \tilde{E}} w(e)$;
 - 7 **return** $(\{a_1, a_2\}, W)$;
-

► **Theorem 9.** *Algorithm 3 is a $\frac{3}{4}$ -approximation of MCAP with $k = 2$. Denote the optimal objective sum of edge weights in MCAP with $k = 2$ as OPT , and the sum of edge weights in iterative SCAP as W , then*

$$W \geq \frac{3}{4}OPT$$

Proof. Denote MCAP with $k = 2$ as 2-MCAP. Let E_1^d and E_2^d be concordant edges in the optimal two arrangements of 2-MCAP. It is always possible to make the concordant edges of the arrangements disjoint by removing the intersection from one of the concordant edge set, that is $E_1^d \cap E_2^d = \emptyset$. Let $E^d = E_1^d \cup E_2^d$. The optimal value is $w(E^d)$.

Denote the optimal set of concordant edges in the first round of Algorithm 3 as E_1^s . The optimal value of SCAP is $w(E_1^s)$. E_1^s can have overlap with the two concordant edge sets of the 2-MCAP optimal solution. Let the intersections be $I_1 = E_1^d \cap E_1^s$ and $I_2 = E_2^d \cap E_1^s$. Let the unique concordant edges be $D_1 = E_1^d - E_1^s$, $D_2 = E_2^d - E_1^s$ and $S = E_1^s - E_1^d - E_2^d$.

After separating the concordant edges in 2-MCAP into the intersections and unique sets, the optimal value of 2-MCAP can be written as $w(E^d) = w(I_1) + w(I_2) + w(D_1) + w(D_2)$, where the four subsets are disjoint. Therefore the smallest weight among the four subsets must be no greater than $\frac{1}{4}w(E^d)$. We prove the approximation ratio under the following two cases and discuss the weight of the second round of SCAP separately:

Case (1): the weight of either D_1 or D_2 is smaller than $\frac{1}{4}w(E^d)$. Because the two arrangements in 2-MCAP are interchangeable, we only prove for the case where $w(D_1) \leq \frac{1}{4}w(E^d)$. A valid arrangement of the second round of SCAP is the second arrangement in 2-MCAP, though it may not be optimal. The maximum concordant edge weights added by the second round of SCAP must be no smaller than $w(D_2)$. Combining the optimal values of two rounds of SCAP, the concordant edge weight is

$$W \geq w(E_1^s) + w(D_2) = w(S) + w(I_1) + w(I_2) + w(D_2) \geq w(E^d) - w(D_1) \geq \frac{3}{4}w(E^d). \quad (2)$$

Case (2): both $w(D_1) \geq \frac{1}{4}w(E^d)$ and $w(D_2) \geq \frac{1}{4}w(E^d)$. The subset with smallest sum of edge weights is now either I_1 or I_2 . Without loss of generality, we assume I_1 has the smallest sum of edge weights and $w(I_1) \leq \frac{1}{4}w(E^d)$. Because the first round SCAP is optimal for the SCAP problem, its objective value should be no smaller than the concordant edge weights of either arrangement in 2-MCAP. Thus

$$w(E_1^s) \geq w(E_2^d) = w(D_2) + w(I_2). \quad (3)$$

A valid arrangement for the second round of SCAP can be either of the arrangements in 2-MCAP optimal solution. Picking the first arrangement of 2-MCAP as the possible (but not necessarily optimal) arrangement for the second round of SCAP, the concordant edge weights added by the second round of SCAP must be no smaller than $w(D_1)$. Therefore, the total sum of concordant edge weights of the optimal solutions of both rounds of SCAP is

$$W \geq w(E_1^s) + w(D_1) \geq w(D_2) + w(I_2) + w(D_1) = w(E^d) - w(I_1) \geq \frac{3}{4}w(E^d). \quad (4)$$

◀

► **Corollary 10.** *An approximation algorithm for MCAP with $k = 2$ can be created by using Algorithm 1 as the oracle for SCAP in Algorithm 3. This approximation algorithm runs in $O(|V||E|)$ time and achieves at least $\frac{3}{16}$ approximation ratio.*

The proof of the corollary is similar to the proof of iterative SCAP approximation ratio. By adding a multiplier of $\frac{1}{4}$ to the right of inequalities (3) and (4), the $\frac{3}{16}$ approximation ratio can be derived accordingly.

6 Integer Linear Programming Formulation for MCAP

MCAP, for general k , can be formulated as an integer linear programming (ILP) to obtain an optimal solution. We rewrite the i^{th} permutation (π_i) , orientation (f_i) and decision $(\mathbf{1}[e \sim (\pi_i, f_i)])$ functions with three boolean variables y_e^i , z_{uv}^i and x_e^i . For $i \in \{1, 2, \dots, k\}$ and $e \in E$, we have:

- $x_e^i = 1$ if edge $e \sim (\pi_i, f_i)$ and 0 otherwise.
- $y_u^i = 1$ if $f_i(u) = 1$ for segment u and 0 if $f_i(u) = 0$.
- $z_{uv}^i = 1$ if $\pi_i(u) < \pi_i(v)$, or segment u is in front of v in arrangement i and 0 otherwise.

In order to account for the edges that are concordant in more than one arrangement in the summation in Equation 1, we define q_e such that $q_e = 1$ if edge e is concordant in one of the k arrangements and 0 otherwise. The constraints for q_e are as follows:

$$q_e \leq \sum_i^k x_e^i \quad (5)$$

$$q_e \leq 1 \quad (6)$$

The objective function becomes

$$\max_{x_e^i, y_u^i, z_{uv}^i} \sum_{e \in E} w(e) \cdot q_e \quad (7)$$

We then add ordering and orientation constraints. If an edge is a tail-head connection, i.e. concordant to the reference genome, $x_e^i = 1$ if and only if $z_{uv}^i = y_u^i = y_v^i$. If an edge is a tail-tail connection, $x_e^i = 1$ if and only if $z_{uv}^i = 1 - y_v^i = y_u^i$. If an edge is a head-tail connection, $x_e^i = 1$ if and only if $z_{uv}^i = 1 - y_u^i = 1 - y_v^i$. If an edge is a head-head connection, $x_e^i = 1$ if and only if $z_{uv}^i = 1 - y_u^i = y_v^i$. The constraints for a tail-head connection are listed below in Equation 8, which enforce the assignment of boolean variables y_e^i , z_e^i and x_e^i :

$$\begin{aligned} x_e^i &\leq y_u^i - y_v^i + 1, \\ x_e^i &\leq y_v^i - y_u^i + 1, \\ x_e^i &\leq y_u^i - z_{uv}^i + 1, \\ x_e^i &\leq z_{uv}^i - y_u^i + 1, \end{aligned} \quad (8)$$

The constraints of other types of connections are similar and detailed in Ma et al. [14]. Additionally, constraints are added so that all segments are put into a total order within each allele. For two segments u, v , segment u will be either precede or follow segment v , i.e. $z_{uv}^i + z_{vu}^i = 1$. For three segments u, v, w , if u precedes v and v precedes w , then u has to precede w : $1 \leq z_{uv}^i + z_{vw}^i + z_{wu}^i \leq 2$.

The total number of constraints as a function of k is $4k|E| + k\binom{|V|}{3} + 2|E| = O(k(|E| + V^3))$. When k increases, the number of constraints grows linearly. When $k = 1$, the ILP formulation reduces to the same formulation as SQUID.

7 Characterizing the Conflict Structures That Imply Heterogeneity

In this section, we ignore edge weights and characterize the graph structures where homogeneous assumption cannot explain all edges. We add a set of segment edges, \hat{E} , to the GSG. Each $\hat{e} \in \hat{E}$ connects the two endpoints of each segment, i.e. $\hat{e} = \{s_h, s_t\}$ for $s \in S$. The representation of GSG becomes $G = (E, \hat{E}, V)$.

► **Definition 11** (Conflict Structures and Compatible Structures). A conflict structure, $CS = (E', \hat{E}', V')$, is a subgraph of a GSG where there exists a set of edges E' that cannot be made concordant using any single arrangement. A compatible structure is a subgraph of a GSG where there exists a single arrangement such that all edges can be made concordant in it.

► **Definition 12** (Simple cycle in GSG). A simple cycle, $C = (E', \hat{E}', \{v_0, \dots, v_{n-1}\})$, is a subgraph of a GSG, such that $E' \subseteq E, \hat{E}' \subseteq \hat{E}$ and $v_i \in V$, with $(v_i, v_{i+1 \bmod n}) \in E' \cup \hat{E}'$ and where $v_i \neq v_j$ when $i \neq j$ except $v_{n-1} = v_0$.

► **Definition 13** (Degree and special degree of a vertex in subgraphs of GSG). Given a subgraph of GSG, $G' = (E', \hat{E}', V')$, $deg_{E'}(v)$ refers to the degree of vertex $v \in V'$ that counts only the edges $e \in E'$ that connect to v . $deg(v)$ refers to the number of edges $e \in E' \cup \hat{E}'$ that connect to v .

► **Theorem 14**. Any acyclic subgraph of GSG is a compatible structure.

► **Theorem 15**. A simple cycle $C = (E', \hat{E}', V')$ is a compatible structure if and only if there are exactly two vertices, v_j and v_i such that $deg_{E'}(v_i) = deg_{E'}(v_j) = 2$ and v_i and v_j belongs to different segments.

The details of the proofs of the above two theorems are in Appendix B.

► **Corollary 16**. A necessary condition for a subgraph (E', \hat{E}', V') to be a conflict structure is that it contains cycles. A sufficient condition for a subgraph (E', \hat{E}', V') to be a conflict structure is that it contains a simple cycle which is not a compatible structure.

The corollary is a direct derivation of Theorem 14 and Theorem 15 when considering general graph structures.

In practice, we determine if a discordant edge, $e = (u, v)$, is involved in a conflict structure by enumerating all simple paths using a modified depth-first search implemented in Networkx [7, 19] between u and v omitting edge e . We add e to each path and form a simple cycle. If the simple cycle satisfies Corollary 16, we stop path enumeration and label the e as discordant edge involved in conflict structure. If the running time of path enumeration exceeds 0.5 seconds, we shuffle the order of DFS and repeat enumeration. If path enumeration for e exceeds 1000 reruns, we label e as undecided.

8 Experimental Results

To produce an efficient, practical algorithm for TSV detection in diploid organisms, we use the following approach, which we denote as D-SQUID: Run the ILP (Section 6) under the diploid assumption by setting $k = 2$ on every connected component of GSG separately. If the ILP finishes or the running time of the ILP exceeds one hour, output the current arrangements.

8.1 D-SQUID Identifies More TSVs in TCGA Samples than SQUID

We calculate the fraction of discordant edges involved in conflict structures (Figure 2a) in 381 TCGA samples from four types of cancers: bladder urothelial carcinoma (BLCA), breast invasive carcinoma (BRCA), lung adenocarcinoma (LUAD) and prostate adenocarcinoma (PRAD). Among all samples, we found less than 0.5% undecided edges out of all discordant edges. The distribution of fraction of discordant edges within conflict structures are different among cancer types. The more discordant edges are involved in conflict structures, the more heterogeneous the sample is. Among four cancer types, PRAD samples exhibit the highest extent of heterogeneity and BRCA samples exhibit the lowest. On average, more than 90% of discordant edges are within conflict structures in all samples across four cancer types. This suggests that TCGA samples are usually heterogeneous and may be partially explained by the fact that TCGA samples are usually a mixture of tumor cells and normal cells [1].

We compare the number of TSVs found by D-SQUID and SQUID (Figure 2b). In all of our results, all of the TSVs found by SQUID belong to a subset of TSVs found by D-SQUID. D-SQUID identifies many more TSVs than SQUID on all four types of cancers.

A discordant edge is termed resolved if it is made concordant in one of the arrangements. Among all discordant edges in all samples, D-SQUID is able to resolve most of them (Figure 2c), while SQUID is only able to resolve fewer than 50% of them. The results demonstrate that D-SQUID is more capable of resolving conflict structures in heterogeneous contexts, such as cancer samples, than SQUID.

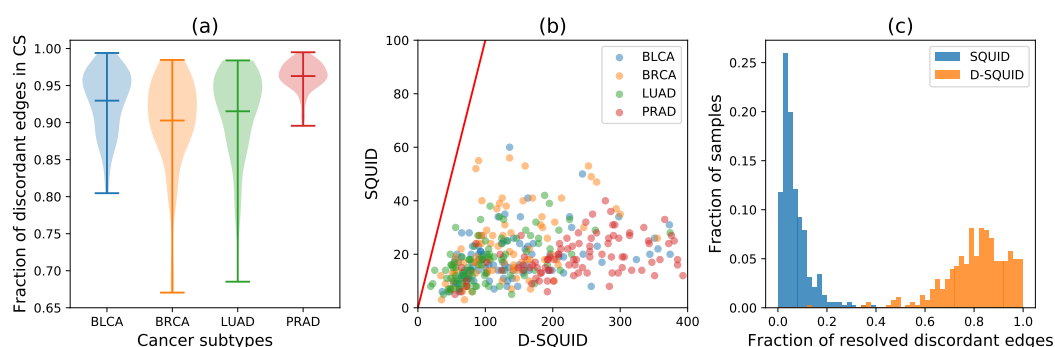
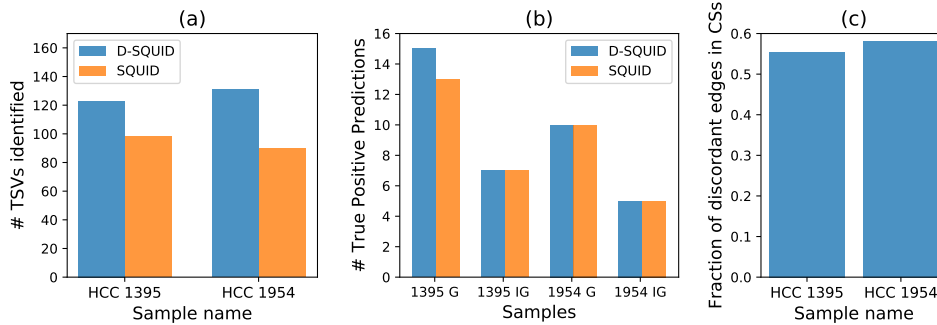


Figure 2 (a) The distribution of fractions of discordant edges that are involved in each identified conflict structure (CS) in four cancer subtypes. Minima, maxima and means of the distributions are marked by horizontal bars. (b) Number of TSVs identified by SQUID versus D-SQUID. (c) Histogram of fractions of resolved discordant edges by SQUID and D-SQUID.

8.2 D-SQUID Identifies More True TSV Events Than SQUID in Cancer Cell Lines

We compare the ability of D-SQUID and SQUID to detect fusion-gene and non-fusion-gene events on previously studied breast cancer cell lines HCC1395 and HCC1954 [6]. The annotation of true SVs is taken from Ma et al. [14]. In both cell lines, D-SQUID discovers more TSVs than SQUID. In HCC1954, D-SQUID identifies the same number of known TSVs including fusions of gene (G) regions and intergenic (IG) regions compared with SQUID. In HCC1395, D-SQUID identifies 2 more true TSV events that are fusions of genic regions. We tally the fraction of discordant edges in conflict structures (Figure 3c) and find similar fractions between HCC1395 and HCC1954, which indicates that the extent of heterogeneity in two samples are similar. Compared to Figure 2a, the fraction in HCC samples is much lower than that in TCGA samples. This matches the fact that two HCC samples contain the same cell type and are both cell line samples, which are known to be less heterogeneous than TCGA samples.



■ **Figure 3** Performance of D-SQUID and SQUID on breast cancer cell lines with experimentally verified SV. (a) Total TSVs found. In both cell line samples, D-SQUID discovered more TSVs than SQUID. (b) Number of known fusion-gene and non-fusion-gene events recovered by D-SQUID and SQUID. G denotes TSVs that affect gene regions. IG denotes TSVs that affect intergenic regions. (c) Fraction of discordant edges in conflict structures.

8.3 Evaluation of approximation algorithms

We evaluate the approximation algorithms for diploid MCAP ($k = 2$) using two different subroutines described in Section 4. In this subsection, $A1$ refers to using Algorithm 1 with worst case runtime $O(|V||E|)$ as a subroutine and $A2$ refers to using Algorithm 2 with worst case runtime $O(|V|^2|E|)$ as a subroutine. Both $A1$ and $A2$ solve SCAP by greedily inserting segments into the best position in the current ordering. While $A1$ only looks at the beginning and ending of the ordering, $A2$ looks at all the positions.

In order to compare the performance of approximations to the exact algorithm using ILP, we run D-SQUID, $A1$ and $A2$ on TCGA samples in Section 8.1. The algorithms are evaluated on runtime and total weight of concordant edges in the rearranged genomes. “Fold difference” on the axes of Figure 4 refers to the ratio of the axis values of D-SQUID over that of $A1$ or $A2$. Both $A1$ and $A2$ output results in a much shorter period of time than D-SQUID. $A2$ achieves better approximation than $A1$, demonstrated by closer-to-one ratio of total concordant edge weight, at a cost of longer run time.

The run time of D-SQUID ILP exceeds one hour on 4.5% of all connected components in all TCGA samples. D-SQUID outputs sub-optimal arrangements in such cases. As a result, approximation algorithms, especially A2, appear to resolve more high-weight discordant edges than D-SQUID in some of the samples in Figure 4, which is demonstrated by data points that fall below 1 on the y axes. A1 resolves more high-weight edges in 10 samples and A2 resolves more high-weight edges in 54 samples than D-SQUID.

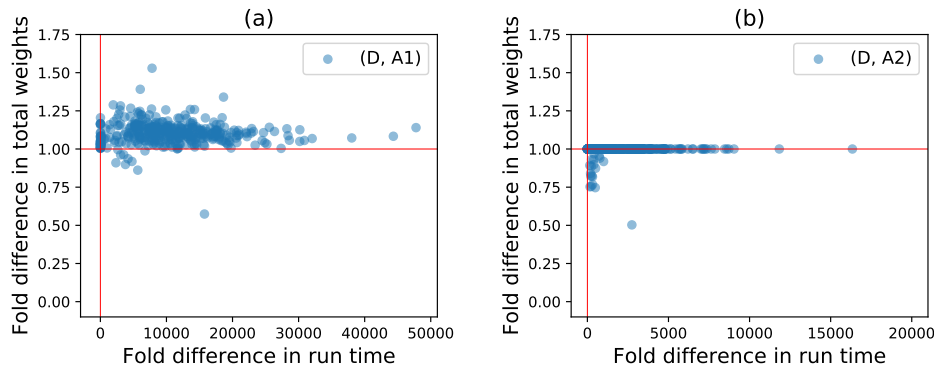


Figure 4 Fold differences (ILP/approx) in run time and total weights of concordant edges resolved by D-SQUID, A1 and A2 on TCGA samples. Horizontal and vertical red lines mark 1.0 on both axes. (a) shows fold differences between D-SQUID and A1. (b) shows fold differences between D-SQUID and A2.

9 Conclusion and Discussion

We present approaches to identify TSVs in heterogeneous samples via the MULTIPLE COMPATIBLE ARRANGEMENT PROBLEM (MCAP). We characterize sample heterogeneity in terms of the fraction of discordant edges involved in conflict structures. In the majority of TCGA samples, the fractions of discordant edges in conflict structures are high compared to HCC samples, which indicates that TCGA samples are more heterogeneous than HCC samples. This matches the fact that bulk tumor samples often contain more heterogeneous genomes than cancer cell lines, which suggests that fraction of conflicting discordant edges is a valid measure of sample heterogeneity.

MCAP addresses this heterogeneity. In 381 TCGA samples, D-SQUID is able to resolve more conflicting discordant edges than SQUID. In HCC cell lines, D-SQUID achieves better performance than SQUID. Since D-SQUID solves MCAP by separating conflicting TSVs onto two alleles, D-SQUID's power to find TSVs generally increases as the extent of heterogeneity increases.

We show that obtaining exact solutions to MCAP is NP-hard. We derive an integer linear programming (ILP) formulation to solve MCAP exactly. We provide a $\frac{3}{16}$ -approximation algorithm for MCAP when the number of arrangements is two ($k = 2$), which runs in time $O(|V||E|)$. It approximates the exact solutions well in heterogeneous TCGA samples.

Several open problems remain. MCAP relies on the number of arrangements (k) to make predictions. It is not trivial to determine the optimal k for any sample. In addition, although MCAP is solved by separating TSVs onto different alleles, there are typically many equivalent phasings. Developing techniques for handling these alternative phasings is an interesting direction for future work. Analyzing the effect of TSVs, especially non-fusion-gene ones, on cellular functions and diseases is another direction of further work.

References

- 1 Dvir Aran, Marina Sirota, and Atul J Butte. Systematic pan-cancer analysis of tumour purity. *Nature Communications*, 6:8971, 2015.
- 2 Ken Chen, John W Wallis, Michael D McLellan, David E Larson, Joelle M Kalicki, Craig S Pohl, Sean D McGrath, Michael C Wendl, Qunyuan Zhang, Devin P Locke, et al. BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature Methods*, 6(9):677, 2009.
- 3 Nadia M Davidson, Ian J Majewski, and Alicia Oshlack. JAFFA: High sensitivity transcriptome-focused fusion gene detection. *Genome Medicine*, 7(1):43, 2015.
- 4 Michael WN Deininger, John M Goldman, and Junia V Melo. The molecular biology of chronic myeloid leukemia. *Blood*, 96(10):3343–3356, 2000.
- 5 Jesse R Dixon, Jie Xu, Vishnu Dileep, Ye Zhan, Fan Song, et al. Integrative detection and analysis of structural variation in cancer genomes. *Nature Genetics*, 50(10):1388, 2018.
- 6 Adi F Gazdar, Venkatesh Kurvari, Arvind Virmani, Lauren Gollahon, Masahiro Sakaguchi, et al. Characterization of paired tumor and non-tumor cell lines established from patients with breast cancer. *International Journal of Cancer*, 78(6):766–774, 1998.
- 7 Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- 8 Steffen Heber, Max Alekseyev, Sing-Hoi Sze, Haixu Tang, and Pavel A Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics*, 18(suppl_1):S181–S188, 2002.
- 9 Fereydoun Hormozdiari, Iman Hajirasouliha, Phuong Dao, Faraz Hach, Deniz Yorukoglu, Can Alkan, Evan E Eichler, and S Cenk Sahinalp. Next-generation VariationHunter: combinatorial algorithms for transposon insertion discovery. *Bioinformatics*, 26(12):i350–i357, 2010.
- 10 Zhiqin Huang, David TW Jones, Yonghe Wu, Peter Lichter, and Marc Zapatka. confFuse: high-confidence fusion gene detection across tumor entities. *Frontiers in Genetics*, 8:137, 2017.
- 11 Wenlong Jia, Kunlong Qiu, Minghui He, Pengfei Song, Quan Zhou, et al. SOAPfuse: an algorithm for identifying fusion transcripts from paired-end RNA-Seq data. *Genome Biology*, 14(2):R12, 2013.
- 12 Ryan M Layer, Colby Chiang, Aaron R Quinlan, and Ira M Hall. LUMPY: a probabilistic framework for structural variant discovery. *Genome Biology*, 15(6):R84, 2014.
- 13 Silvia Liu, Wei-Hsiang Tsai, Ying Ding, Rui Chen, Zhou Fang, et al. Comprehensive evaluation of fusion transcript detection algorithms and a meta-caller to combine top performing methods in paired-end RNA-seq data. *Nucleic Acids Research*, 44(5):e47–e47, 2015.
- 14 Cong Ma, Mingfu Shao, and Carl Kingsford. SQUID: transcriptomic structural variation detection from RNA-seq. *Genome Biology*, 19(1):52, 2018.
- 15 Andrew McPherson, Fereydoun Hormozdiari, Abdalnasser Zayed, Ryan Giuliany, Gavin Ha, et al. deFuse: an algorithm for gene fusion discovery in tumor RNA-Seq data. *PLoS Computational Biology*, 7(5):e1001138, 2011.
- 16 Daniel Nicorici, Mihaela Satalan, Henrik Edgren, Sara Kangaspeska, Astrid Murumagi, Olli Kallioniemi, Sami Virtanen, and Olavi Kilkku. FusionCatcher—a tool for finding somatic fusion genes in paired-end RNA-sequencing data. *BioRxiv*, page 011650, 2014.
- 17 Nicholas A Nystrom, Michael J Levine, Ralph Z Roskies, and J Scott. Bridges: a uniquely flexible HPC resource for new communities and data analytics. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 30. ACM, 2015.
- 18 Tobias Rausch, Thomas Zichner, Andreas Schlattl, Adrian M Stütz, Vladimir Benes, and Jan O Korbel. DELLY: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics*, 28(18):i333–i339, 2012.
- 19 Robert Sedgewick. *Algorithms in C, Part 5: Graph Algorithms, Third Edition*. Addison-Wesley Professional, third edition, 2001.

- 20 Fritz J Sedlazeck, Philipp Rescheneder, Moritz Smolka, Han Fang, Maria Nattestad, Arndt von Haeseler, and Michael C Schatz. Accurate detection of complex structural variations using single-molecule sequencing. *Nature Methods*, 15(6):461–468, 2018.
- 21 Wandaliz Torres-García, Siyuan Zheng, Andrey Sivachenko, Rahulshimham Vegesna, Qianghu Wang, Rong Yao, Michael F Berger, John N Weinstein, Gad Getz, and Roel GW Verhaak. PRADA: pipeline for RNA sequencing data analysis. *Bioinformatics*, 30(15):2224–2226, 2014.
- 22 Xiaoke Wang, Renata Q Zamolyi, Hongying Zhang, Vera L Pannain, Fabiola Medeiros, Michele Erickson-Johnson, Robert B Jenkins, and Andre M Oliveira. Fusion of HMGA1 to the LPP/TPRG1 intergenic region in a lipoma identified by mapping paraffin-embedded tissues. *Cancer Genetics and Cytogenetics*, 196(1):64–67, 2010.

A Proof of NP-hardness

► **Theorem 1.** *SCAP is NP-hard.*

Proof. To prove the NP-hardness, we reduce from MAX-2-SAT problem. It is necessary and sufficient to show that for any MAX-2-SAT problem, a genome segment graph (GSG) can be constructed in polynomial time, and the SCAP objective directly tells the objective of the MAX-2-SAT problem. For any MAX-2-SAT instance, we are going to construct a GSG such that the satisfiability of a clause is indicated by the concordance of an edge.

Given a MAX-2-SAT problem with n booleans $\{x_1, x_2, \dots, x_n\}$ and m clauses $\{c_1, c_2, \dots, c_m\}$, the key gadget is the segments for boolean variables and clauses and the edges between them (Figure 5A). For each boolean variable x_i , a segment X_i is constructed and termed as a boolean segment. For each clause c_i , a segment C_i is constructed and termed as a clause segment. To ensure that the correspondence between the edge concordance and the clause satisfiability as well as the correspondence between the orientation of boolean segments and the assignment of boolean variables, we add edges between clause segments and boolean segments in the following way. For clause c_i that involves boolean x_{i_1} , an edge is added between the head of X_{i_1} and the head of C_i if clause c_i contains the negation of x_{i_1} , otherwise the edge is between the tail of X_{i_1} and the head of C_i . When the literal is x_{i_1} , setting the orientation of segment X_{i_1} to be 1 indicates assigning True to variable x_{i_1} and leads to the concordance of the edge; when the literal is \bar{x}_{i_1} , setting the orientation of segment X_{i_1} to be 0 indicates assigning False to variable x_{i_1} and leads to the edge concordance. The edge between clause c_i and the other involved boolean variable x_{i_2} is added in the same principle. We call the edge between boolean segments and the clause segments as Type 1 edge. Type 1 edges have weight of 1.

Two extra edges between the two boolean segments involved in each clause are added. This is the Type 2 edge with weight of 1. For each clause c_i that involves boolean x_{i_1} and x_{i_2} , two edges are added between X_{i_1} and X_{i_2} as in Table 1. When both literals in c_i are True, there are two concordant Type 1 edges; when only one literal in c_i is True, one and only one of the two Type 2 edges is guaranteed to be concordant, to compensate for the decrease of concordant Type 1 edges.

An extra $n + m + 1$ segments are added that we term blocking segments and denote as $\{B_1, B_2, \dots, B_{n+m+1}\}$. Suppose w_1 and w_2 are large positive weights, and $w_2 \gg w_1 \gg 1$. Type 3 edges with edge weight w_2 are constructed between each adjacent pair of blocking segments, specifically between the tail of B_i and the head of B_{i+1} ($\forall i \in [1, n + m]$). Type 3 edges are used to enforce the order and orientation among blocking segments. Type 4 edges with weight w_1 are constructed between blocking segments and the other types of segments. Specifically, when $i \leq n$, an edge is added between the tail of segment B_i and both the head and the tail of X_i , as well as between the tail and the head of X_i and both the head of B_{i+1} .

Similarly when $n < i \leq n + m$, two edges are added between the tail of B_i and C_{i-n} , and two other edges are added between the head and tail of C_{i-n} and B_{i+1} . Type 4 edges are used to enforce the relative order between blocking segments and the boolean and clause segments. But the orientation of the boolean and clause segments can be changed freely.

■ **Table 1** Construction of Type 4 edges based on the clause.

clause c_i	edge 1	edge 2
$x_{i_1} \vee x_{i_2}$	head of X_{i_1} to head of X_{i_2}	tail of X_{i_1} to tail of X_{i_2}
$\bar{x}_{i_1} \vee x_{i_2}$	tail of X_{i_1} to head of X_{i_2}	head of X_{i_1} to tail of X_{i_2}
$x_{i_1} \vee \bar{x}_{i_2}$	tail of X_{i_1} to head of X_{i_2}	head of X_{i_1} to tail of X_{i_2}
$\bar{x}_{i_1} \vee \bar{x}_{i_2}$	head of X_{i_1} to head of X_{i_2}	tail of X_{i_1} to tail of X_{i_2}

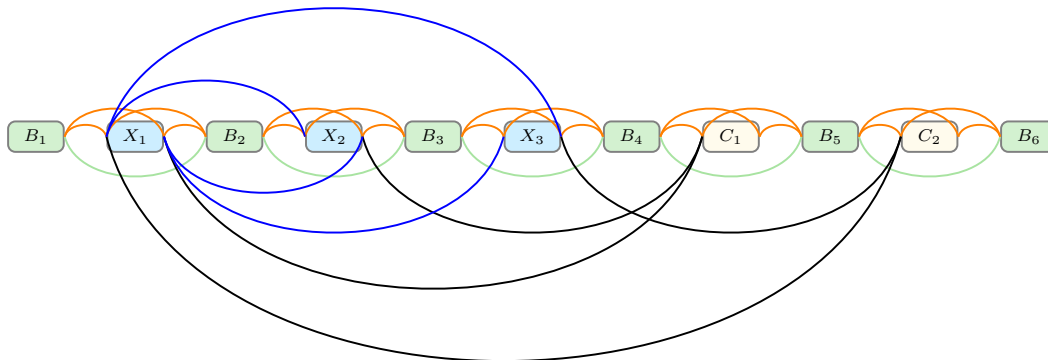
We first prove that the order of the blocking segments in the optimal arrangement is $B_1 < B_2 < \dots < B_{n+m+1}$ and the orientations of them are all in forward strand, where $<$ denotes the ordering between segments. Under the arrangement that uses the forward strand of all $\{B_i\}$ and have an order of $B_1 < B_2 < \dots < B_{n+m+1}$, the sum of concordant edge weights is at least $(n + m)w_2$. If the optimal arrangement contains any violations of the adjacencies between B_i and B_{i+1} , there will at least one Type 3 edge that does not connect blocking segments in a tail-to-head manner and become a discordant edge in the arrangement. Therefore, the optimal arrangement can at most have an objective value of $(n + m - 1)w_2 + 4(n + m)w_1 + 4m$. Since $w_2 \gg w_1 \gg 1$, the objective value is smaller than $(n + m)w_2$, and the arrangement is not optimal, which contradicts the assumption. Therefore assuming the whole chain of segments is not reverse complemented, the orientations of blocking segments are all in forward strand, and order is $B_1 < B_2 < \dots < B_{n+m+1}$ in the optimal arrangement.

We then prove that the Type 2 edges restrict the order of all segments but not the orientation of boolean and clause segments. The order between blocking segments and boolean segments must be $B_i < X_i < B_{i+1}$, the order between blocking and clause segments must be $B_i < C_{i-n} < B_{i+1}$, and all boolean segments must be before clause segments. When the order is $B_i < X_i < B_{i+1}$ among the three segments, and the orientations of B_i and B_{i+1} are both in forward strand, the concordant edge weights of Type two edge sum to $2w_1$ no matter whether X_i is in forward strand or inverted. The same weight can be achieved for order $B_i < C_{i-n} < B_{i+1}$. The arrangement with order $B_1 < X_1 < B_2 < \dots < B_n < X_n < B_{n+1} < C_1 < B_{n+2} < \dots < C_m < B_{n+m+1}$ and with all blocking segments in their forward strand will achieve a sum of concordant edge weight $(n + m)w_2 + 2(n + m)w_1$ at least. This concordant weight is summed over Type 3 and Type 4 edges. However, if the optimal arrangement violates any $B_i < X_i < B_{i+1}$ or $B_i < C_{i-n} < B_{i+1}$ order, the violated triplet can achieve at most w_1 of concordant edge weights, and thus the maximum sum of concordant edge weights is $(n + m)w_2 + 2(n + m - 1)w_1 + w_1 + 4m$. Since $w_1 \gg 1$, the “optimal” arrangement objective is smaller than $(n + m)w_2 + 2(n + m)w_1$, which contradicts the optimality. Therefore, the order of all segments in the optimal arrangement must be

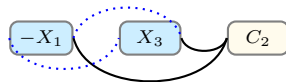
$$B_1 < X_1 < B_2 < \dots < B_n < X_n < B_{n+1} < C_1 < B_{n+2} < \dots < C_m < B_{n+m+1}.$$

Third, we prove that under the above segment order there are always two concordant edges of weight 1 when clause segment C_i has any concordant Type 1 edge. Suppose there is a clause c_i involving boolean variables x_{i_1} and x_{i_2} , segment C_i has one Type 1 edge between X_{i_1} and one Type 1 edge between X_{i_2} . When both Type 1 edges are concordant, both

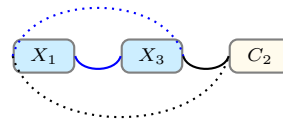
(A)



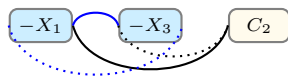
(B)



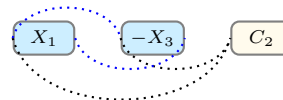
(C)



(D)



(E)



■ **Figure 5** (A) Constructed GSG for boolean expression $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3)$. There is a segment for each boolean variable x_i (blue) and clause c_i (white), and 6 blocking segments (green) to separate between boolean segments and clause segments. Type 1 edges, black edges, are connecting between boolean segments and clause segments. Type 2 edges, blue edges, are connecting between a pair of boolean segments that appear in the same clause. Type 3 edges, green edges in the figure, are chaining the blocking segments. Type 4 edges, orange edges, are connecting between blocking and boolean / clause segments. (B-E) The subgraph corresponding clause $\bar{x}_1 \vee x_3$. $-X_1$ and $-X_3$ means the segment is inverted. Solid lines indicate the concordant edges in the arrangement, and dotted lines indicate the discordant edges. (B) The clause is satisfied with both literals satisfied. (C) The clause is satisfied with x_3 satisfied. (D) The clause is satisfied with \bar{x}_1 satisfied. (E) The clause is not satisfied.

Type 2 edges between X_{i_1} and X_{i_2} are discordant (Figure 5B). When only one of the Type 1 edges is concordant, there is also one Type 2 edge between X_{i_1} and X_{i_2} that is concordant (Figure 5C,D). When neither of the Type 1 edges is concordant, both of the two Type 2 edges between X_{i_1} and X_{i_2} are discordant (Figure 5E). In this case, there is zero concordant edges of weight 1 incident to C_i . Any arrangement solution of objective value W that satisfies the above segment order has $W - (n + m)w_2 - 2(n + m)w_1$ concordant edges of weight 1. Therefore, the arrangement solution will have $\frac{1}{2}(W - (n + m)w_2 - 2(n + m)w_1)$ clause segments with non-zero concordant Type 1 edges.

When multiple clauses involve the same pair of segment, multi-edges between X_{i_1} and X_{i_2} are constructed to make sure that two edges of weight 1 are contributed by any clause segment when it has non-zero concordant Type 1 edges.

Suppose the optimal number of satisfied clauses of the MAX-2-SAT instance is OPT_m and the optimal sum of concordant edge weights of the constructed SCAP instance is OPT_s , the following inequality holds: $\frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1) \geq OPT_m$. Given the optimal solution of the MAX-2-SAT instance, a SCAP solution can be constructed by reversing segment X_i if x_i is assigned to False while keeping the order of $B_1 < X_1 < B_2 < \dots < B_n < X_n < B_{n+1} < C_1 < B_{n+2} < \dots < C_m < B_{n+m+1}$. By the construction of the Type 1 edges, a clause segment will have at least one concordant Type 1 edge if and only if it corresponds to a satisfied clause in the MAX-2-SAT solution. Denoting the objective value of the constructed solution of arrangement problem as W and applying the third proof, we have the following equality $OPT_m = \frac{1}{2}(W - (n+m)w_2 - 2(n+m)w_1)$. Since the optimal objective value of the arrangement problem is at least W ,

$$OPT_m = \frac{1}{2}(W - (n+m)w_2 - 2(n+m)w_1) \leq \frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1). \quad (9)$$

Meanwhile $\frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1) \leq OPT_s$. Given the optimal solution of arrangement problem, there are $\frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1)$ clause segments with non-zero concordant Type 1 edges. Construct a MAX-2-SAT solution by assigning False to boolean variables if the corresponding boolean segment is reversed otherwise assigning True. The concordance of Type 1 edges guarantees that the corresponding literals in the MAX-2-SAT clauses are True. Thus the constructed MAX-2-SAT solution will have $\frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1)$ satisfied clauses, which is smaller than or equal to the optimal number of satisfied clauses. Therefore

$$\frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1) \leq OPT_m. \quad (10)$$

Combining inequality (9) and inequality (10), the maximum number of satisfied clauses in MAX-2-SAT instance can be directly calculated from the optimal concordant edge weights in the arrangement problem, that is, $OPT_m = \frac{1}{2}(OPT_s - (n+m)w_2 - 2(n+m)w_1)$. ◀

B Proof of Characterization of Conflict Structures

► **Theorem 4.** *Any acyclic subgraph of GSG is a compatible structure.*

Proof. We show that any acyclic subgraph with N edges ($|E'| + |\hat{E}'| = N$), $G'_N = (E', \hat{E}', V')$, of GSG is a compatible structure by induction.

When $|E'| + |\hat{E}'| = 1$, G'_1 is a compatible structure because no other edge in G' is in conflict with the only edge $e \in E'$.

Assume the theorem hold for any acyclic subgraph that contains n edges. Let $G'_{n+1} = (E', \hat{E}', V')$ be an acyclic subgraph with $n+1$ edges. Since G'_{n+1} is acyclic, there must be a leaf edge that is incident to a leaf node. Denote the leaf node as v_b and the leaf edge $e = (u_a, v_b) \in E' \cup \hat{E}'$ ($a, b \in \{h, t\}$). By removing edge e and leaf node v_b , the subgraph $G'_n = (E' - \{e\}, \hat{E}' - \{e\}, V' - \{v_b\})$ is also acyclic and contains n edges. According to the assumption, G'_n is a compatible structure and there is an arrangement of the segments in which all edges in $E' \cup \hat{E}' - \{e\}$ is concordant. Because no other edge in $E' \cup \hat{E}'$ except e connects to v_b , it is always possible to place segment v back to the arrangement such that e is concordant. Specifically, one of the four placing options will satisfy edge e : the beginning of the arrangement with orientation 1, the beginning with orientation 0, the end with orientation 1 and the end with orientation 0. Therefore, G'_{n+1} is a compatible structure. By induction, acyclic subgraph G'_N of GSG with any $|E'|$ is a compatible structure. ◀

► **Theorem 5.** *A simple cycle $C = (E', \hat{E}', V')$ is a compatible structure if and only if there are exactly two vertices, v_j and v_i such that $\deg_{E'}(v_i) = \deg_{E'}(v_j) = 2$ and v_i and v_j belongs to different segments.*

Proof. We prove sufficiency and necessity separately in Lemma 6 and Lemma 7. ◀

► **Lemma 6.** *If C is a compatible structure, there are exactly two vertices, v_i, v_j that belong to different segments, such that $\deg_{E'}(v_i) = \deg_{E'}(v_j) = 2$*

Proof. We discuss compatibility in two cases:

Case (1): All edges are concordant in C . Sort the vertices by genomic locations in ascending order and label the first vertex v_1 and the last v_n , assuming $|V'| = n$. Similarly, sort the set of segments S' in C by the values of their permutation function π and label the first segment s^1 and the last s^m , assuming $|S'| = m$. Since concordant connections can only be tail-head connections (e.g. Figure 1 b,c), $v_1 = s_t^1$ and $v_n = s_h^m$. Since C is a simple cycle, all vertices $v \in V'$ have $\deg(v) = 2$. Because v_1 and v_n are the first and last vertices in this arrangement, the edges incident to v_1 or v_n must be in E' . It follows that the two edges incident to v_1 connects to s_t^2 and s_h^m . Similarly, edges incident to v_n connects to s_t^1 and s_t^{n-1} . Therefore, we have $\deg_{E'}(v_1) = \deg_{E'}(v_n) = 2$. Any other vertex v_i ($1 < i < n$) is connected by one $e \in E'$ and one $\hat{e} \in \hat{E}'$ and thus has $\deg_{E'}(v_i) = 1$.

Case (2): Some edges are discordant in C . If discordant edges exist in cycle C , according to the definition of compatible structure, segments in C can be arranged such that all edges are concordant. This reduces to case (1). ◀

► **Lemma 7.** *If there are exactly two vertices in V' that belong to different segments, v_i and v_j , such that $\deg_{E'}(v_i) = \deg_{E'}(v_j) = 2$, then C is a compatible structure.*

Proof. Let v_i and v_j be the one of the end points of segments s^i and s^j ($i \neq j$), respectively. We can arrange s^i and s^j such that $\pi(s^i) = \min_{s \in S'} \pi(s)$, $\pi(s^j) = \max_{s \in S'} \pi(s)$ and that $v_i = s_t^i$, $v_j = s_h^j$. Rename v_i to v_1 and v_j to v_n . Since C is a simple cycle, we can find two simple paths, P_1 and P_2 , between v_1 and v_n and there is no edge between P_1 and P_2 . Let P'_1 and P'_2 denote P_1 and P_2 that exclude v_1 and v_n and the edges incident to v_1 and v_n . Since P'_1 and P'_2 as acyclic subgraphs of GSG, according to Theorem 14, P'_1 and P'_2 are compatible structures and therefore segments in P'_1 and P'_2 can be arranged so that all edges are concordant. Denote the first and last vertices in the arranged P'_1 as v_2 and v_3 , and the first and last vertices in the arranged P'_2 as v_4 and v_5 . Because all the edges are concordant in P'_1 , v_2 and v_3 are the head and tail of the first and last segments in P'_1 . Because only v_1 and v_n have $\deg_{E'} = 2$ in C , v_2 must be connected to v_1 or v_n and v_3 must be connected to v_n or v_1 . A similar argument applies to v_4 and v_5 . To ensure concordance of edges connected to v_1 and v_n , if v_n is connected to v_2 and v_1 is connected to v_3 , we flip all the segments in P'_1 . The similar operation is applied to v_4, v_5 and P'_2 . Now we have a compatible structure. ◀

Faster Pan-Genome Construction for Efficient Differentiation of Naturally Occurring and Engineered Plasmids with Plaster

Qi Wang 

Systems, Synthetic, and Physical Biology (SSPB) Graduate Program,
Rice University, Houston, TX 77005, USA
qw17@rice.edu

R. A. Leo Elworth 

Department of Computer Science, Rice University, Houston, TX 77005, USA
r.a.leo.elworth@rice.edu

Tian Rui Liu 

Department of Computer Science, Rice University, Houston, TX 77005, USA
tl62@rice.edu

Todd J. Treangen 

Department of Computer Science, Rice University, Houston, TX 77005, USA
treangen@rice.edu

Abstract

As sequence databases grow, characterizing diversity across extremely large collections of genomes requires the development of efficient methods that avoid costly all-vs-all comparisons [17]. In addition to exponential increases in the amount of natural genomes being sequenced, improved techniques for the creation of human engineered sequences is ushering in a new wave of synthetic genome sequence databases that grow alongside naturally occurring genome databases. In this paper, we analyze the full diversity of available sequenced natural and synthetic plasmid genome sequences. This diversity can be represented by a data structure that captures all presently available nucleotide sequences, known as a pan-genome. In our case, we construct a single linear pan-genome nucleotide sequence that captures this diversity. To process such a large number of sequences, we introduce the *plaster* algorithmic pipeline. Using *plaster* we are able to construct the full synthetic plasmid pan-genome from 51,047 synthetic plasmid sequences as well as a natural pan-genome from 6,642 natural plasmid sequences. We demonstrate the efficacy of *plaster* by comparing its speed against another pan-genome construction method as well as demonstrating that nearly all plasmids align well to their corresponding pan-genome. Finally, we explore the use of pan-genome sequence alignment to distinguish between naturally occurring and synthetic plasmids. We believe this approach will lead to new techniques for rapid characterization of engineered plasmids. Applications for this work include detection of genome editing, tracking an unknown plasmid back to its lab of origin, and identifying naturally occurring sequences that may be of use to the synthetic biology community. The source code for fully reconstructing the natural and synthetic plasmid pan-genomes as well for *plaster* are publicly available and can be downloaded at <https://gitlab.com/qiwangrice/plaster.git>.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Molecular sequence analysis; Applied computing → Computational genomics

Keywords and phrases comparative genomics, sequence alignment, pan-genome, engineered plasmids

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.19

Funding *Qi Wang*: Q. W. was supported by funds from Rice University and by funds from the National Institute for Neurological Disorders and Stroke (NINDS) of the National Institutes of Health under award number R21NS106640.

R. A. Leo Elworth: R. A. L. E. was supported by the National Science Foundation (DMS-1547433) and was partially supported by the FunGCAT program from the Office of the Director of National



© Qi Wang, Ryan A. L. Elworth, Tian Rui Liu, and Todd J. Treangen;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 19; pp. 19:1–19:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the Army Research Office (ARO) under Federal Award No. W911NF-17-2-0089.

Tian Rui Liu: T.R.L. was supported by funds from Rice University.

Todd J. Treangen: T.J.T was supported by startup funds from Rice University and was partially supported by the FunGCAT program from the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the Army Research Office (ARO) under Federal Award No. W911NF-17-2-0089.

Acknowledgements The authors would like to thank Dr. Caleb Bashor for critical discussion and feedback, and Dr. Joanne Kamens from Addgene for providing full access to the synthetic plasmids utilized in this study. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, ARO, or the US Government.

1 Introduction

Thanks to the advancement of sequencing and genome-editing technologies, over the past decade genome engineering has become more affordable and accessible. Plasmids are commonly found as double-stranded DNA, which can replicate independently from chromosomal DNA [12]. They are ubiquitous in bacteria and provide benefits such as enhanced host range [4], antibiotic resistance [18], and even forced cooperation [20]. As plasmids are easy to engineer and can confer functions in a broad range of species, they are widely used in biology labs for understanding and modifying genetic elements. To date, more than 65 thousand engineered plasmids have been deposited in the Addgene repository [13]. With the benefits of genetic engineering microbial sequences also come risks [3], some with significant dual use of research concerns [22]. In response, methods have been designed both to detect signatures of engineering [1] and also trace back synthetic plasmids to a lab of origin using deep learning methods [19]. Although deep learning shows potential to characterize engineered plasmids and identify the lab-of-origin, there is a need for explainable, white-box approaches that can provide a full list of sequence features specific to a biological function or lab. We thus propose a novel method that enables the detection of a plasmid’s lab-of-origin with precise characterization the diversity within and among all engineered plasmid sequences.

Traditionally, a single genome is chosen as a reference to describe genome diversity within a group. Unfortunately, a single genome usually fails to reveal the full picture of similarities and discrepancies among all individuals. The “pan-genome” concept was introduced to reflect the diversity of all strains within a specific clade [31, 27]. This concept is similar to the problem of finding a smaller set of founder sequences, which can map to a given sequence in the group, called the *founder sequence reconstruction* problem [30]. The pan-genome is designed to describe both the core genome shared by all the individuals and the accessory genome contained by only some strains [27]. This concept can be applied to different domains. In metagenomic studies, a pan-genome can highlight essential genetic elements responsible for adaptations to the environment and the co-evolution interactions among the microorganisms [17]. In terms of building a phylogenetic tree, pan-genomes can detect weak evolutionary signals, which may be omitted by multiple sequence alignment. It also opens up the possibilities for discovering the origin of unknown organisms [33], pathogen transmission history [6], or the inference of cancer cell evolution [10].

In microbes, the pan-genome is commonly defined as the union set of genes that exist in all the genomes in a selected clade [27]. Many bioinformatic tools have been developed to build gene-based pan-genomes, such as PanOCT [8], PGAP [35], and Roary [24]. Given

their gene-centric construction of the pan-genome, they are vulnerable to missing genes [34] and do not include intergenic regions, which can have substantial impact on phenotypes [28]. To address these limitations, methods such as Piggy [29] build pan-genomes from intergenic regions. In this paper we move away from a gene and intergenic centric approach; the term pan-genome refers to the sum of all core and accessory DNA sequence fragments which are longer than a minimal sequence fragment length l (*plaster* default $l = 50bp$).

In order to efficiently analyze vast numbers of DNA sequences, there is a need to create a pan-genome sequence that encodes all existing variations in the minimal possible size [21]. To solve this problem efficiently via pan-genomics we introduce *plaster*, a new pan-genome construction algorithm inspired by fast DNA clustering techniques [5]. *plaster*'s speed exceeds the fastest existing tool seq-seq-pan and can be easily scaled to handle large data sizes. Seq-seq-pan [14] is a recently introduced method for efficient pan-genome construction that was shown to be an order of magnitude faster than the fastest available methods for pan genome construction [7]. In brief, it relies on progressiveMauve to catalog all of the differences (including insertions, deletions, substitutions, inversions, and rearrangements) within a set of genomes through multiple sequence alignment. Then, it applies majority vote to decide on consensus sequences from a set of segments of aligned sequences and merges those sequences with delimiter sequences. Our approach *plaster* employs a similar approach, but instead we calculate pairwise alignments with NUCmer and identify unaligned regions using dnadiff. We then append all unaligned regions along with delimiter sequences to the end of the reference sequence Fig. 1. We introduce the algorithm of *plaster* in detail and compare its performance with seq-seq-pan. In addition, we describe differences and similarities between the synthetic and natural plasmid pan-genome sequences based on pairwise alignment results.

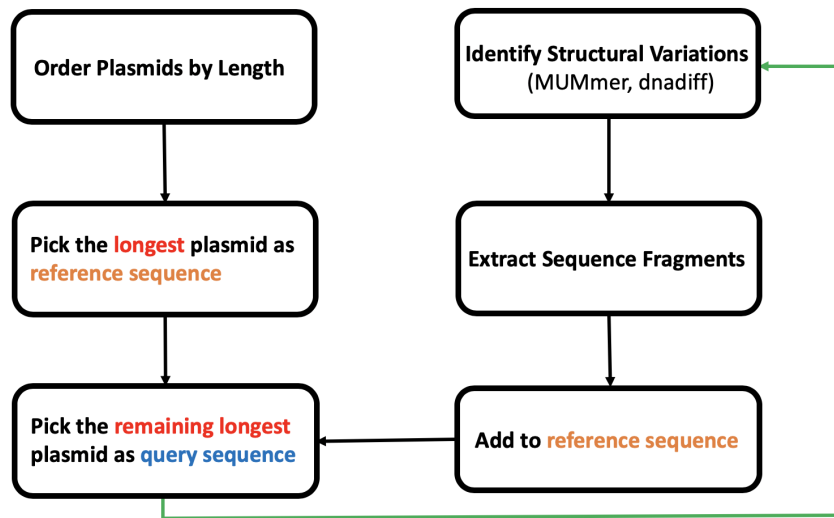
2 Methods

2.1 *plaster* Workflow

The goal of our algorithm is to construct a pan-genome P from a set $S = [s_0, s_1, \dots, s_n]$ of n genome sequences. Throughout this paper, we focus on plasmid sequences, though any arbitrary sequences can be used to construct the pan-genome P . A high level overview of our algorithmic pipeline is outlined in Fig. 1. Though not necessary to run the *plaster* software, in all analyses in this paper we first sort S so that $|s_0| \geq |s_1| \geq \dots \geq |s_n|$ where $|s|$ denotes the length of a sequence s . This initialization offers optimal performance of the method and we recommend it to be used as a standard practice for users of *plaster*.

As shown in Fig. 1, the pipeline for our full *plaster* algorithm can be broken up into a series of discrete tasks. After ordering the sequences by length, the pan-genome P , also referred to as the reference sequence, is initialized as the longest sequence s_0 in S . We then proceed to build the full pan-genome by looping through all the remaining sequences, $[s_1, s_2, \dots, s_n]$, and performing the steps detailed in Fig. 1. We refer to each of these sequences as a query sequence which we compare to the current pan-genome reference sequence.

For each query sequence, *plaster* begins by identifying the regions that do not align between the query and current reference sequence. For this, NUCmer and dnadiff [25] (from the MUMmer package [16], default dnadiff parameters were used). Pairwise genome alignment is performed by NUCmer; following alignment, dnadiff provides detailed analysis of all differences between the two sequences. Once the differences between the query and reference sequences have been calculated, we add unaligned query sequence regions to the pan-genome reference. The .report file output by dnadiff contains the total alignment percentage



■ **Figure 1** *plaster* Pipeline: To build a pan-genome sequence, first we order all of the input sequences by length. We next select the longest plasmid as an initial reference sequence and the second longest plasmid as the first query sequence. We perform pairwise local alignment to identify DNA sequences that are not contained in the current pan-genome. We then extract the novel sub-sequences from the query sequence and update the pan-genome with these sequences. We continue the pan-genome construction process by selecting the next longest sequence in the remaining data set as the next query sequence. We iterate over all input sequences until the entire set has been processed.

between the reference and query sequence. If the alignment percentage is zero, that is, the query sequence is totally unaligned to the reference, we add the entire query to the plasmid pan-genome by appending a delimiter sequence and the entire query sequence to the end.

Following this procedure, there are five different types of structural variants we capture: gaps, duplications, breaks, relocations, and inversions. Each of these represents a different type of sequence variant. For the purpose of capturing the minimal pan-genome, we only care about variations where there are nucleotides in the query sequence that are not described in the current reference. This includes: i) gaps, which identifies indels between two consistently ordered and oriented alignments, and ii) breaks, which refers to a fragment of query sequence not aligned to the reference sequence. Inversions, relocations, and duplications are sequence variants of interest, but do not require an update to the linear pan-genome. By default, if the length of a gap or break area is longer than l (default is 50bp), the corresponding unaligned query sequence area will be appended to the end of the reference sequence after an additional delimiter sequence. This process of finding and appending all differences between a query sequence and the current pan-genome reference sequence continues until every sequence in S has been iterated over. The final output of *plaster* is the resulting reference sequence which represents a linear pan-genome P for the genomes of S .

3 Results

Having developed a novel algorithm for efficiently constructing pan-genomes that can handle very large numbers of sequences, we set out to categorize the differences between all known naturally occurring and synthetic plasmids. To justify the efficacy and utility of our newly developed *plaster* algorithm, we perform experiments on the running time of our method.

The run-time for *plaster* also includes the time for sorting the input sequences based on length. We build the full pan-genomes of all natural and synthetic plasmids and we take a first look at a novel approach for categorizing plasmids as natural or synthetic based on percent alignment against these two pan-genomes. All the experiments in this paper are written in python and run on a server running Ubuntu 18.04 LTS with two Intel Xeon Gold 6138 2.0 GHz processors at 1 terabyte of RAM.

3.1 Evaluation of sensitivity

A total of 43 *M. tuberculosis* genomes were used to build a pan-genome sequence in [14]. However, only 41 genomes were still available in [23]. We used these 41 *M. tuberculosis* genomes to build a pan-genome sequence using both seq-seq-pan and *plaster*. Seq-seq-pan requires 34 min to construct the pan-genome sequence. On the other hand, it only takes *plaster* around 4.2 min. The total length of the pan-genome sequence built by *plaster* is 385,335 bp shorter than the one from seq-seq-pan. However, the pan-genome sequence generated by *plaster* can align a total of 8,828 bp more (or 99.994% average alignment percentage compared to 99.989% for seq-seq-pan) when performing pairwise alignment of all 41 *M. tuberculosis* sequences as query sequences against the pan-genome built from these same 41 genomes as the reference sequence. This shows that *plaster* is an order of magnitude faster than seq-seq-pan while simultaneously producing a more compacted pan-genome sequence and capturing more detailed variations within it.

3.2 Evaluation of speed

Given the rapid increase in the number of natural and engineered plasmids, there is a need to develop a platform which can build pan-genomes for large data sets within a reasonable time and be able to update the pan-genome information quickly for newly sequenced genomes. To evaluate the pan-genome construction speed of *plaster*, we first performed experiments on subsets of synthetic plasmid sequences and compared the performance of *plaster* against the state of the art seq-seq-pan pan-genome construction method. We randomly selected subsets of 10, 100, and 1000 synthetic plasmids from the full set of all synthetic plasmids. Then, we recorded the wall clock time for building a pan-genome from these sequences from start to finish.

As seen in Table 2, the performance of *plaster* for building a pan-genome sequence ranges from 10 to 100 times faster than seq-seq-pan. *plaster* processes plasmids at a rate of 0.25 seconds/plasmid for 10 plasmids, 0.3 seconds/plasmid for 100 plasmids, and 0.337 seconds/plasmid for 1000 plasmids. For seq-seq-pan, the rate is 2.56 seconds/ plasmid, 4.7 seconds/plasmid, and 35.3 seconds/plasmid for 10, 100, and 1000 plasmids respectively. From these results we can see that seq-seq-pan will not scale to building pan-genome sequences for all known synthetic or natural plasmids. On the other hand, *plaster* is able to build a pan-genome for 51,047 complete engineered plasmid sequences in 8.9 hours.

■ **Table 1** Run-time, sensitivity, and pan-genome length for 41 *M. tuberculosis* genomes.

	Seq-seq-pan	<i>plaster</i>
Run-time (s)	2058.64	252.215
Pan-genome length (bp)	4,874,793	4,489,458
Total Aligned Length (bp)	180,681,735	180,690,563
Average Alignment Percentage (%)	99.989±0.00298	99.994±0.00514

■ **Table 2** Run-time for building a pan-genome sequence and updating a pan-genome sequence with one additional new sequence. We compare the speed of *plaster* to seq-seq-pan when building pan-genome sequences using 10, 100, and 1000 random plasmids. We repeated the 10 and 100 sequence runs 20 times and report their average and standard deviation.

No. of plasmids	Elapsed wall clock time (s)		Elapsed wall clock time (s)	
	Pan-genome Construction		Pan-genome Update	
	seq-seq-pan	<i>plaster</i>	seq-seq-pan	<i>plaster</i>
10	25.265±0.736	1.90±0.0717	3.5±0.163	0.22±0.029
100	495.5±40.926	22.189±0.401	7.84±1.89	0.239±0.0243
1000	35295.3	337.2	86.679	0.886

In addition to the times for full pan-genome construction in Table 2, we compared the timings for the pan-genome update speed. This update speed is the speed at which we can add a single new sequence to an already fully constructed pan-genome. For this experiment, we randomly selected one plasmid from our synthetic plasmid data set and added it to the pan-genome sequences created in the previous experiment for 10, 100 and 1000 plasmids using either seq-seq-pan or *plaster*.

As demonstrated by the results of Table 2, the update speed of seq-seq-pan grows intractably for larger pan-genomes when compared to *plaster*. *plaster* is about 15 times faster for the 10 plasmid pan-genome sequence, 30 times faster for the 100 plasmid pan-genome sequence, and 100 times faster for the 1000 plasmid pan-genome sequence. Given the speed of single sequence updates and the rapid growth of sequence databases, we envision that *plaster* could be used as an online algorithm which updates a pan-genome sequence every time new sequences are added to the database. A final timing experiment was performed showing the update speed for *plaster* for all of a set of 51,047 synthetic plasmid sequences. Fig. 2 shows these update times. The curvature of Fig. 2 suggests that we have created a closed pan-genome, where update times gradually converge to a nearly constant time as the pan-genome grows large.

3.3 Full Natural and Synthetic Plasmid Pan-Genomes

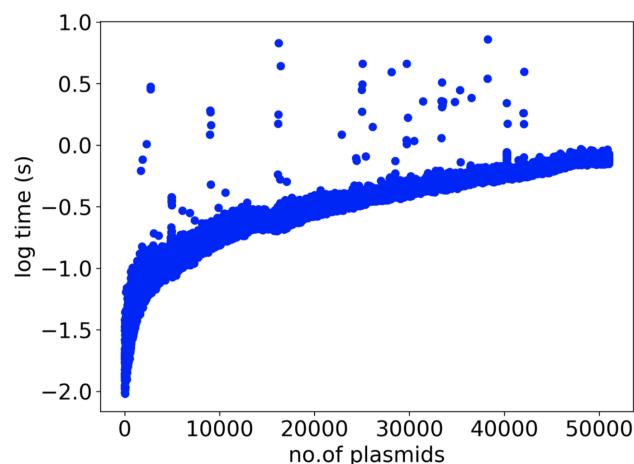
We built a full synthetic plasmid pan-genome and full natural plasmid pan-genome using *plaster* based on 51,047 synthetic plasmid sequences, 73,727 partial synthetic plasmid sequences, and 6,642 natural plasmids. All the synthetic plasmids came from the Addgene engineered plasmid database from January 2019 [13]. A JavaScript Object Notation (JSON) file containing these DNA sequences was obtained directly from Addgene. These sequences were grouped into four categories which were full plasmid sequences submitted by Addgene, partial sequences (segments of the plasmid) submitted by Addgene, full plasmid sequences submitted by a depositing lab, and partial sequences submitted by a depositing lab. There were a total of 51,047 plasmids with complete sequences, in which 23,875 were submitted by Addgene, and 73,727 partial sequences. The average size of a complete synthetic plasmid is 7,159 bp.

To construct the full synthetic plasmid pan-genome, we first ordered all the plasmids with full sequences based on their lengths. Then, we used those ordered plasmids as input to *plaster* to build a full pan-genome sequence. It took about 8.9 hours to build the pan-genome sequence and the final resulting linear pan-genome consisted of 8,307,070 bp. The total number of nucleotides of all the synthetic plasmids with complete sequences was 365,468,935 bp. After building the pan-genome sequence with only full plasmid sequences, we generated

a second pan-genome sequence using both full and partial sequences. We ranked the partial sequences based on their lengths and input those sequences into the *plaster* pipeline as query sequences. Given the online nature of *plaster*, we were able to use the already constructed pan-genome for the full synthetic plasmids as the initial reference sequence, iterating over all the partial sequences as new query sequences. *plaster* spent less than 35 hours to construct the final pan-genome for both full and partial sequences. The total number of nucleotides of this synthetic plasmid pan-genome sequence using both full and partial sequences is 18,163,933. The total number of nucleotides of all the sequences used to construct the pan-genome was 442,923,876. For natural plasmids, we obtained the DNA sequences from the latest database of plasmid sequences [2]. All of these natural plasmids are complete sequences that were curated from the entire NCBI database [23]. There are a total of 6,642 plasmids in this database with an average size of 128,953 bp. The total number of nucleotides of all natural plasmids was nearly one billion base pairs (856,388,404 bp total length). It took *plaster* around 50.2 hr (wall clock time) to build the full natural plasmid pan-genome based on these sequences. The total number of nucleotides in the resulting pan-genome sequence was 205,605,679 bp. The size of the pan-genome sequence is around 24% of the total size of the natural plasmids.

3.4 Pan-genome Sequence Evaluation and Natural Versus Synthetic Differentiation

An exciting new area of research in bioinformatics is in characterizing engineered DNA and in differentiating engineered versus naturally occurring nucleotide sequences. To assess our synthetic and natural pan-genomes, we realigned all natural and synthetic plasmids against these two pan-genomes and report on the percent alignment for each of these plasmids. We also highlight how this technique unveils a novel approach for differentiating synthetic and natural plasmid sequences, given that each is expected to align well to its corresponding pan-genome but not align well to the other pan-genome. Aligning a given query sequence to both pan-genomes can therefore be used as an initial test for evidence of genomes having been subjected to genome-editing.



■ **Figure 2** Cumulative run time per individual plasmid used when building the synthetic plasmid pan-genome sequence. x-axis is the cumulative number of plasmids. y-axis is the cumulative log scale time. As more plasmids are added to the pan-genome, the run time for each individual plasmid increases. The run-time growth rate decreases after a few thousand plasmids are appended as the sequence shifts from a more open pan-genome to a more closed pan-genome.

To assess the representation of each of the plasmids within the newly constructed pan-genome sequences, we did pairwise alignment using MUMmer. We used the plasmid pan-genome sequences as the reference and all the plasmids from the two sets of plasmids as query sequences. The percentage of a query sequence aligned to the pan-genome sequence is a good estimator for evaluating how well the query sequence was included as part of the corresponding pan-genome sequence. Figure 3a shows that, for most of the synthetic plasmids, more than 75% of the DNA sequence aligns to the synthetic plasmid pan-genome sequence. On the other hand, when natural plasmids are aligned to this same synthetic plasmid pan-genome in Fig. 3b, most of the natural plasmids have below 5% of their DNA aligning well. In Fig. 3c and Fig. 3d, we perform this same experiment but we align synthetic and natural plasmids against the natural plasmid pan-genome. Again, most of the natural plasmids are well aligned to the corresponding natural pan-genome sequence with more than 75% of the DNA aligning to the pan-genome in Fig. 3c. On the other hand, smaller portions of synthetic plasmids align well to the natural plasmid pan-genome sequence in Fig. 3d. The alignment percentages for synthetic plasmids range mainly from 25% to 75%

To evaluate the utility of natural and synthetic plasmid pan-genomes for differentiating natural and synthetic plasmids, we randomly selected 1000 synthetic plasmids (submitted to Addgene from January 2019 to June 2019) and 1000 natural plasmids from [9]. None of these plasmids were used in the pan-genome construction steps. As such, we used these 2000 total plasmids as a validation test data set. Our classification was performed as follows: Given an unclassified plasmid p , if its pairwise alignment percentage of p compared to the synthetic pan-genome SPG is above the threshold t AND its pairwise alignment percentage against the natural pan-genome NPG is below the threshold t , we classified p as a synthetic plasmid (and vice versa). Note: if p has alignment percentage above or below the threshold t for both synthetic and natural plasmid pan-genomes, we leave the sequence unclassified. Table 3 indicates that by mapping to the SPG and NPG , an unknown plasmid can be differentiated between being a synthetic or natural plasmid with high specificity. For synthetic plasmid classification, a high threshold yields high sensitivity. On the other hand, natural plasmid categorization has a high sensitivity with a low threshold.

To investigate the impact of the order of input sequences on building the pan-genome, we created a synthetic pan-genome using full synthetic plasmids with the input order starting from the shortest to the longest sequence. All the other parameters remained the same. The total number of nucleotides of the full synthetic plasmid pan-genome with this reverse input order is 6,585,872 bp. This is 1,721,198 bp shorter than the pan-genome built starting from the longest to the shortest sequence. The average pairwise alignment percentage of a plasmid against the initial pan-genome and the pan-genome with reversed input order are 91.52 ± 11.63 % and 89.12 ± 13.656 % respectively. Pairwise alignment results indicate that the alignments against the pan-genome with reversed input order are slightly worse than that against the initial synthetic plasmid pan-genome, but most of the synthetic plasmids still have more than 75% alignment against the full synthetic pan-genome.

In addition, we analyzed the number of synthetic plasmids that aligned to each nucleotide of the two pan-genome sequences. When aligning the plasmids against the synthetic pan-genome, most of the plasmids were mapped at the beginning of the synthetic plasmid pan-genome sequence (results not shown). In other words, the start of the synthetic plasmid pan-genome captures most of the nucleotide sequences shared by synthetic plasmids. The fragment with the highest number of aligned plasmids, which started at position 6077 in the synthetic plasmid pan-genome sequence, is annotated by *prokka* [26] as gene *bla* with protein product of “Beta-lactamase TEM precursor”. The results of aligning synthetic plasmids

■ **Table 3** Differentiation of natural and synthetic plasmids through pan-genome alignment. We classify 2000 new plasmids as either synthetic or natural plasmids based on alignment percentage against synthetic and natural plasmid pan-genomes. We calculated the sensitivity, specificity and F-score for each result.

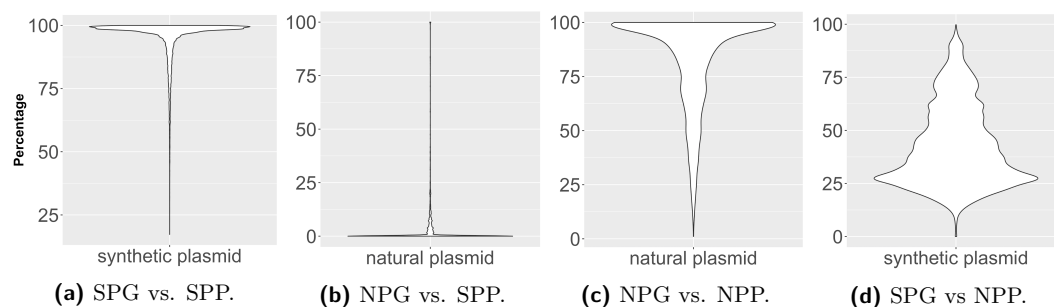
Threshold (t) (Alignment %)	Synthetic Plasmids			Natural Plasmids		
	Sensitivity	Specificity	F-score	Sensitivity	Specificity	F-score
40	0.579	0.998	0.73	0.767	1	0.868
60	0.775	0.999	0.87	0.683	0.999	0.811
80	0.821	0.999	0.90	0.536	0.998	0.697
85	0.773	0.999	0.872	0.502	0.995	0.667

and natural plasmids against the natural pan-genome are shown in Fig. 4. Specifically, the natural plasmid pan-genome sequence also includes some synthetic plasmid fragments (see Fig. 4a). Among the fragments with more than 10,000 plasmids mapped to them, two were assigned functions by *prokka*, with the rest having no results. Of these two fragments, one started at position 95,790,389 and was annotated as *lacZ* Beta-galactosidase enzyme; the other started at position 160,749,886 and was annotated as gene *bla* (Beta-lactamase). Both *lacZ* and *bla* were involved in one of the first minimal fully synthetic plasmids [15, 32], and known to be ubiquitous to the synthetic plasmidome.

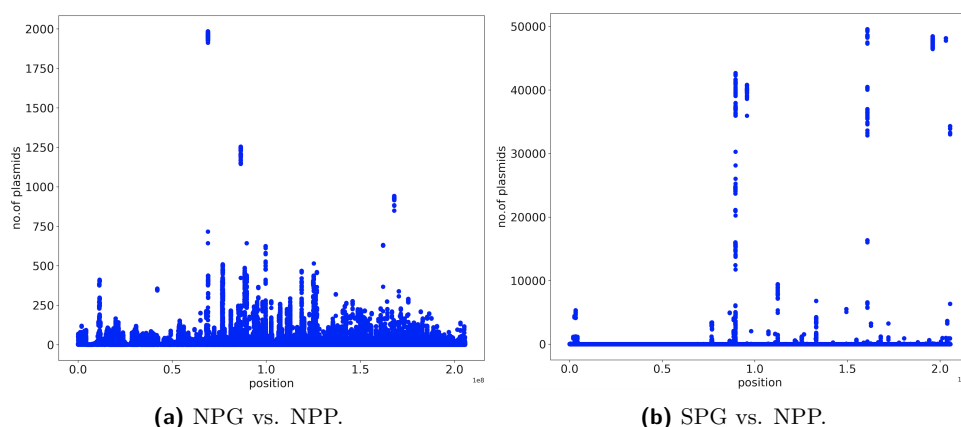
4 Discussion and Conclusions

In this paper, we introduced a novel pipeline for building pan-genomes, called *plaster*, which takes advantage of closed nature of pan-genomes when possible and displays an order of magnitude improvements to the pan-genome construction speed when compared with the fastest existing method in seq-seq-pan. For instance, given a new sequence, it can rapidly update the pan-genome sequence within 0.01s for an already existing pan-genome built from 1000 plasmids. Using *plaster*, to the best of our knowledge we constructed the first ever pan-genomes for natural and synthetic plasmids. These pan-genomes reflect the core sequences as well as the sequence diversity that exists among sequenced natural and engineered plasmids.

Alongside these newly created pan-genomes, we presented a novel technique that serves as a first step in inferring whether a plasmid is natural or synthetic. The previous work of [19] used machine learning to attempt to infer labs of origin for synthetic DNA. One common



■ **Figure 3** Alignment percentages for naturally occurring plasmids vs synthetic plasmids comparisons. (a) SPG=Synthetic plasmid genomes, (b) SPP=Synthetic plasmid pan-genome, (c) NPG=Natural plasmid genomes, NPP=Natural plasmid pan-genome, (d) SPG=Synthetic plasmid genome.



■ **Figure 4** Natural plasmid genome alignment (a) , and synthetic plasmid genome alignment (b), with respect to the Natural plasmid pan-genome. x-axis represents the position in the pan-genome (1 to 205 Mbp), y-axis represents the number of plasmids that align to a given position in the natural plasmid pan-genome.

disadvantage of machine learning approaches, in particular neural networks, suffer from poor explainability due to their black box nature [11]. Our approach presented here performs rapid pairwise sequence alignment of the query sequence against the natural and synthetic plasmid pan-genome and, as such, provides a white-box approach that allows the user to directly query the regions that are shared between natural and synthetic plasmids, specific to a given lab-of-origin, as well as what regions can be used to differentiate plasmids that have undergone human engineering [1]. Given a synthetic plasmid, our pipeline could be used to recover the full set of structural variations to fully categorize why it was determined to be synthetic as well as what engineering it has undergone.

There are several areas for future research left open by this work. As mentioned, alignment against pan-genomes can yield a full set of differences between a suspected synthetic plasmid and all natural and synthetic nucleotides contained in the pan-genomes. To investigate new ways to infer a possible lab of origin, the synthetic pan-genome could include labels for all of its contained variation and the lab(s) where that variation has been seen before. If a machine learning approach is ultimately preferred and is the most accurate for determining an origin lab or discriminating natural versus synthetic, the set of all mutations and structural variations for a given plasmid will still be crucial to have in the set of features for the final inference procedure. Functional annotations could also be added to determine what a particular lab was trying to achieve with the particular structural variations and mutations that they introduced into a plasmid. Investigations into fundamental improvements to the *plaster* method also remain as future work. For instance, an arbitrary choice was made to use variations greater than 50 bp when adding new nucleotides to the pan-genome. There will be pros and cons for varying this value to higher and lower values. This minimum alignment length could also be tuned depending on the final purpose of constructing the pan-genome, for instance for determining a lab of origin for synthetic plasmids. Tweaks to this value, as well as other algorithmic improvements, should push these alignments towards all being 100 percent alignment as well as aiding in the following step of differentiating natural and synthetic plasmids.

References

- 1 Jonathan E Allen, Shea N Gardner, and Tom R Slezak. DNA signatures for detecting genetic engineering in bacteria. *Genome biology*, 9(3):R56, 2008.
- 2 Lauren Brooks, Mo Kaze, and Mark Siström. A Curated, Comprehensive Database of Plasmid Sequences. *Microbiol Resour Announc*, 8(1):e01325–18, 2019.
- 3 Hans Bügl, John P Danner, Robert J Molinari, John T Mulligan, Han-Oh Park, Bas Reichert, David A Roth, Ralf Wagner, Bruce Budowle, Robert M Scripp, et al. DNA synthesis and biological security. *Nature biotechnology*, 25(6):627, 2007.
- 4 Jean Cury, Pedro H Oliveira, Fernando de la Cruz, and Eduardo PC Rocha. Host Range and Genetic Plasticity Explain the Coexistence of Integrative and Extrachromosomal Mobile Genetic Elements. *Molecular biology and evolution*, 35(9):2230–2239, 2018.
- 5 Robert C Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, 2010.
- 6 Mark Eppinger, Talima Pearson, Sara SK Koenig, Ofori Pearson, Nathan Hicks, Sonia Agrawal, Fatemeh Sanjar, Kevin Galens, Sean Daugherty, Jonathan Crabtree, et al. Genomic epidemiology of the Haitian cholera outbreak: a single introduction followed by rapid, extensive, and continued spread characterized the onset of the epidemic. *MBio*, 5(6):e01721–14, 2014.
- 7 Corinna Ernst and Sven Rahmann. PanCake: a data structure for pangenomes. In *German Conference on Bioinformatics 2013*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
- 8 Derrick E Fouts, Lauren Brinkac, Erin Beck, Jason Inman, and Granger Sutton. PanOCT: automated clustering of orthologs using conserved gene neighborhood for pan-genomic analysis of bacterial strains and closely related species. *Nucleic acids research*, 40(22):e172–e172, 2012.
- 9 Valentina Galata, Tobias Fehlmann, Christina Backes, and Andreas Keller. PLSDB: a resource of complete bacterial plasmids. *Nucleic acids research*, 47(D1):D195–D202, 2018.
- 10 Chris D Greenman, Erin D Pleasance, Scott Newman, Fengtang Yang, Beiyuan Fu, Serena Nik-Zainal, David Jones, King Wai Lau, Nigel Carter, Paul AW Edwards, et al. Estimation of rearrangement phylogeny for cancer genomes. *Genome research*, 22(2):346–361, 2012.
- 11 Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):93, 2018.
- 12 Finbarr Hayes. The function and organization of plasmids. In *E. coli Plasmid Vectors*, pages 1–17. Springer, 2003.
- 13 Melanie Herscovitch, Eric Perkins, Andy Baltus, and Melina Fan. Addgene provides an open forum for plasmid sharing. *Nature biotechnology*, 30(4):316, 2012.
- 14 Christine Jandrasits, Piotr W Dabrowski, Stephan Fuchs, and Bernhard Y Renard. seq-seq-pan: Building a computational pan-genome data structure on whole genome alignment. *BMC genomics*, 19(1):47, 2018.
- 15 Wlodek Mandeckci, Mark A Hayden, Mary Ann Shallcross, and Elizabeth Stotland. A totally synthetic plasmid for general cloning, gene expression and mutagenesis in *Escherichia coli*. *Gene*, 94(1):103–107, 1990.
- 16 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(1):e1005944, 2018.
- 17 Tobias Marschall, Manja Marz, Thomas Abeel, Louis Dijkstra, Bas E Dutilh, Ali Ghaffaari, Paul Kersey, Wigard P Kloosterman, Veli Makinen, Adam M Novak, et al. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018.
- 18 Elizabeth Anne McMillan, Sushim K Gupta, Laura Williams, Thomas Jové, Lari M Hiott, Tiffanie A Woodley, John B Barrett, Charlene Renee Jackson, Jamie L Waslienko, Mustafa Simmons, et al. Antimicrobial Resistance Genes, Cassettes, and Plasmids present in *Salmonella enterica* associated with US Food Animals. *Frontiers in microbiology*, 10:832, 2019.

- 19 Alec AK Nielsen and Christopher A Voigt. Deep learning to predict the lab-of-origin of engineered DNA. *Nature communications*, 9(1):3135, 2018.
- 20 Teresa Nogueira, Daniel J Rankin, Marie Touchon, François Taddei, Sam P Brown, and Eduardo PC Rocha. Horizontal gene transfer of the secretome drives the evolution of bacterial cooperation and virulence. *Current Biology*, 19(20):1683–1691, 2009.
- 21 Tuukka Norri, Bastien Cazaux, Dmitry Kosolobov, Veli Mäkinen, et al. Minimum Segmentation for Pan-genomic Founder Reconstruction in Linear Time. In *18th International Workshop on Algorithms in Bioinformatics (WABI 2018)*. Schloss Dagstuhl Leibniz Center for Informatics, 2018.
- 22 Ryan S Noyce, Seth Lederman, and David H Evans. Construction of an infectious horsepox virus vaccine from chemically synthesized DNA fragments. *PLoS one*, 13(1):e0188453, 2018.
- 23 Nuala A O’Leary, Mathew W Wright, J Rodney Brister, Stacy Ciuffo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, et al. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, 44(D1):D733–D745, 2015.
- 24 Andrew J Page, Carla A Cummins, Martin Hunt, Vanessa K Wong, Sandra Reuter, Matthew TG Holden, Maria Fookes, Daniel Falush, Jacqueline A Keane, and Julian Parkhill. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, 31(22):3691–3693, 2015.
- 25 Adam M Phillippy, Michael C Schatz, and Mihai Pop. Genome assembly forensics: finding the elusive mis-assembly. *Genome biology*, 9(3):R55, 2008.
- 26 Torsten Seemann. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*, 30(14):2068–2069, 2014.
- 27 Hervé Tettelin, Vega Massignani, Michael J Cieslewicz, Claudio Donati, Duccio Medini, Naomi L Ward, Samuel V Angiuoli, Jonathan Crabtree, Amanda L Jones, A Scott Durkin, et al. Genome analysis of multiple pathogenic isolates of *Streptococcus agalactiae*: implications for the microbial “pan-genome”. *Proceedings of the National Academy of Sciences*, 102(39):13950–13955, 2005.
- 28 Harry A Thorpe, Sion C Bayliss, Laurence D Hurst, and Edward J Feil. Comparative analyses of selection operating on nontranslated intergenic regions of diverse bacterial species. *Genetics*, 206(1):363–376, 2017.
- 29 Harry A Thorpe, Sion C Bayliss, Samuel K Sheppard, and Edward J Feil. Piggy: a rapid, large-scale pan-genome analysis tool for intergenic regions in bacteria. *Gigascience*, 7(4):giy015, 2018.
- 30 Esko Ukkonen. Finding founder sequences from a set of recombinants. In *International Workshop on Algorithms in Bioinformatics*, pages 277–286. Springer, 2002.
- 31 George Vernikos, Duccio Medini, David R Riley, and Herve Tettelin. Ten years of pan-genome analyses. *Current opinion in microbiology*, 23:148–154, 2015.
- 32 Barry L Wanner. Molecular cloning of Mu d (bla lacZ) transcriptional and translational fusions. *Journal of bacteriology*, 169(5):2026–2030, 1987.
- 33 Tom A Williams, Peter G Foster, Cymon J Cox, and T Martin Embley. An archaeal origin of eukaryotes supports only two primary domains of life. *Nature*, 504(7479):231, 2013.
- 34 Derrick E Wood, Henry Lin, Ami Levy-Moonshine, Rajiswari Swaminathan, Yi-Chien Chang, Brian P Anton, Lais Osmani, Martin Steffen, Simon Kasif, and Steven L Salzberg. Thousands of missed genes found in bacterial genomes and their analysis with COMBRES. *Biology direct*, 7(1):37, 2012.
- 35 Yongbing Zhao, Jiayan Wu, Junhui Yang, Shixiang Sun, Jingfa Xiao, and Jun Yu. PGAP: pan-genomes analysis pipeline. *Bioinformatics*, 28(3):416–418, 2011.

Rapidly Computing the Phylogenetic Transfer Index

Jakub Truskowski¹ 

LIRMM, CNRS, Université Montpellier, Montpellier, France
jakub.truskowski@lirmm.fr

Olivier Gascuel

Unité Bioinformatique Evolutive, Département de Biologie Computationnelle, USR 3756,
Institut Pasteur et CNRS, Paris, France
LIRMM, CNRS, Université Montpellier, Montpellier, France
olivier.gascuel@pasteur.fr

Krister M. Swenson 

LIRMM, CNRS, Université Montpellier, Montpellier, France
swenson@lirmm.fr

Abstract

Given trees T and T_o on the same taxon set, the *transfer index* $\phi(b, T_o)$ is the number of taxa that need to be ignored so that the bipartition induced by branch b in T is equal to some bipartition in T_o . Recently, Lemoine et al. [13] used the transfer index to design a novel bootstrap analysis technique that improves on Felsenstein's bootstrap on large, noisy data sets. In this work, we propose an algorithm that computes the transfer index for all branches $b \in T$ in $O(n \log^3 n)$ time, which improves upon the current $O(n^2)$ -time algorithm by Lin, Rajan and Moret [14]. Our implementation is able to process pairs of trees with hundreds of thousands of taxa in minutes and considerably speeds up the method of Lemoine et al. on large data sets. We believe our algorithm can be useful for comparing large phylogenies, especially when some taxa are misplaced (e.g. due to horizontal gene transfer, recombination, or reconstruction errors).

2012 ACM Subject Classification Applied computing → Bioinformatics; Theory of computation → Design and analysis of algorithms

Keywords and phrases large phylogenies, bootstrap analysis, tree comparison, data structures on trees

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.20

Acknowledgements We would like to thank Jeet Sukuraman for his help with the Dendropy package.

1 Introduction

The need to compare phylogenetic trees arises in many contexts in computational biology. In bootstrap analysis [11], trees inferred from bootstrapped data sets are compared to the tree inferred from the original data set (known as the *reference tree*) in order to evaluate the statistical support for every branch in the inferred topology. Distance measures between phylogenies, such as the Robinson-Foulds distance [9] or quartet distance [6], are also frequently used to evaluate the accuracy of inference methods by comparing the trees they produce with the “ground truth” trees that were used to simulate the data set. Finally, distance measures between phylogenetic trees allow researchers to cluster trees inferred from different genes to reveal groups of genes with similar patterns of evolution due to horizontal gene transfer, duplication, loss, or recombination [12].

¹ Present address: Borealis AI, Waterloo, Ontario, Canada



The most popular approach to comparing phylogenies compares the sets of splits (*bipartitions*) induced by the two trees. In the bootstrap setting, each branch in the reference tree is assigned a *bootstrap score* which is the fraction of bootstrap trees that contain a branch inducing the same split as the branch in the reference tree. Higher scores imply more confidence in the inferred branches. This is known as Felsenstein’s bootstrap (FBP). When the goal is to compare the global similarity of a pair of trees, a related approach is to compute the number of splits that are found in one of the trees but not both. This is known as the Robinson-Foulds (RF) distance and is widely used for comparing phylogenies. The RF distance can be computed in $O(n)$ time thanks to an algorithm by Day [9].

Both FBP and the RF distance are widely used due to their low computational cost and ease of interpretation. However, these approaches become inadequate when dealing with large, noisy data sets where large splits are unlikely to be recovered exactly. For example, if the data set contains a small number of recombinant sequences, the location of those sequences may vary between the bootstrap trees and the reference tree. Even a small number of such sequences can impact many splits in the tree, leading to low bootstrap values across the tree, despite the fact that the location of most sequences is strongly supported by the data. This problem is exacerbated in large trees, where the risk of including a rogue taxon is elevated. To illustrate this problem, Lemoine *et al.* [13] showed that trees built on a random sampling of HIV strains have generally higher bootstrap values than a tree built on the full set of strains.

Recently, Lemoine *et al.* [13] addressed this shortcoming of FBP by introducing a new bootstrap procedure based on finding similar, rather than identical splits in the bootstrap trees. For each split b in the reference tree and each bootstrap tree T_o , their method computes the *transfer index* $\phi(b, T_o)$ between the split and the bootstrap tree, which is the minimum number of taxa that need to be removed from the taxon set for b to equal some split b_o in T_o . The bootstrap support value assigned to b is then 1.0 minus its transfer index normalized by the size (minus 1.0) of the smaller of the bipartition sets induced by b , averaged over all bootstrap trees (see Davila Felipe *et al.* [8] for the statistical justification of this formula). Lemoine *et al.* show that this approach is considerably more robust to the presence of unstable taxa. The transfer index is computed using a quadratic-time algorithm due to Lin, Rajan and Moret [14], which is efficient for moderate-sized data sets, but becomes less practical for trees containing thousands or tens of thousands of taxa. In contrast, modern tree-building methods can reconstruct trees with hundreds of thousands of taxa [16, 4], which creates a need for scalable post-processing and analysis tools.

In this work, we present a fast exact algorithm for computing the transfer index for all edges in the reference tree with respect to a bootstrap tree. Our algorithm runs in $O(n \log^3 n)$ time where n is the number of taxa in the data set, which is considerably faster than the $O(n^2)$ required by Lin, Rajan and Moret [14]. Our prototype Python implementation [1] is able to process pairs of trees with tens of thousands of taxa in a matter of seconds on a standard laptop computer. Our C implementation for balanced trees [2] enables us to carry out the analyses of Lemoine *et al.* at least an order of magnitude faster, and makes it possible to compute the transfer index on data sets with tens or hundreds of thousands of sequences. Computations that took nearly nine hours now take about ten minutes. Pairwise comparisons of trees with hundreds of thousands of taxa can be completed within minutes.

2 Preliminaries

We consider binary trees with n leaves uniquely labeled from a set L of size n . Take a tree T . A branch $b \in E(T)$ can be described by the bipartition $\{A, B\}$ that it induces on L when removed from the tree ($A \cup B = L$). For $L' \subseteq L$, define $b|L' = \{A \cap L', B \cap L'\}$.

Consider two leaf-labeled binary trees T and T_o with branches $b \in E(T)$ and $b_o \in E(T_o)$. The *transfer distance* $\delta(b, b_o)$ is the number of leaves that must be ignored so that b and b_o are equal

$$\delta(b, b_o) = n - \max(\{|L'| \mid b|L' = b_o|L'\})$$

The *transfer index* for b with respect to T_o is the minimum transfer distance over all possible $b_o \in E(T_o)$:

$$\phi(b, T_o) = \min(\{\delta(b, b_o) \mid b_o \in T_o\})$$

This work is about computing $\phi(b, T_o)$ for all $b \in E(T)$. We first describe an algorithm for a special case of the problem, corresponding to situations where T_o is a balanced tree. We then present a modification of the algorithm that enables us to process general trees efficiently. Finally, we evaluate the performance of our algorithm on simulated and empirical data sets and discuss directions for future research.

3 An algorithm for balanced trees

In this section, we describe an efficient algorithm to compute $\phi(b, T_o)$ for all $b \in E(T)$ when T_o is a balanced tree. For the purposes of this manuscript, we say that a tree T_o is *balanced* if there exists a node $r \in V(T_o)$ (which we call the *root*) such that the path from r to any leaf of T_o contains at most $C \log n$ edges for some constant C . Under this assumption, our algorithm can compute $\phi(b, T_o)$ for all branches $b \in E(T)$ in $O(n \log^3 n)$ time. Balanced trees are common in phylogenetics, as both the Kingman coalescent (standard population genetics model) and Yule (standard speciation model) trees are expected to be balanced [18].

For convenience, we will work with transfer distances between rooted subtrees. For a node $u \in V(T)$, let $L(u)$ be the set of leaves descendant from u . The *rooted transfer distance* is

$$\delta_r(v, u) = |L(v) \cup L(u)| - |L(v) \cap L(u)|$$

for $v \in V(T_o)$ and $u \in V(T)$. The (unrooted) transfer distance between the corresponding splits b_v and b_u is then

$$\delta(b_v, b_u) = \min(\{\delta_r(v, u), n - \delta_r(v, u)\}). \tag{1}$$

Let T be a rooted binary tree. For any node u in $V(T)$, let $c_h(u)$ and $c_l(u)$ be the children of u such that $|L(c_h(u))| \geq |L(c_l(u))|$. We call $c_h(u)$ and $c_l(u)$ the *heavy* and *light* child of u , respectively. We refer to the edge $(u, c_h(u))$ as the *heavy edge*. If both children of u have an equal number of leaf descendants, we pick the heavy child arbitrarily.

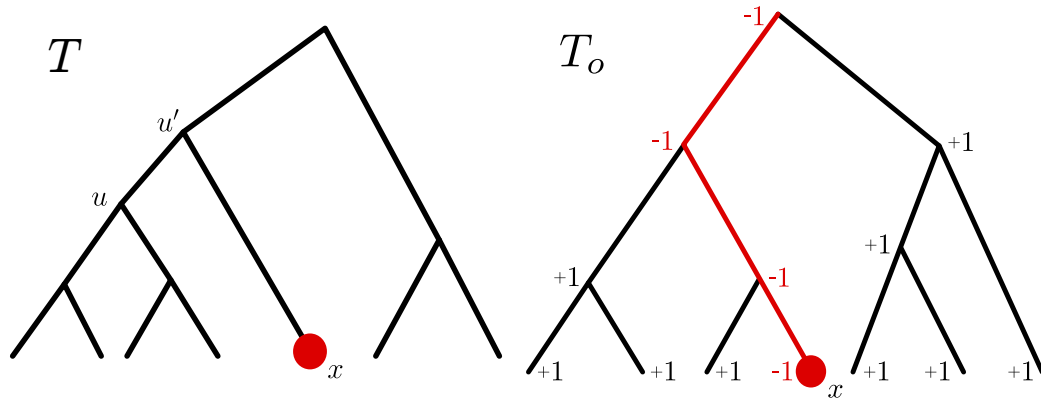
The main idea of the algorithm is to exploit the structure of both T and T_o when computing the transfer index for subsequent splits. Suppose that T contains two vertices u, u' such that $L(u') = L(u) \cup \{x\}$ for some leaf x . Unsurprisingly, we can show that the transfer distance values $\delta_r(v, u)$ and $\delta_r(v, u')$ differ by at most 1.

► **Lemma 1.** *Let u and u' be nodes in $V(T)$ such that $L(u') = L(u) \cup \{x\}$ for some leaf x . Let $P = v_1, \dots, v_k$ be the path from x to r in T_o , where $v_1 = x$ and $v_k = r$. Then, for any $1 \leq i \leq k$*

$$\delta_r(v_i, u') = \delta_r(v_i, u) - 1,$$

and for any $v \notin P$

$$\delta_r(v, u') = \delta_r(v, u) + 1.$$



■ **Figure 1** The difference between the rooted transfer distance to u and u' , for all nodes in T_o . The rooted transfer distance decreases for all nodes on the path from x to the root and increases for all other nodes.

Proof. Any node v on the path between r and x in T_o has x as its descendant, so $\delta_r(v, u') = \delta_r(v, u) - 1$ as x does not have to be removed from $L(v)$ to obtain $L(u')$, but it has to be removed to obtain $L(u)$. The opposite holds for nodes outside the path from r to x , so the distance increases by one – see Figure 1. ◀

The above lemma suggests a strategy for efficiently computing $\delta_r(v, u')$ from $\delta_r(v, u)$. For each node $v \in V(T_o)$, we will maintain a variable $D[v]$ which, when combined with a global counter, maintains the invariant $\delta_r(v, u) = D[v] + \text{counter}$. Algorithm *AddLeaf* updates D when moving from u to u' . The global counter is incremented, effectively increasing the transfer distance for all nodes. To compensate for this, each D along the path from x to the root is decreased by 2, thus maintaining the invariant.

■ **Algorithm 1** AddLeaf(x).

```

global counter
 $v \leftarrow x$  {Note:  $x \in V(T_o)$ .}
while  $v \neq r$  do
     $D[v] \leftarrow D[v] - 2$ 
     $v \leftarrow v.\text{parent}$ 
end while
counter  $\leftarrow$  counter + 1
    
```

■ **Algorithm 2** RemoveLeaf(x).

```

global counter
 $v \leftarrow x$  {Note:  $x \in V(T_o)$ .}
while  $v \neq r$  do
     $D[v] \leftarrow D[v] + 2$ 
     $v \leftarrow v.\text{parent}$ 
end while
counter  $\leftarrow$  counter - 1
    
```

Finding the node v that minimizes $\delta_r(v, u)$ can be achieved by using a dynamic data structure similar to a heap [7]. The total running time of *AddLeaf* is $O(\log^2 n)$ for balanced trees as the main loop is executed $O(\log n)$ times and each update of the structure takes $O(\log n)$ time.

When $L(u')$ differs from $L(u)$ by more than one leaf, we update D by calling *AddLeaf* for all leaves in $L(u') - L(u)$.

We now present the complete algorithm as Algorithm 3. First, we initialize the D values by computing the rooted transfer distance from the empty subtree to every subtree in T_o . The rooted distance from the empty subtree equals the number of leaves in $L(v)$. Then, we choose a leaf x in T and traverse its ancestors, calling *AddLeaf* on leaves descended from nodes off the path from x to r . To ensure that *AddLeaf* is called a limited number of times

on a leaf, we terminate the traversal when the current node is the light child of its parent. In that case, we undo the leaf additions using the routine *RemoveLeaf* and restart the process from a new leaf.

■ **Algorithm 3** ComputeTransferIndices(T_o, T).

```

Set  $D[v] \leftarrow |L(v)|, \forall v \in V(T_o)$ 
 $counter \leftarrow 0$ 
for all  $x \in L(T)$  do
   $curNode \leftarrow x$ 
  AddLeaf( $x$ )
  while  $curNode \neq r$  do
    if  $curNode$  is the heavy child of its parent then
      for all  $y \in L(sibling(curNode))$  do
        AddLeaf( $y$ )
      end for
       $\phi(b_{parent(curNode)}, T_o) \leftarrow \min_{v \in V(T_o)} (\{D[v] + counter, n - D[v] - counter\})$ 
       $curNode \leftarrow parent(curNode)$ 
    else
      for all  $y \in L(curNode)$  do
        RemoveLeaf( $y$ )
      end for
      break
    end if
  end while
end for

```

We can prove the following property.

► **Lemma 2.** *The number of calls made to *AddLeaf* during the execution of *ComputeTransferIndices* is at most $n \log_2 n$.*

Proof. Let x be a leaf in T . The number of times *AddLeaf* is called with x given as the argument is at most the number of nodes v on the path from x to the root such that $|L(v)| \leq \frac{1}{2}|L(parent(v))|$. There can be at most $\log_2 n$ such nodes since $|L(r)| = n$ and $|L(x)| = 1$. We obtain the result by summing over all leaves in T . ◀

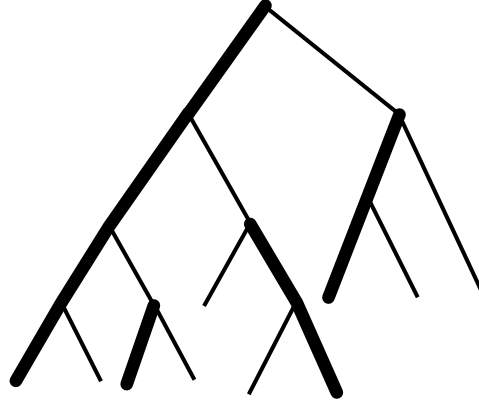
Since the running time of *AddLeaf* is $O(\log^2 n)$, it follows that the total running time of the algorithm is $O(n \log^3 n)$.

4 General Trees

4.1 The data structure

The performance of Algorithm 3 crucially depends on the height of the phylogeny. When T_o is a caterpillar tree and T is balanced, the running time of the algorithm can be as bad as $O(n^2 \log^2 n)$. In order to derive a fast algorithm for general trees, we need a data structure that would allow us to efficiently navigate splits in the tree, regardless of the topology. Our approach is based on *heavy path decompositions*, originally introduced by Sleator and Tarjan [17].

Recall that a heavy child is a node that has at least as many descendant leaves as its sibling, and that a heavy edge is an edge that connects a heavy child to its parent. A *heavy path* is a sequence of nodes v_1, \dots, v_k such that (v_i, v_{i+1}) is a heavy edge for all $1 \leq i \leq k - 1$



■ **Figure 2** A phylogeny with heavy paths marked in thick lines.

(see Figure 2). A heavy path is called *maximal* if v_k is a leaf and v_1 is either the root or the light child of its parent. We let $\mathcal{P}(T)$ denote the set of maximal heavy paths in T .

► **Lemma 3.** *Let x be any leaf in T_o . The path from x to the root of T_o intersects at most $\lceil \log n \rceil + 1$ distinct maximal heavy paths.*

Proof. Let (u, v) be an edge on the path from x to the root such that u and v belong to different maximal heavy paths. Without loss of generality, we can assume that u is the parent of v . Then $|L(u)| \geq 2|L(v)|$ since otherwise (u, v) would be a heavy edge. Since $L(x) = 1$ and $L(\text{root}(T_o)) = n$, it follows that there are at most $\lceil \log n \rceil$ such edges, which gives the result. ◀

We will think of T as a collection of maximal heavy paths. For each maximal heavy path $p = v_1, \dots, v_k$, we recursively define a *path search tree* $\mathcal{S}(p)$ as follows. The root of $\mathcal{S}(p)$ is associated with p . The children of a node associated with a path $p' = v_{p'1}, \dots, v_{p'k'}$ are associated with subpaths $v_{p'1}, \dots, v_{p'\lfloor k'/2 \rfloor}$ and $v_{p'\lfloor k'/2 + 1 \rfloor}, \dots, v_{p'k'}$; we refer to them as *intervals* and write $[a, b]$ to denote both the node associated with the path a, \dots, b and the path itself. If p' has only one element, it is a leaf in $\mathcal{S}(p)$. We also write $\text{first}(p) := v_1$, $\text{last}(p) := v_k$ and $\text{path}(x)$ for the unique maximal heavy path that contains x .

For each interval $[a, b]$ we maintain variable $D[a, b]$ and maintain the invariant

$$\delta_r(v, u) = \sum_{\{[a, b] \in \mathcal{S}(p) \mid v \in [a, b]\}} D[a, b] + \text{counter} \quad (2)$$

for all $v \in V(T_o)$, where u is the node in T under consideration. Moreover, we maintain variables $\text{minval}[a, b]$ and $\text{maxval}[a, b]$ with the invariant

$$\text{minval}[a, b] = \min_{v \in [a, b]} \sum_{\{[y, z] \in \mathcal{S}(p) \mid v \in [y, z] \subseteq [a, b]\}} D[y, z] \quad (3)$$

$$\text{maxval}[a, b] = \max_{v \in [a, b]} \sum_{\{[y, z] \in \mathcal{S}(p) \mid v \in [y, z] \subseteq [a, b]\}} D[y, z] \quad (4)$$

Informally, $\text{minval}[a, b]$ is the minimum value of the rooted transfer distance between any node in $[a, b]$ and the current split in T , up to a constant. That is, the node v that minimizes $\sum_{\{[y, z] \in \mathcal{S}(p) \mid v \in [y, z] \subseteq [a, b]\}} D[y, z]$ also minimizes $\delta_r(v, u)$. In particular, if $[a, b]$ is the root node of its path search tree, we have

$$\min_{v \in [a, b]} \delta_r(v, u) = \text{minval}[a, b] + \text{counter} \quad (5)$$

Analogously, for $maxval[a, b]$ we have

$$\max_{v \in [a, b]} \delta_r(v, u) = maxval[a, b] + counter \quad (6)$$

While we are not aware of this particular data structure having been defined previously, we note that somewhat similar search tree structures on phylogenies have been developed before for large-scale phylogenetic tree reconstruction [5] and computing the quartet distance between trees [3]. Heavy path decompositions have also been introduced in database literature for computing edit distances on trees [15].

4.2 The algorithm

Algorithm 7 follows the same strategy as the algorithm for balanced trees, with a few modifications designed to accommodate potentially long paths between the root and leaves in T_o . First, we initialize the path search trees by setting $D[x, x] = |L(x)|$ for every node $x \in V(T_o)$ and $D[a, b] = 0$ for every interval with $a \neq b$. We also set $minval[a, b] = |L(b)|$, $maxval[a, b] = |L(a)|$, and $counter = 0$. It can be easily verified that the invariants given by Equations 2, 3, and 4 are satisfied.

The algorithm then repeatedly calls functions *AddLeafGeneral* and *RemoveLeafGeneral* to move between splits of T , updating the transfer distances to splits in T_o . We use path search trees together with the function *UpdatePath* to efficiently update the minimum transfer distance values within each heavy path, making use of Equations 5 and 6. Given a node x , the algorithm traverses the intervals in the path search tree of the heavy path containing x while updating D , $minval$, and $maxval$ values to reflect the fact that the distance has changed by the same amount for x and every node above it (see Figure 3).

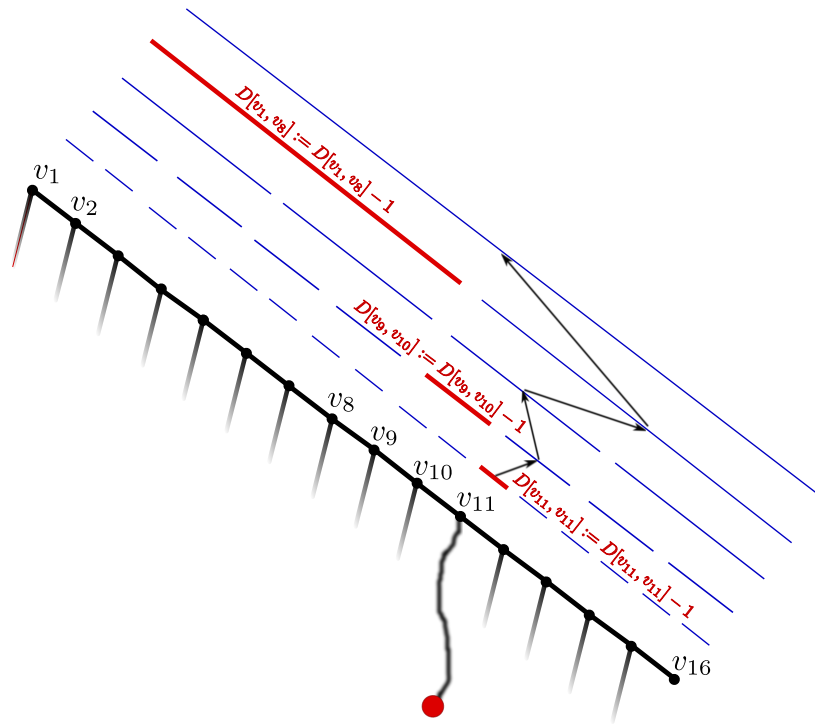
■ Algorithm 4 UpdatePath(x, d).

```

 $D[x, x] \leftarrow D[x, x] + d$ 
 $minval[x, x] \leftarrow minval[x, x] + d$ 
 $maxval[x, x] \leftarrow maxval[x, x] + d$ 
 $[a, b] \leftarrow [x, x]$ 
while  $[a, b]$  is not root interval do
   $[a_p, b_p] \leftarrow parent\_interval([a, b])$ 
  if  $b_p = b$  then
     $[a_s, b_s] \leftarrow sibling\_interval([a, b])$ 
     $D[a_s, b_s] \leftarrow D[a_s, b_s] + d$ 
     $minval[a_s, b_s] \leftarrow minval[a_s, b_s] + d$ 
     $maxval[a_s, b_s] \leftarrow maxval[a_s, b_s] + d$ 
  end if
   $minval[a_p, b_p] \leftarrow \min\{minval[a_s, b_s], minval[a, b]\} + D[a_p, b_p]$ 
   $maxval[a_p, b_p] \leftarrow \max\{maxval[a_s, b_s], maxval[a, b]\} + D[a_p, b_p]$ 
   $[a, b] \leftarrow [a_p, b_p]$ 
end while

```

► **Lemma 4.** *Each call to UpdatePath changes the value of $\sum_{\{[a, b] \in \mathcal{S}(p) \mid v \in [a, b]\}} D[a, b]$ by d for all vertices v ancestral to x (including x) in the path p . Moreover, each call preserves invariants 3 and 4.*



■ **Figure 3** Updating the D values associated with the heavy path v_1, \dots, v_{16} during the call to $UpdatePath(v_{11}, -2)$. Intervals whose D values have been updated are coloured in red.

Proof. Let y be a node in p that is above x . Let $[a_m, b_m]$ be the minimal interval in $\mathcal{S}(p)$ that contains both x and y and let $[a_l, b_l]$ and $[a_r, b_r]$ be the left and right child of $[a_m, b_m]$ in $\mathcal{S}(p)$, respectively. By the choice of $[a_m, b_m]$, x and y cannot both belong to the same child of $[a_m, b_m]$. Since y is above x in T_o , it follows that $y \in [a_l, b_l]$ and $x \in [a_r, b_r]$. We have $b_m = b_r$ by the definition of the path search tree, so $D[a_l, b_l]$ will be increased by d at the iteration of the *while* loop when $[a, b] = [a_r, b_r]$ and $minval[a_l, b_l]$ and $maxval[a_l, b_l]$ will be increased by the same amount, preserving invariants 3, 4. On the other hand, it can be easily verified that all the other variables $D[a, b]$ such that $y \in [a, b]$ will remain unchanged, since $[a_l, b_l]$ is the only interval containing y that is the left child of its parent and has a right child that contains x . ◀

■ **Algorithm 5** AddLeafGeneral(x).

```

global counter
UpdatePath( $x, -2$ )
while first(path( $x$ ))  $\neq$  root( $T_o$ ) do
     $x \leftarrow$  parent(first(path( $x$ )))
    UpdatePath( $x, -2$ )
end while
counter  $\leftarrow$  counter + 1
    
```

■ **Algorithm 6** RemoveLeafGeneral(x).

```

global counter
UpdatePath( $x, 2$ )
while first(path( $x$ ))  $\neq$  root( $T_o$ ) do
     $x \leftarrow$  parent(first(path( $x$ )))
    UpdatePath( $x, 2$ )
end while
counter  $\leftarrow$  counter - 1
    
```

► **Lemma 5.** After each call to AddLeafGeneral or RemoveLeafGeneral, Equation 2 is satisfied.

Proof. Equation 2 holds for the initial values of $D[a, b]$ and *counter*. The calls to *UpdatePath* from within *AddLeafGeneral* decreases the sum $\sum_{\{[a,b] \in \mathcal{S}(p) \mid v \in [a,b]\}} D[a, b]$ by exactly 2 for each node v on the path from x to the root of T_o . The variable *counter* increases by 1, which means that the sum

$$\sum_{\{[a,b] \in \mathcal{S}(p) \mid v \in [a,b]\}} D[a, b] + \textit{counter}$$

is decreased by 1 for each node on the path from x to the root and increased by 1 for all other nodes. This, together with Lemma 1 gives the result for *AddLeafGeneral*. The case of *RemoveLeafGeneral* is proved by analogous reasoning. ◀

▶ **Theorem 6.** *Algorithm 7 computes the transfer index for all splits in T .*

Proof. Lemma 5 and Equation 5 ensure that at each iteration of the *while* loop, the value $\textit{minval}[\textit{first}(p), \textit{last}(p)] + \textit{counter}$ is equal to the minimum value of the rooted transfer distance among the nodes in p . Using Equation 1 and taking the minimum over all the paths yields the result. ◀

■ **Algorithm 7** ComputeTransferIndicesGeneral(T_o, T).

```

Initialize  $D[., .], \textit{minval}[., .], \textit{maxval}[., .]$ .
counter  $\leftarrow 0$ 
for all  $x \in L(T)$  do
  curNode  $\leftarrow x$ 
  AddLeafGeneral( $x$ )
  while curNode  $\neq r$  do
    if curNode is the heavy child of its parent then
      for all  $y \in L(\textit{sibling}(\textit{curNode}))$  do
        AddLeafGeneral( $y$ )
      end for
       $\textit{min1} \leftarrow \min_{p \in \mathcal{P}(T_o)} \textit{minval}[\textit{first}(p), \textit{last}(p)] + \textit{counter}$ 
       $\textit{min2} \leftarrow \min_{p \in \mathcal{P}(T_o)} (n - \textit{maxval}[\textit{first}(p), \textit{last}(p)]) - \textit{counter}$ 
       $\phi(b_{\textit{parent}(\textit{curNode}), T_o}) \leftarrow \min \{\textit{min1}, \textit{min2}\}$ 
      curNode  $\leftarrow \textit{parent}(\textit{curNode})$ 
    else
      for all  $y \in L(\textit{curNode})$  do
        RemoveLeafGeneral( $y$ )
      end for
      break
    end if
  end while
end for

```

4.3 Running time analysis

The running time of Algorithm 7 is $O(n \log^3 n)$. Each call to *UpdatePath* takes at most $O(\log n)$ time. During the execution of *AddLeafGeneral* and *RemoveLeafGeneral*, *UpdatePath* is called at most $O(\log n)$ times by Lemma 3, which gives a bound of $O(\log^2 n)$ for each call to any of these functions. Finally, by Lemma 2, *AddLeafGeneral* is executed

at most $O(n \log n)$ times, which means that the overall running time is $O(n \log^3 n)$. Finding $\phi(b_{curNode}, T_o)$ can be solved efficiently using a heap at a cost of $O(\log n)$ for each call to *UpdatePath*, which increases the total running time only by a constant factor.

5 Implementations

We have two implementations currently available. A proof-of-concept implementation of Algorithm 7 was written in Python, making use of the Dendropy package [19]. This implementation is available at [1]. A definitive C implementation is a work in progress; Algorithm 3 has been integrated into the current Booster code base [2].

To obtain speedups for trees with a high degree of similarity, our Python implementation first traverses both trees and identifies, in linear time, maximal identical subtrees that occur in both trees. These trees are then replaced with single nodes whose weight equals the number of leaves in the collapsed subtree. After this pre-processing step, every *AddLeaf* and *RemoveLeaf* operation changes all the relevant counters by the weight of the node, rather than by 1. Every edge within the collapsed subtrees has transfer index equal to 0. This procedure can result in substantial speedups for highly similar trees (see next section).

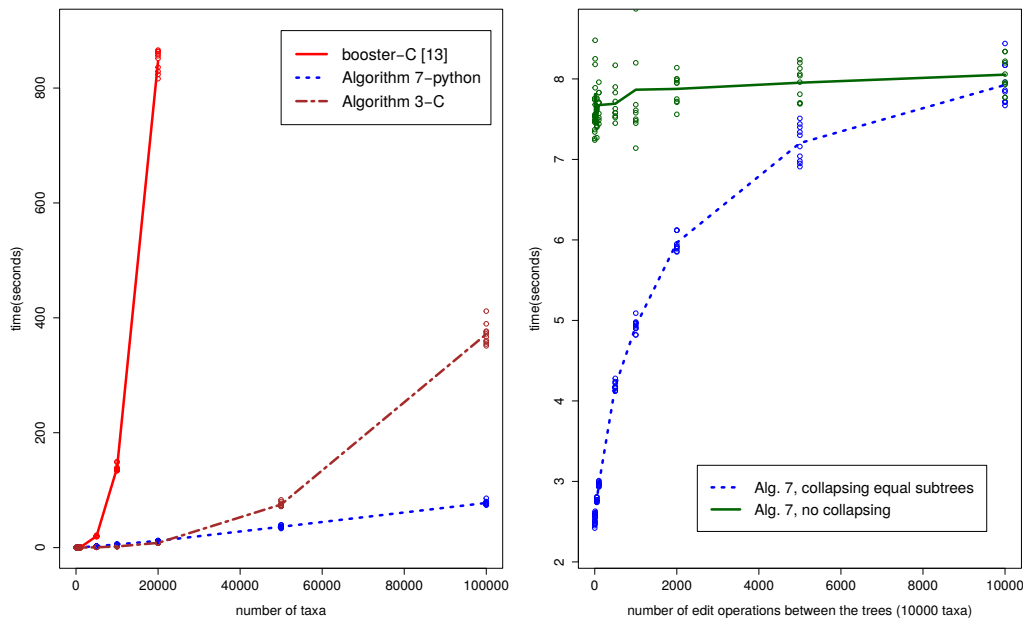
6 Experiments

We evaluated the performance of our algorithm on several simulated and empirical data sets. In all simulation experiments, we simulated pairs of trees by sampling the first tree topology uniformly at random and applying random edit operations to obtain the second tree. Each edit operation chose two random nodes such that neither node is ancestral to the other (with respect to some arbitrary rooting of the tree) and swapped the subtrees rooted at the chosen nodes. By increasing the number of applications of the edit operation, we decreased the similarity of the trees.

In the first experiment, we generated pairs of trees with sizes varying between 100 and 100000 taxa. Each pair of trees was generated using the number of edit operations equal to 0.2 times the number of taxa. Experiments were performed on a Linux laptop with 8 GB of RAM and an Intel i5 2.3 GHz processor using one thread for each run. The results are given in Figure 4(left). The running time of our Python program is close to linear in the number of taxa; even for 100000 taxa, it takes less than 100 seconds to compute the transfer indices. Our C implementation of Algorithm 3 is faster for smaller trees with up to 20000 taxa, but becomes considerably slower for very large trees. This is not surprising as trees sampled from the uniform distribution are likely to be unbalanced, as the mean diameter of a tree is $O(\sqrt{n})$ [10]. Booster is the slowest of the three algorithms and runs out of memory for trees with more than 20000 taxa.

In the second experiment, we investigated the impact of tree dissimilarity on the running time. Experiments were performed on an Intel Xeon 2.1 GHz processor with 32 GB of RAM using a single thread. We generated pairs of trees with 10000 taxa each while varying the number of edit operations from 0 to 10000. As we can see in Figure 4(right), tree dissimilarity has little, if any, effect on the running time of the algorithm when the preprocessing procedure is not applied. In contrast, collapsing identical subtrees can lead to 2- to 3-fold speedups for highly similar tree pairs.

In the final experiment, we repeated the analysis from Lemoine et al. on several HIV and mammalian data sets. Each data set contained 1000 bootstrap replicates and the number of taxa varied between 571 and 9147. The results are given in Table 1. On the largest data



■ **Figure 4** Left: The running times for our two implementations and the code by Lemoine et al. as a function of the number of taxa. Right: Running times for a pair of 10000-taxon trees as a function of the similarity of the trees.

set, our Python program completed the analysis in less than 2 hours and our C program completed the analysis in less than 12 minutes, compared to almost 9 hours for Booster, which is written in C. Smaller data sets were processed in a matter of minutes by Booster and our Python implementation, while our C implementation was at least an order of magnitude faster than Booster in all cases.

■ **Table 1** Running times for four data sets from Lemoine et al.. Our C implementation presents substantial advantages.

Data set	taxa	#bootstraps	booster-C [13]	Alg. 7-Python	Alg. 3-C
HIV/Full	9147	1000	8h51m	1h47m	12m
HIV/Medium_Sample1	571	1000	1m	3m23s	6s
HIV/Medium_Sample2	571	1000	1m	3m32s	6s
Mammals/raxml	1449	1000	10m32s	10m36s	30s

7 Conclusion

We have presented an algorithm for rapidly computing, for every split in tree T , the minimum transfer distance to the closest split in another tree T_o . The running time of the algorithm is $O(n \log^3 n)$, which is close to linear in the size of input trees. We expect to be able to remove a logarithmic factor with improved bookkeeping to maintain the minimum value of the transfer distance. Without this improvement, our prototype implementation scales almost linearly in practice, enabling us to compare pairs of trees with hundreds of thousands of taxa in a matter of minutes. This is comparable to several widely-used implementations of the RF distance, which makes our tool an interesting alternative to the RF distance for comparing large, noisy trees.

The most direct practical implication of this work is scaling up the TBE bootstrap analysis by Lemoine et al. Our method gives at least an order of magnitude speedup on all of the data sets they analyzed, and will enable the analysis of bootstrap data sets for trees of tens to hundreds of thousands of taxa within hours of CPU time.

On small and moderate-size data sets, our Python implementation is slower than the Booster software of Lemoine et al., which is likely due to their optimized implementation. Our C implementation for balanced trees always significantly out-performs Booster, however, and we are in the process of expanding it to the complete Algorithm 7.

There are several possible directions for future research. It would be interesting to see if the techniques developed here could be used to design novel methods for finding consensus trees. Another natural direction is to investigate whether global distance measures between trees, rather than splits, could be designed based on the transfer index.

References

- 1 https://bitbucket.org/thekswenson/rapid_transferindex/.
- 2 <https://github.com/thekswenson/booster>.
- 3 GS Brodal, R Fagerberg, and CNS Pedersen. Computing the quartet distance between evolutionary trees in time $O(n \log n)$. *Algorithmica*, 38(2):377–395, 2004.
- 4 Daniel G Brown and Jakub Trzuskowski. Fast phylogenetic tree reconstruction using locality-sensitive hashing. In *Algorithms in Bioinformatics*, pages 14–29. Springer, 2012.
- 5 Daniel G Brown and Jakub Trzuskowski. Fast error-tolerant quartet phylogeny algorithms. *Theoretical Computer Science*, 483:104–114, 2013.
- 6 D Bryant, J Tsang, PE Kearney, and M Li. Computing the quartet distance between evolutionary trees. In *Proceedings of SODA 2000*, pages 285–286, 2000.
- 7 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.
- 8 M Dávila Felipe, J-B Domelevo Entfellner, F Lemoine, J Trzuskowski, and O Gascuel. Distribution and asymptotic behavior of the phylogenetic transfer distance. *Journal of Mathematical Biology*, April 2019.
- 9 WHE Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of classification*, 2(1):7–28, 1985.
- 10 Péter L Erdős, Michael A Steel, László A Székely, and Tandy J Warnow. A few logs suffice to build (almost) all trees: part II. *Theoretical Computer Science*, 221(1-2):77–118, 1999.
- 11 Joseph Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4):783–791, 1985.
- 12 K Gori, T Suchan, N Alvarez, N Goldman, and C Dessimoz. Clustering genes of common evolutionary history. *Molecular biology and evolution*, 33(6):1590–1605, 2016.
- 13 F Lemoine, J-B Domelevo Entfellner, E Wilkinson, D Correia, M Dávila Felipe, T de Oliveira, and O Gascuel. Renewing Felsenstein’s phylogenetic bootstrap in the era of big data. *Nature*, 556(7702):452, 2018.
- 14 Yu Lin, Vaibhav Rajan, and Bernard ME Moret. A metric for phylogenetic trees based on matching. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(4):1014–1022, 2012.
- 15 M Pawlik and N Augsten. RTED: a robust algorithm for the tree edit distance. *Proceedings of the VLDB Endowment*, 5(4):334–345, 2011.
- 16 Morgan N Price, Paramvir S Dehal, and Adam P Arkin. FastTree 2—approximately maximum-likelihood trees for large alignments. *PloS one*, 5(3):e9490, 2010.
- 17 DD Sleator and RE Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.
- 18 Mike Steel. *Phylogeny: discrete and random processes in evolution*. SIAM, 2016.
- 19 Jeet Sukumaran and Mark T Holder. DendroPy: a Python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571, 2010.

Empirical Performance of Tree-Based Inference of Phylogenetic Networks

Zhen Cao

Department of Computer Science, Rice University, 6100 Main Street, Houston, TX 77005, USA
zhen.cao@rice.edu

Jiafan Zhu

Department of Computer Science, Rice University, 6100 Main Street, Houston, TX 77005, USA
jiafan.zhu@rice.edu

Luay Nakhleh 

Department of Computer Science, Rice University, 6100 Main Street, Houston, TX 77005, USA
nakhleh@rice.edu

Abstract

Phylogenetic networks extend the phylogenetic tree structure and allow for modeling vertical and horizontal evolution in a single framework. Statistical inference of phylogenetic networks is prohibitive and currently limited to small networks. An approach that could significantly improve phylogenetic network space exploration is based on first inferring an evolutionary tree of the species under consideration, and then augmenting the tree into a network by adding a set of “horizontal” edges to better fit the data.

In this paper, we study the performance of such an approach on networks generated under a birth-hybridization model and explore its feasibility as an alternative to approaches that search the phylogenetic network space directly (without relying on a fixed underlying tree). We find that the concatenation method does poorly at obtaining a “backbone” tree that could be augmented into the correct network, whereas the popular species tree inference method ASTRAL does significantly better at such a task. We then evaluated the tree-to-network augmentation phase under the minimizing deep coalescence and pseudo-likelihood criteria. We find that even though this is a much faster approach than the direct search of the network space, the accuracy is much poorer, even when the backbone tree is a good starting tree.

Our results show that tree-based inference of phylogenetic networks could yield very poor results. As exploration of the network space directly in search of maximum likelihood estimates or a representative sample of the posterior is very expensive, significant improvements to the computational complexity of phylogenetic network inference are imperative if analyses of large data sets are to be performed. We show that a recently developed divide-and-conquer approach significantly outperforms tree-based inference in terms of accuracy, albeit still at a higher computational cost.

2012 ACM Subject Classification Applied computing → Genomics; Applied computing → Computational biology

Keywords and phrases Phylogenetic networks, species tree, tree-based networks, multi-locus phylogeny

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.21

Funding *Luay Nakhleh*: NSF grants DBI-1355998, CCF-1302179, CCF-1514177, CCF-1800723, and DMS-1547433.

Acknowledgements The authors would like to thank Dr. Huw A. Ogilvie for his help.

1 Introduction

As evidence of reticulation (hybridization, horizontal gene transfer, etc.) in the evolutionary histories of diverse sets of species across the Tree (or, more appropriately in our context, Network) of Life continues to grow, increasingly sophisticated methods for phylogenetic



© Zhen Cao, Jiafan Zhu, and Luay Nakhleh;
licensed under Creative Commons License CC-BY

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).

Editors: Katharina T. Huber and Dan Gusfield; Article No. 21; pp. 21:1–21:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

network inference are being developed to incorporate processes such as incomplete lineage sorting (ILS), gene duplication and loss, and gene flow [20, 21, 15, 16, 24, 28, 27]. These methods, which are mostly statistical in nature, are computationally prohibitive due, in part, to the complex space of phylogenetic networks that is explored. This, in turn, has limited the applicability of such methods to data sets with small numbers of taxa, loci, and reticulation events. To ameliorate this computational challenge, a divide-and-conquer approach was recently introduced [26], where this accurate, yet computationally expensive, method could be used to infer small networks that are then merged to produce a phylogenetic network on the full data set. The accuracy of the method notwithstanding, the inference of small networks remained a computational bottleneck.

An alternative approach that could be considered would first infer an underlying “species tree”, and then augment this tree into a network by adding reticulations to it to fit the data under some criterion that incorporates reticulation. The benefit of such an approach is that it would utilize one of a wide array of species tree inference methods that are both accurate and efficient, thus drastically reducing the space of phylogenetic networks to explore by limiting them to those “based” on inferred trees. Indeed, the question of whether a species tree can be accurately inferred in the presence of reticulation has been partially explored from theoretical [12] as well as empirical [4] angles. An extensive simulation study on small data sets demonstrated the problems with inferring “the” species tree in the presence of high rates of gene flow [13]. However, a more general question to ask is whether tree inference methods can infer a tree, not necessarily the species tree (if one insists on using this designation), that can be augmented into the correct network. Along the same lines, the class of tree-based networks was introduced [6] and the set of trees that characterize a phylogenetic network in the presence of ILS was revisited [29].

In this paper, we study the performance of tree-based inference of phylogenetic networks by exploring two popular methods for inferring a start tree and two network criteria that scale to evaluating large networks. We consider synthetic networks that are generated under a birth-hybridization model and data simulated under the multispecies network coalescent [19]. We find that the concatenation method does poorly at inferring a backbone tree of the network, whereas the commonly used method ASTRAL [23] performs significantly better at this task. However, even when a correct network-backbone tree is used, augmenting such a tree into the correct network is a challenging task and results in poor accuracy under both the pseudo-likelihood and minimizing deep coalescences criteria of [21, 22]. It is worth noting that the size of phylogenetic networks we consider in our study makes use of the likelihood criterion of [20] infeasible. We demonstrate that the divide-and-conquer approach of [26] yields more accurate results, yet is orders of magnitude slower than tree-based inference. We demonstrate that combining the strengths of the two – the speed of tree-based inference and the accuracy of the divide-and-conquer approach – could provide a promising approach to large-scale network inference. Finally, our implementations of tree-based phylogenetic network inference are implemented in PhyloNet [14, 17].

2 Background

A **phylogenetic network** Ψ on a set \mathcal{X} of taxa is a rooted, directed, acyclic graph (DAG) whose leaves are bijectively labeled by \mathcal{X} . For $\Psi = (V(\Psi), E(\Psi))$, the set $V(\Psi)$ of nodes contains a root node with in-degree 0 and out-degree 2, tree nodes with in-degree 1 and out-degree 2, reticulation nodes with in-degree 2 and out-degree 1, and leaf nodes with in-degree 1 and out-degree 0. If v is a reticulation node, then the two edges incident into it

are reticulation edges, and all edges incident into tree nodes are tree edges. In a statistical setting, reticulation edges are associated with inheritance probabilities, and all edges have lengths, so that the phylogenetic network defines a distribution over gene trees under the multispecies network coalescent [20].

A **phylogenetic tree** T can be viewed as a phylogenetic network with no reticulation nodes. There is a natural relationship between a phylogenetic network and the set of trees it displays.

► **Definition 1.** *A tree T on set \mathcal{X} of taxa is displayed by a phylogenetic network Ψ (also on set \mathcal{X} of taxa) if T can be obtained from Ψ by removing a set of reticulation edges followed by forced contraction of every node v of in- and out-degree 1, where the edges (u, v) and (v, w) are replaced by a single edge (u, w) and v is deleted.*

We denote by $\mathcal{T}(\Psi)$ the set of all trees displayed by Ψ .

Several methods for inference of phylogenetic networks in the presence of incomplete lineage sorting have been devised. These include parsimony methods [18], maximum likelihood methods [19, 20], maximum pseudo-likelihood methods [21, 27], and Bayesian methods [15, 16, 24, 28]. These methods have poor scalability in terms of time and memory requirements and are applicable to very small data sets. The pseudo-likelihood methods were devised to ameliorate the problem of computing the full likelihood of phylogenetic networks, but these suffer from the large network space they need to explore.

One potential approach to tackling the computational requirements of phylogenetic network inference is based on first inferring a “species tree” and then adding a set of reticulation edges to it, in the hope of obtaining the true network. Indeed, as discussed above, the plausibility of this approach has been addressed, albeit in a limited fashion [4, 12, 6, 29, 13]. If this approach works in practice, it would tremendously reduce the phylogenetic network space to search and, consequently, scale phylogenetic network inference to much larger data sets. The goal of this work is to systematically study this approach on phylogenetic networks generated under a birth-hybridization model to assess its potential. Next, we describe our implementation of this approach.

3 Methods

Consider a data set $\mathcal{S} = (S_1, S_2, \dots, S_m)$ and $\mathcal{G} = (g_1, g_2, \dots, g_m)$, where S_i is the alignment of a set of orthologous sequences of locus i in the genomes of a set \mathcal{X} of taxa, and g_i is a rooted gene tree for locus i inferred from S_i . Throughout this work we assume that loci are independent and each locus is recombination-free.

Generally, a backbone-based approach to inferring a phylogenetic network Ψ on set \mathcal{X} of taxa follows two steps.

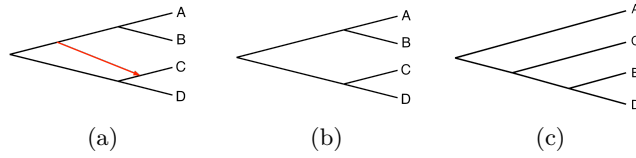
Step 1. Build a start tree T , either using the sequences \mathcal{S} directly or using the estimated gene trees \mathcal{G} .

Step 2. Augment the start tree T into a network Ψ by adding a set E_r of reticulation edges to it to optimize some criterion given either \mathcal{S} or \mathcal{G} .

A desired property of the start tree T in Step 1 is that it is a backbone of the network, that is, it is displayed by the true network so that the latter is obtainable by the backbone-based approach. This property is illustrated in Figure 1.

3.1 Step 1: Building a start tree

As we only consider reticulation and incomplete lineage sorting (ILS) in this paper (that is, we do not consider gene duplication and loss, for example), we considered two methods that are heavily used to infer a species tree when incongruence is suspected to be due to ILS. The



■ **Figure 1 From a backbone tree to a phylogenetic network.** (a) Phylogenetic network Ψ . (b) A backbone tree that is displayed by Ψ and, thus, can be augmented into the true network. (c) A tree that cannot be augmented to produce Ψ and therefore is not a backbone of the network.

first method infers a tree on the concatenation of the sequence data \mathcal{S} . For this purpose, we used IQ-TREE [9] to infer a tree from the concatenated sequences. The second method infers a species tree from individual gene trees in \mathcal{G} . For this purpose, we used ASTRAL-III [23]. Both of these methods have been shown to have good efficiency and accuracy in practice.

3.2 Step 2: Augmenting the start tree into a network

Let Ψ be a phylogenetic network (including the case where Ψ is a tree). Adding a reticulation edge to Ψ is done by selecting two edges (u, v) and (x, y) in Ψ , replacing them by (u, m) , (m, v) , (x, m') , and (m', y) , and adding an edge (m, m') . This operation results in creating a new reticulation node, m' , in the phylogenetic network Ψ and results in a network with one more reticulation node than those in Ψ . The reticulation edge can be added to initial edges or added edges. Hence, obtained networks are not limited to tree-based networks.

Consider an optimality criterion Φ that is defined on a given network Ψ and input data \mathcal{I} (\mathcal{I} could be sequences or trees, and the criterion has to be appropriate for the type of data used). Given a maximum number of reticulation nodes to consider MAX , one implementation of tree-based inference of networks follows the steps of Algorithm **StepwiseAugment**, given a start tree T .

■ **Algorithm 1** StepwiseAugment.

```

1  $\Psi^0 = T$ ;
2 for  $j = 0$  to  $(MAX - 1)$  do
3    $\Psi^{j+1} \leftarrow \Psi^j$ ;
4   foreach  $\Psi'$  obtained by adding a reticulation edge to  $\Psi^j$  do
5     if  $\Phi(\Psi', \mathcal{I})$  is better than  $\Phi(\Psi^{j+1}, \mathcal{I})$  then
6        $\Psi^{j+1} \leftarrow \Psi'$ ;
7     end
8   end
9 end
10 return  $\Psi^{MAX}$ ;
```

In other words, we exhaustively evaluate the optimal networks with $j + 1$ reticulations that can be obtained by adding a single reticulation to the optimal network found with j reticulations. It is important to highlight here that this implementation is not the most exhaustive way to augment a tree into a network with a given number of reticulations. The most exhaustive way of augmenting a tree on n leaves into a phylogenetic network with k reticulations considers $\prod_{i=0}^{k-1} \binom{2n-2+3i}{2}$ networks, which is $O(n^{2k})$ when $k \ll n$. Our implementation reduces this number to $\sum_{i=0}^{k-1} \binom{2n-2+3i}{2} = O(kn^2)$.

Another approach to augment a start tree into a network is to heuristically search for reticulation edges to add in a local search fashion. We implemented **LocalSearchAugment** which starts with a tree T and then randomly chooses one of a set of operations to apply in order to augment the current network into a new candidate network. If the resulting network has a better optimality score, it is adopted as the new current network and the search continues; otherwise, the move is rejected (or rejected with a probability) and a new candidate network is considered. The set of moves we considered consist of: (i) adding a new reticulation edge; (ii) removing one of the reticulation edges that have been added; (iii) relocating the head of an added reticulation edge; (iv) relocating the tail of an added reticulation edge; (v) reversing the direction of an added reticulation edge; (vi) replacing an added reticulation edge with a new one; and, (vii) modifying an edge parameter like the inheritance probability and branch length in the network (if criterion Φ requires such parameters). We define a probability distribution on this set of moves, and in each iteration of the local search, a move is selected randomly based on this distribution. The heuristic search applies random restarts to ameliorate the local optima issue, where each search stops when the number of consecutively rejected moves (they would be rejected because the proposed solution candidates do not improve the optimality criterion) reaches a pre-set threshold. It is important to emphasize that **LocalSearchAugment** never removes any of the edges that exist in the start tree T .

For Φ , we considered two criteria in this study: the pseudo-likelihood criterion of [21] and the minimizing deep coalescence (MDC) criterion of [18]. Both of these criteria use gene trees as input data. The likelihood criterion of [20] is infeasible to compute on the data sets we consider in this study.

3.3 Evaluating the inferred trees and networks

While several dissimilarity measures exist for comparing networks and, by extension, trees and networks, e.g., [2, 3, 8], these measures are very sensitive to the misplacement of reticulation nodes, even in cases when networks agree on much of their structure. Therefore, in this paper, we used two measures of dissimilarity for tree-to-network and network-to-network comparisons.

Letting RF be the Robinson-Foulds distance [11] (the size of the symmetric difference between the two trees, divided by 2), the dissimilarity between a tree T and a network Ψ is:

$$D(T, \Psi) = \min_{T' \in \mathcal{T}(\Psi)} \text{RF}(T, T'). \quad (1)$$

For network-to-network comparison, we can extend the notion of displaying to networks: We say phylogenetic network Ψ' is displayed by phylogenetic network Ψ if Ψ' can be obtained by removing a set of reticulation nodes and applying forced contraction to Ψ (but unlike in the case of trees, some reticulation nodes remain in the network). We denote by $\mathcal{N}(\Psi)$ the set of all networks displayed by Ψ (obviously, $\mathcal{T}(\Psi) \subseteq \mathcal{N}(\Psi)$). We denote by $r(\Psi)$ the number of reticulations and $t = 2|\mathcal{N}| - 1$ the number of nodes for any displayed tree of Ψ . Let Ψ_t and Ψ_i be true and inferred networks, respectively, and $\Psi'_t \in \mathcal{N}(\Psi_t)$ and $\Psi'_i \in \mathcal{N}(\Psi_i)$ be the two displayed networks that have the smallest distance $d(\Psi'_t, \Psi'_i)$, as computed by [8], over all displayed networks of the two networks Ψ_t and Ψ_i , and $D(\Psi_t, \Psi_i) = \min_{\Psi'_t \in \mathcal{N}(\Psi_t), \Psi'_i \in \mathcal{N}(\Psi_i)} d(\Psi'_t, \Psi'_i)$. If more than one pair of displayed networks have the smallest distance, a pair with the largest number of reticulation nodes is selected. We now define:

- True positives: $TP(\Psi_t, \Psi_i) = 2 \cdot r(\Psi'_t) + t - d(\Psi'_t, \Psi'_i)$.
- True positives rate: $TPR(\Psi_t, \Psi_i) = TP(\Psi_t, \Psi_i) / (t + 2 \cdot r(\Psi_t))$.
- False positives: $FP(\Psi_t, \Psi_i) = 2(r(\Psi_i) - r(\Psi'_i)) + d(\Psi'_t, \Psi'_i)$.

- False positives rate: $FPR(\Psi_t, \Psi_i) = FP(\Psi_t, \Psi_i)/(t + 2 \cdot r(\Psi_i))$.
- False negatives rate: $FNR(\Psi_t, \Psi_i) = 1 - TPR(\Psi_t, \Psi_i)$.

For example, if the networks of Figure 1(a) and Figure 1(c) are the true network Ψ_t and inferred network Ψ_i , respectively, then Ψ'_t and Ψ'_i are the networks of Figure 1(b) and Figure 1(c), respectively. In this case, we have $d(\Psi'_t, \Psi'_i) = 3$, $r(\Psi'_t) = 0$, $TP = 4$, $FP = 3$, $TPR = 4/9$, and $FPR = 3/7$.

4 Results and Discussion

To assess the performance of tree-based inference of phylogenetic networks, we tested both methods described above on simulated data. Furthermore, we analyzed a biological data set and contrasted the results to those obtained by other methods.

For the simulated data, we used the same 24 networks used in [26], as those were generated under a birth-hybridization model and varied in their complexity. For each integer in $[0, 5]$, four networks in the data set have that number of reticulation nodes. Two of the 24 networks are not tree-based [25].

As in [26], these networks were divided into 3 groups of hardness (in terms of inference): 8 “easy” (E), 8 “medium” (M), and 8 “hard” (H). Each network has 16 taxa and 1 outgroup. For each network, we generated 100 gene trees with two individuals per species using the program `ms` [7], and then generated sequences of length 1000 using `Seq-gen` [10] under the GTR model. We set the population mutation rate at 0.02, base frequencies of A, C, G and T at $[0.2112, 0.2888, 0.2896, 0.2104]$, respectively, and the transition probabilities at $[0.2173, 0.9798, 0.2575, 0.1038, 1, 0.2070]$. We then ran the aforementioned methods on the data. In terms of the complexity of the data sets, the number of distinct gene trees (out of 100) in the 24 data sets varied between 69 and 100. That is, the rate of ILS is quite high.

4.1 Accuracy of the start tree

4.1.1 Performance of the concatenation method

By concatenating the sequence data of all loci, we obtained for each network 34 sequences of length 100000 each (since we have two individuals per species). We then inferred a tree on the concatenated sequences of each of the 24 data sets using `IQ-TREE` [9]. As there are 34 taxa in the resulting trees but there are only 17 species, we first identified the data sets where the two individuals from each of the 17 species form a monophyletic group. We found that for 21 data sets, the resulting tree group the two individuals from each species monophyletically, whereas in the three remaining data sets, the individuals of some species were not monophyletic. For the 21 data sets, we “collapsed” the two individuals in the inferred tree into a single leaf with the respective taxon name so as to compare the accuracy of the inferred “species tree” to the true phylogenetic network.

Of the 21 data sets, we found that 12 of them resulted in species trees that are displayed by, or are backbone trees of, their corresponding true network. The remaining nine trees had an average distance, based on Eq. (1), of 3.33.

These results illustrate that in the presence of ILS and hybridization, concatenation has a poor performance; only in half of the 24 data sets did the method infer a tree that could be augmented into the true network.

4.1.2 Performance of ASTRAL

We then turned to assessing the accuracy of species trees inferred by ASTRAL-III [23]. We inferred a gene tree on the sequence alignment of each locus using IQ-TREE [9] and used these inferred gene trees as the input set \mathcal{G} of gene trees to ASTRAL-III.

ASTRAL-III had a much better performance than species tree inference on the concatenated sequences. In fact, ASTRAL-III inferred a backbone tree that could be augmented into the respective true network in 87.5% of the data sets. More specifically, 21 out of 24 species trees inferred by ASTRAL-III are backbone trees of the true networks. For the 3 data sets where the inferred tree could not serve as a backbone of the true network, the average distance between the trees and the true networks, as computed by Eq. (1), was 2.67.

This shows that the start tree built from inferred gene trees using ASTRAL-III is much better than concatenation using IQ-TREE. This is because the rate of ILS is high, and ASTRAL-III considers the gene tree topology conflicts.

To explore whether the performance of ASTRAL-III would improve with more loci, we increased the number of gene trees simulated on each network to 1,000 and used the true gene trees as input to ASTRAL-III. Now, ASTRAL-III inferred a tree that is the backbone of its respective true network for 22 of the 24 data sets.

4.2 Accuracy of the inferred networks

Given the accuracy of the species trees produced by ASTRAL-III, we used it as the start tree for network inference. Since the input gene trees and output species trees of ASTRAL-III are all unrooted, we rerooted all the trees at the designated outgroup and deleted it. Then we used the rooted gene trees and rooted species trees as input to both network inference procedures. We evaluated the performance of the network inference on NOTS (Night Owls Time-Sharing Service), which is a batch scheduled High-Throughput Computing (HTC) cluster.

4.2.1 Performance of StepwiseAugment

We ran **StepwiseAugment** on the start trees obtained by ASTRAL-III on all 24 data sets, under the MDC criterion, while specifying to the inference method the true number of reticulations. While knowing the true number of reticulations is not doable in practice, we made this decision for two reasons. First, we wanted to test how the approach works under the ideal situation (of knowing the true number of reticulations). Second, determining the number of reticulations is not doable in a systematic way with the MDC criterion. The correctness achieved by **StepwiseAugment** in this case is 41.7%, reflecting that in only 10 out of the 24 cases did the method infer the correct networks. For the 3 data sets where the start trees are not backbone trees of the networks, obviously, the method could not infer the correct network. For the 17 data sets with at least one reticulation and correct start trees, the method was able to find the correct reticulation in only 8 of them. These results show that **StepwiseAugment** coupled with the MDC criterion is not a viable approach for inferring phylogenetic networks.

4.2.2 Performance of LocalSearchAugment

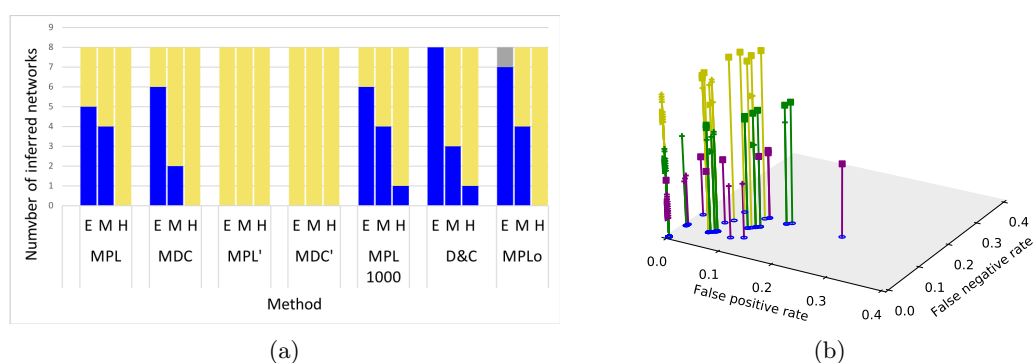
We then set out to study the performance of **LocalSearchAugment** when using the ASTRAL-III species tree as the start tree and under both the pseudo-likelihood and MDC criteria. One might ask: If the “quasi brute-force” procedure **StepwiseAugment** did not perform well, could a local search heuristic perform better? The answer in our case is positive.

The reason for this is because when searching for an optimal network with $k + 1$ reticulations, **StepwiseAugment** is already “stuck” with the k reticulations it had identified already, and this might not necessarily result in an optimal solution. A truly brute-force approach for considering all networks with MAX reticulations on a start tree T would first identify the set of all possible reticulations that could be added to T (including “dependent” reticulations, where a reticulation edge is added between two reticulation edges or between one tree and one reticulation edge), and then consider every subset of them. Such an approach is not only computationally infeasible for the size of the data sets we consider, but is also very hard to implement because of the dependent reticulations.

Since inference under neither MDC nor pseudo-likelihood is equipped with a systematic way to determine the number of reticulations (i.e., a stopping rule), we ran **LocalSearchAugment** under both criteria in two different ways. In one way, we set the number of reticulations for each data set to be the true number for that data set. In the figures below, we label results based on this setting as MPL (for maximum pseudo-likelihood) and MDC (for minimizing deep coalescences). In the other way, we set the maximum number of reticulations at 5, which is the largest number of reticulations in all 24 data sets. In the figures below, we label results based on this setting as MPL' (for maximum pseudo-likelihood) and MDC' (for minimizing deep coalescences). The local heuristic search on each data set was repeated 20 times with random restarts to obtain the network estimates.

We also ran MPL on 1,000 gene trees to explore its convergence when the amount of data is larger. In the figures below, results of this run are labeled MPL1000. Finally, we compared these methods to two methods that can scale to larger data sets: The maximum pseudo-likelihood method of [21], where the network space is searched directly without a fixed start tree (the maximum number of reticulations is set at the true value), and the newly developed divide-and-conquer method of [26] when the full set of trinets is considered in the divide step (this method does not require a pre-set number of reticulations). The results of these runs are labeled MPLo and *D&C*, respectively, in the figures below. For these two methods, the input gene trees is the set of gene trees inferred by IQ-TREE.

The results of all these runs are summarized in Figure 2.



■ **Figure 2 Accuracy of inferred networks.** (a) Blue corresponds to the number of inferred networks that are topologically identical to the true networks. Yellow corresponds to the number of inferred networks that share backbone networks of true networks. Gray corresponds to all other cases. (b) Yellow corresponds to MPL. Green corresponds to MDC. Purple corresponds to *D&C*. Triangles correspond to easy (E), crosses correspond to medium (M) and squares correspond to Hard (H). The z-axis has no meaning but is used for ease of visualization.

As Figure 2(a) shows, when the true number of reticulations is assumed, both MPL and MDC perform relatively similarly, with MDC outperforming MPL by one of the easy data sets and MPL outperforming MDC by two of the medium data sets. Neither of the two methods obtains the correct network on any of the 8 hard data sets. When the true number of reticulations is not assumed, MPL' and MDC's do not infer the correct network on any of the 24 data sets. For all these four methods, when the correct network is not inferred, the distance between the inferred network and true network is $D(\Psi_t, \Psi_i) = 0$. That is, while the methods do not infer the correct network, the inferred networks share an underlying structure with the true ones.

Increasing the number of gene trees in the input to 1,000 results in a slight improvement to MPL, where the true network is inferred on two more data sets. Furthermore, searching the network space directly under the MPL criterion performs almost similarly to a tree-based approach, with the only difference being that the method (MPLo) now infers two more networks correctly and infers a wrong network in the case of the 8 easy networks. It is important, though, to emphasize again that for MPL, MDC, MPL1000, and MPLo, the true number of reticulations is assumed. As the results show, *D&C* has the best accuracy where not only does it always either infer the correct network or a network that shares an underlying structure with the true one, but it also does so without *a priori* knowledge of the true or maximum number of reticulations.

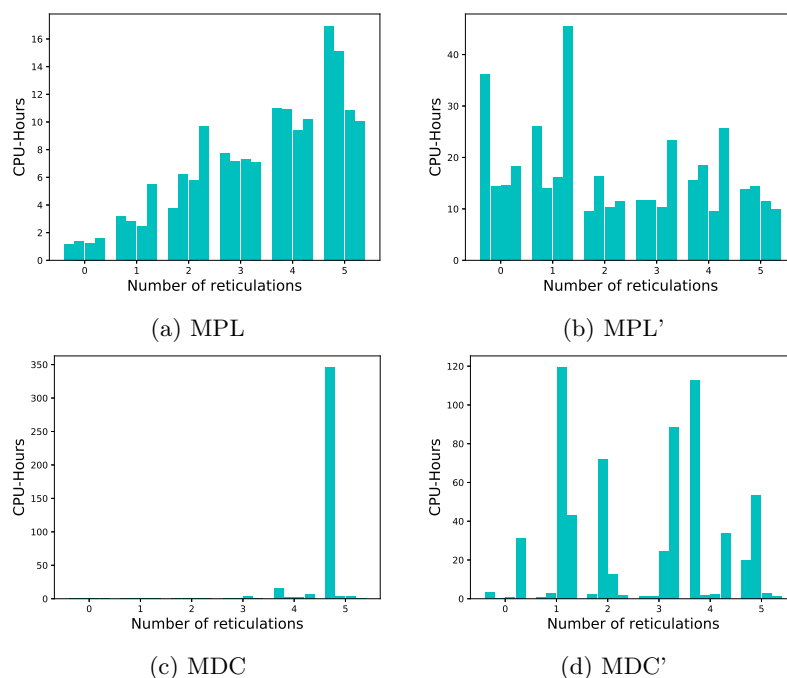
To better understand the behavior of these methods, we inspected the FPR and FNR of the methods. As Figure 2(b) shows, on average, the FPR and FNR of MPL are 15.2% and 11.1% higher than those of *D&C*, respectively, and those of MDC are 52.1% and 28.2% higher than those of *D&C* respectively.

We show the running times of tree-based network inference under both MPL and MDC in Figure 3. The figure shows that almost all data sets are analyzed within 16 hours by MPL when the true number of reticulations is assumed, but that time increases to about 40 hours for some data sets when the search is allowed to explore networks up to 5 reticulations. Inference based on the MDC criterion, on the other hand, takes much longer in some cases, and as the figure shows, the number of reticulations itself is not the only determinant of the running time. The placement of the reticulations in the network is a major factor of the complexity, a result similar to that shown in the case of computing the likelihood of networks [29, 5].

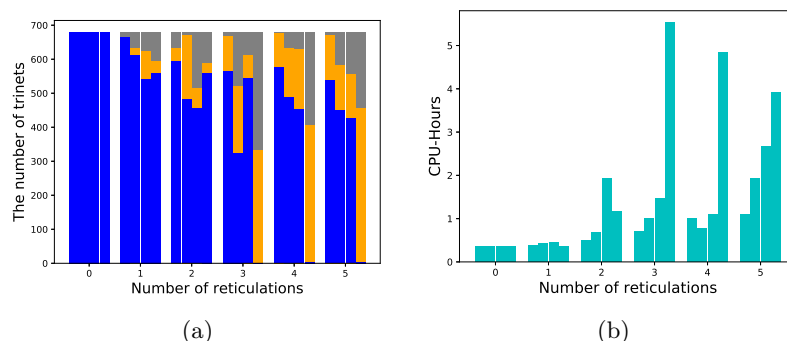
On average, when compared with 1636.8 CPU hours spent by *D&C*, the tree-augmentation methods only spend 7.0, 16.4, 17.0, 26.4 and 3.8 hours for MPL, MDC, MPL', MDC' and MPL1000, respectively. The computational bottleneck of *D&C* comes from the substantial time that it takes MCMC-SEQ [16] to infer each of the 680 3-taxon subnetworks (though this can be easily parallelizable, as the inferences of subnetworks are done independently).

4.3 Towards combining the strengths of tree-based and *D&C* inference

As shown above, tree-based inference is much faster than inference by *D&C*, whereas the latter produces more accurate results. As we mentioned, the majority of the running time of *D&C* comes from the costly step of inferring the 3-taxon subnetworks using the expensive Bayesian approach of [16]. The question that we set out to explore here is: Can the running time of *D&C* be improved by utilizing tree-based inference of the trinetts? We limit our attention in this study to one part of this question, namely, how does tree-based inference perform in terms of inferring the 3-taxon subnetwork topologies (when using *a priori* knowledge of the true number of reticulation)? To explore this question, we considered



■ **Figure 3** Running times of tree-based network inference based on the MPL and MDC criteria. Each subfigure shows the running times on 24 networks for the respective method. Each bar represents the CPU hours taken to infer this network. The x-axis is the number of reticulations, where each number of reticulations has 4 networks with that number.



■ **Figure 4** The accuracy and running time of tree-based inference of 3-taxon subnetworks. (a) Accuracy of the inferred subnetworks. Each bar represents the number of subnetworks in each network that are identical (blue), inside (orange), others (grey) to the true subnetworks. (b) Running time for inferring subnetworks. Each bar represents the CPU hours spent to infer all subnetworks for each network. The bars are arranged in a way that they correspond in a 1-1 manner to Figure 3 in [26].

each 3-taxon subset of the 17 taxa in each data set and inferred a network on it (with the true number of reticulation) using tree-based inference (minimizing deep coalescence) starting from all three possible topologies. Figure 4 shows the accuracy and running times of this approach. As the figure shows, on average, the running time of 3-taxon subnetwork inference is reduced from 1636.8 to 1.4 CPU hours with a loss of only about 7% in accuracy when considering identical subnetworks and about 1% when considering backbone networks. This

is a massive improvement in the running times with hardly any sacrifice in the topological accuracy. However, there are two caveats here. First, the true number of reticulations is assumed in this case (whereas that number is not assumed in [26]). Second, the merger step of the *D&C* method of [26] assumes knowledge of the divergence times of the nodes in the 3-taxon subnetworks. A promising direction that emerges from these results is that combining tree-based inference with the Bayesian method of [16] could potentially provide an accurate and fast approach to inferring the subnetworks and, consequently, improving the running of *D&C* without sacrificing its accuracy.

4.4 Analysis of empirical data set

We reanalyzed an empirical data set of rainbow skinks [1]. Selecting 11 taxa and 22 individuals, we inferred 100 gene trees from aligned sequences of 100 loci using IQ-TREE. We inferred a start species tree from gene trees using ASTRAL-III, which is shown in Figure 5(a). We then

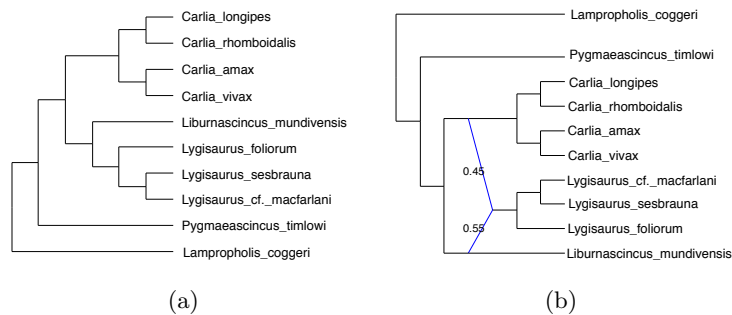


Figure 5 The inferred species tree and network of the empirical data set. (a) The ASTRAL species tree from IQTREE gene trees. (b) The inferred network with 1 reticulation using maximum pseudo-likelihood relying on the backbone tree.

rooted the gene trees and start species trees at the *Lampropholis guichenoti* as an outgroup, and deleted it. We then ran **LocalSearchAugment** under the pseudo-likelihood criterion on the data and set the maximum number of reticulations at 1, in order to compare with the results reported in [26]. The method took 6 minutes to obtain the network shown in Figure 5(b), while *D&C* took 3670 CPU-hours. While the start species tree agrees with the inferred species tree reported in Figure 2 in [1], the tree-based network is different from the network inferred by *D&C* and reported in Figure 5 in [26], once again, demonstrating the efficiency of tree-based inference, but its limitations in terms of accuracy when run on large data sets.

5 Conclusions and future work

In this paper, we set out to study the performance of tree-based inference of phylogenetic networks, as this approach would be promising for large-scale phylogenetic network inference provided it has good accuracy. While we find the method to be much faster than a recently introduced divide-and-conquer approach, its accuracy is inferior to the latter. However, the approach is accurate for inference of small-scale networks, which could prove to be valuable for speeding up the divide-and-conquer approach while maintaining its accuracy. For example, the topologies of the subnetworks could be inferred using tree-based inference, and then the Bayesian method of [16] is run to only estimate the divergence times, rather than estimating the topologies as well. We identify this as an important direction for future research.

Another important open question whose answer would have practical implications on searching the network space: is an optimal phylogenetic network with $k + 1$ reticulations (under some optimality criterion) obtainable by adding a reticulation event to an optimal network with k reticulations? While the answer to this question could be negative for all optimality criteria (likelihood, pseudo-likelihood, MDC, etc.), the answer could be positive for certain classes of phylogenetic networks.

References


- 1 Jason G Bragg, Sally Potter, Ana C Afonso Silva, Conrad J Hoskin, Benjamin YH Bai, and Craig Moritz. Phylogenomics of a rapid radiation: the Australian rainbow skinks. *BMC Evolutionary Biology*, 18(1):15, 2018.
- 2 Gabriel Cardona, Merce Llabrés, and Francesc Rosselló. Two results on distances for phylogenetic networks. In *Advances in Bioinformatics*, pages 93–100. Springer, 2010.
- 3 Gabriel Cardona, Mercè Llabrés, Francesc Rosselló, and Gabriel Valiente. On Nakhleh’s metric for reduced phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(4):629–638, 2009.
- 4 Ruth Davidson, Pranjali Vachaspati, Siavash Mirarab, and Tandy Warnow. Phylogenomic species tree estimation in the presence of incomplete lineage sorting and horizontal gene transfer. *BMC Genomics*, 16(10):S1, 2015.
- 5 RA Leo Elworth, Huw A Ogilvie, Jiafan Zhu, and Luay Nakhleh. Advances in computational methods for phylogenetic networks in the presence of hybridization. In *Bioinformatics and Phylogenetics*, pages 317–360. Springer, 2019.
- 6 Andrew R Francis and Mike Steel. Which phylogenetic networks are merely trees with additional arcs? *Systematic Biology*, 64(5):768–777, 2015.
- 7 R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18:337–338, 2002.
- 8 Luay Nakhleh. A metric on the space of reduced phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 7(2):218–222, 2010.
- 9 Lam-Tung Nguyen, Heiko A. Schmidt, Arndt von Haeseler, and Bui Quang Minh. IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular Biology and Evolution*, 32(1):268–274, November 2014.
- 10 A. Rambaut and N. C. Grassly. Seq-gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comp. Appl. Biosci.*, 13:235–238, 1997.
- 11 David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.
- 12 Sebastien Roch and Sagi Snir. Recovering the treelike trend of evolution despite extensive lateral genetic transfer: a probabilistic analysis. *Journal of Computational Biology*, 20(2):93–112, 2013.
- 13 Claudia Solís-Lemus, Mengyao Yang, and Cécile Ané. Inconsistency of species-tree methods under gene flow. *Systematic Biology*, 2016.
- 14 C. Than, D. Ruths, and L. Nakhleh. PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC Bioinformatics*, 9(1):322, 2008.
- 15 D. Wen, Y. Yu, and L. Nakhleh. Bayesian Inference of Reticulate Phylogenies under the Multispecies Network Coalescent. *PLoS Genetics*, 12(5):e1006006, 2016.
- 16 Dingqiao Wen and Luay Nakhleh. Co-estimating Reticulate Phylogenies and Gene Trees from Multi-locus Sequence Data. *Systematic Biology*, 67(3):439–457, 2018.
- 17 Dingqiao Wen, Yun Yu, Jiafan Zhu, and Luay Nakhleh. Inferring Phylogenetic Networks Using PhyloNet. *Systematic Biology*, 67(4):735–740, 2018.
- 18 Y. Yu, R.M. Barnett, and L. Nakhleh. Parsimonious inference of hybridization in the presence of incomplete lineage sorting. *Systematic Biology*, 62(5):738–751, 2013.

- 19 Y. Yu, J.H. Degnan, and L. Nakhleh. The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. *PLoS Genetics*, 8:e1002660, 2012.
- 20 Y. Yu, J. Dong, K. Liu, and L. Nakhleh. Maximum likelihood inference of reticulate evolutionary histories. *Proceedings of the National Academy of Sciences*, 111(46):16448–6453, 2014.
- 21 Y. Yu and L. Nakhleh. A Maximum Pseudo-likelihood Approach for Phylogenetic Networks. *BMC Genomics*, 16:S10, 2015.
- 22 Y. Yu, N. Ristic, and L. Nakhleh. Fast Algorithms and Heuristics for Phylogenomics under ILS and Hybridization. *BMC Bioinformatics*, 14(Suppl 15):S6, 2013.
- 23 Chao Zhang, Maryam Rabiee, Erfan Sayyari, and Siavash Mirarab. ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics*, 19(6):153, 2018.
- 24 Chi Zhang, Huw A Ogilvie, Alexei J Drummond, and Tanja Stadler. Bayesian Inference of Species Networks from Multilocus Sequence Data. *Molecular Biology and Evolution*, 35(2):504–517, 2018.
- 25 Louxin Zhang. On tree-based phylogenetic networks. *Journal of Computational Biology*, 23(7):553–565, 2016.
- 26 Jiafan Zhu, Xinhao Liu, Huw A Ogilvie, and Luay K Nakhleh. A Divide-and-Conquer Method for Scalable Phylogenetic Network Inference from Multi-locus Data. *Bioinformatics*, 2019. To appear.
- 27 Jiafan Zhu and Luay Nakhleh. Inference of Species Phylogenies from Bi-allelic Markers Using Pseudo-likelihood. *Bioinformatics*, 34:i376–i385, 2018.
- 28 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):1–32, January 2018.
- 29 Jiafan Zhu, Yun Yu, and Luay Nakhleh. In the light of deep coalescence: revisiting trees within networks. *BMC Bioinformatics*, 17(14):415, 2016.

A Combinatorial Approach for Single-cell Variant Detection via Phylogenetic Inference

Mohammadamin Edrisi

Department of Computer Science, Rice University, Houston, TX, USA
edrisi@rice.edu

Hamim Zafar 

Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
hzafar@andrew.cmu.edu

Luay Nakhleh¹ 

Department of Computer Science, Rice University, Houston, TX, USA
nakhleh@rice.edu

Abstract

Single-cell sequencing provides a powerful approach for elucidating intratumor heterogeneity by resolving cell-to-cell variability. However, it also poses additional challenges including elevated error rates, allelic dropout and non-uniform coverage. A recently introduced single-cell-specific mutation detection algorithm leverages the evolutionary relationship between cells for denoising the data. However, due to its probabilistic nature, this method does not scale well with the number of cells. Here, we develop a novel combinatorial approach for utilizing the genealogical relationship of cells in detecting mutations from noisy single-cell sequencing data. Our method, called scVILP, jointly detects mutations in individual cells and reconstructs a perfect phylogeny among these cells. We employ a novel Integer Linear Program algorithm for deterministically and efficiently solving the joint inference problem. We show that scVILP achieves similar or better accuracy but significantly better runtime over existing methods on simulated data. We also applied scVILP to an empirical human cancer dataset from a high grade serous ovarian cancer patient.

2012 ACM Subject Classification Applied computing → Computational genomics

Keywords and phrases Mutation calling, Single-cell sequencing, Integer linear programming, Perfect phylogeny

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.22

Funding This work was supported by the National Science Foundation Grant IIS-1812822 to L.N.

1 Introduction

Recently developed single-cell sequencing technologies provide unprecedented resolution in studying genetic variability within a complex tissue [27, 19]. By enabling the sequencing of the genomes of individual cells, these approaches help in delineating the genomic landscape of rare cell populations and provide insights into the somatic evolutionary process. While having single-cell resolution can have major impact on our understanding in neurobiology, immunobiology and other domains [27], it is particularly useful for investigating intratumor heterogeneity and clonal evolution in cancer [20, 29]. Interplay of mutation and selection gives rise to intratumor heterogeneity [21, 18], which contributes to tumor progression and metastasis [25] and can cause tumor relapse via drug resistance [10, 2]. To-date, tumors were mostly analyzed by sequencing bulk samples that consist of admixture of DNA from

¹ Corresponding Author



thousands to millions of cells [9, 22]. Such datasets provide aggregate variant allele frequency (VAF) profiles of somatic mutations, which are subjected to further deconvolution into tumor subpopulations [6]. However, VAFs are noisy, provide limited resolution in identifying rare subpopulations [19] and their deconvolution into clones depends on strong assumptions and therefore may not hold in practice [1].

By enabling the direct measurement of cellular genotypes, single-cell sequencing (SCS) data resolves intratumor heterogeneity to a single-cell level [8, 14]. However, the preparation of SCS data calls for whole genome amplification (WGA) process that can amplify the limited DNA material of a single cell to meet the amount of DNA needed for sequencing [29]. This extensive genome amplification elevates the noise level in SCS data and creates a unique error profile different from bulk sequencing [29]. Technical errors in SCS data include nonuniform coverage, allelic dropout (ADO), false-positive (FP) and false-negative (FN) errors [19, 29]. ADO, i.e., preferential nonamplification of one allele in a heterozygous mutation can result in the loss of the mutated allele removing all evidences of the mutation. On the other hand, random errors introduced during the early stages of amplification can appear as false positive artifacts due to uneven allelic amplifications. The nonuniform coverage in SCS further results in missing data at the sites with insufficient coverage.

Since traditional next-generation sequencing (NGS) variant callers such as GATK [5] do not account for SCS specific errors, variant callers such as Monovar [31] and SCcaller [7] have been specifically designed to deal with the technical errors in SCS. Monovar pools sequencing data across cells to address the problem of low coverage and probabilistically accounts for ADO and FP errors. SCcaller independently detects variants in each cell and accounts for local allelic amplification biases. In another direction, tumor phylogeny inference methods [12, 30, 28] that account for the errors have been developed for studying tumor heterogeneity. A new variant caller SCI Φ [23] combines single-cell variant calling with the reconstruction of the tumor phylogeny. It leverages the fact that somatic cells are related by a phylogeny and employs a Markov Chain Monte Carlo algorithm for calling mutations in each single cell. However, being a probabilistic method, SCI Φ does not scale well with the size of the data and takes a long time to converge on a solution. At the same time, depending on the starting point, it might converge to different solutions.

Here, we present a novel combinatorial formulation for jointly identifying mutations in single cell data by employing the underlying phylogeny. Our approach, scVILP (**s**ingle-**c**ell **V**ariant calling via **I**nteger **L**inear **P**rogram) assumes that the somatic cells evolve along a phylogenetic tree and mutations are acquired along the branches following the infinite sites model as have been used in previous bulk and single-cell studies [6, 12, 23]. We aim to identify the set of single-nucleotide variants in the single cells and genotype them in such a way so that it maximizes the probability of the observed read counts and also the cells are placed at the leaves of a perfect phylogeny that satisfies the infinite sites assumption (ISA). Our solution is deterministic and we solve this problem using a novel Integer Linear Program (ILP) that achieves similar accuracy as SCI Φ but performs significantly better than SCI Φ in terms of runtime. For whole-exome sequencing (WES) data consisting of large numbers of somatic mutation loci, running ILP-solvers can result in high memory consumption. To address this, we introduce a divide-and-conquer version of scVILP where the data matrix is partitioned by columns, scVILP is run on the subsets, and then the results are merged via another ILP formulation to resolve any violations of the ISA assumption in the full dataset. The supertree-based approach achieves similar accuracy as SCI Φ but is much faster.

2 Method

In this section, we formulate the joint inference of mutations in single cells and the cellular phylogeny as a combinatorial optimization problem. Following [23], we start by first identifying potential mutation loci and then solve the joint inference problem using a novel Integer Linear Program (ILP).

2.1 Input Data

The input data for scVILP corresponds to a read count matrix for n cells in the dataset. Let us assume that we identify m loci as putative mutated loci. For the joint inference, our input is an $n \times m$ matrix, $X_{n \times m} = (X_{ij})$, where $X_{ij} = (r_{ij}, v_{ij})$ denotes the reference and variant read counts for cell i and locus j . $r_{ij} + v_{ij}$ is the coverage at this locus.

2.2 Model for Amplification Error

WGA process introduces false positive and false negative errors in single-cell data. We assume that α and β respectively denote false positive and false negative error rates of single-cell data. We introduce a binary variable Y_{ij} that denotes the true genotype of mutation j in cell i . $Y_{ij} = 1$ and $Y_{ij} = 0$ respectively denote the presence and absence of the mutation. We introduce another variable D_{ij} that denotes the amplified genotype (after introduction of WGA errors) of mutation j in cell i . Following previous WGA error models [12, 30], we introduce the following likelihood scheme:

$$P(D_{ij}|Y_{ij}) = \begin{cases} \alpha, & \text{for } D_{ij} = 1, Y_{ij} = 0 \\ 1 - \alpha, & \text{for } D_{ij} = 0, Y_{ij} = 0 \\ \beta, & \text{for } D_{ij} = 0, Y_{ij} = 1 \\ 1 - \beta, & \text{for } D_{ij} = 1, Y_{ij} = 1 \end{cases} \quad (1)$$

The above likelihood scheme (Eq. (1)) can also be rewritten as:

$$\begin{aligned} P(D_{ij} = 0|Y_{ij}) &= (1 - \alpha)^{(1-Y_{ij})} \beta^{Y_{ij}} \\ P(D_{ij} = 1|Y_{ij}) &= \alpha^{(1-Y_{ij})} (1 - \beta)^{Y_{ij}} \end{aligned} \quad (2)$$

2.3 Read Count Model

The read counts (r_{ij}, v_{ij}) at locus j of cell i is modeled using a Beta-Binomial distribution as commonly used for bulk sequencing data [9] given by:

$$P(r_{ij}, v_{ij}|D_{ij}) = \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_{D_{ij}})^{v_{ij}} (1 - \mu_{D_{ij}})^{r_{ij}} \quad (3)$$

where, $\mu_{D_{ij}}$ denotes the parameter of the Binomial distribution and it represents the probability of drawing a variant read. This parameter depends on the amplified genotype and is a Beta distributed variable. When $D_{ij} = 0$, $\mu_{D_{ij}} = \mu_0$ denotes the probability of observing a variant read due to sequencing error, for $D_{ij} = 1$, $\mu_{D_{ij}} = \mu_1$ denotes the probability of observing a variant read. Given the amplified genotype D_{ij} , the probability of observing read count (r_{ij}, v_{ij}) is given by

$$\begin{aligned} P(r_{ij}, v_{ij}|D_{ij} = 0) &= \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_0)^{v_{ij}} (1 - \mu_0)^{r_{ij}} \\ P(r_{ij}, v_{ij}|D_{ij} = 1) &= \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_1)^{v_{ij}} (1 - \mu_1)^{r_{ij}} \end{aligned} \quad (4)$$

The joint probability of read counts and amplified genotype given the true genotype can be computed by multiplying the probabilities in Eq. (2) and Eq. (4) as given by

$$\begin{aligned} P(r_{ij}, v_{ij}, D_{ij} = 0 | Y_{ij}) &= \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_0)^{v_{ij}} (1 - \mu_0)^{r_{ij}} \times \alpha^{(1-Y_{ij})} (1 - \beta)^{Y_{ij}} \\ P(r_{ij}, v_{ij}, D_{ij} = 1 | Y_{ij}) &= \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_1)^{v_{ij}} (1 - \mu_1)^{r_{ij}} \times (1 - \alpha)^{(1-Y_{ij})} \beta^{Y_{ij}} \end{aligned} \quad (5)$$

2.4 Likelihood Function

The likelihood function of the true genotypes for the cells for an $n \times m$ read count matrix is given by

$$\mathcal{L}(Y) = \prod_{i=1}^n \prod_{j=1}^m P(r_{ij}, v_{ij} | Y_{ij}) \quad (6)$$

For computing $P(r_{ij}, v_{ij} | Y_{ij})$, we treat the amplified genotype D_{ij} as a nuisance parameter and marginalize it out as given below:

$$\begin{aligned} P(r_{ij}, v_{ij} | Y_{ij}) &= \sum_{D_{ij} \in \{0,1\}} P(r_{ij}, v_{ij}, D_{ij} | Y_{ij}) \\ &= \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_0)^{v_{ij}} (1 - \mu_0)^{r_{ij}} \times \alpha^{(1-Y_{ij})} (1 - \beta)^{Y_{ij}} \\ &\quad + \binom{r_{ij} + v_{ij}}{v_{ij}} (\mu_1)^{v_{ij}} (1 - \mu_1)^{r_{ij}} \times (1 - \alpha)^{(1-Y_{ij})} \beta^{Y_{ij}} \end{aligned} \quad (7)$$

If we denote the Binomial distributions as below:

$$\begin{aligned} \text{Bin}_{ij}(\mu_0) &= \binom{r_{ij} + v_{ij}}{v_{ij}} \mu_0^{v_{ij}} (1 - \mu_0)^{r_{ij}} \\ \text{Bin}_{ij}(\mu_1) &= \binom{r_{ij} + v_{ij}}{v_{ij}} \mu_1^{v_{ij}} (1 - \mu_1)^{r_{ij}} \end{aligned} \quad (8)$$

then $P(r_{ij}, v_{ij} | Y_{ij})$ can be rewritten as:

$$P(r_{ij}, v_{ij} | Y_{ij}) = \left[\text{Bin}_{ij}(\mu_0)\alpha + \text{Bin}_{ij}(\mu_1)(1 - \alpha) \right]^{(1-Y_{ij})} \left[\text{Bin}_{ij}(\mu_0)(1 - \beta) + \text{Bin}_{ij}(\mu_1)\beta \right]^{Y_{ij}} \quad (9)$$

Here our goal is to find a matrix Y such that the likelihood function defined in Eq. (6) is maximized. Taking the logarithm of the likelihood function in Eq. (6) and negating it, we obtain the negative log-likelihood function:

$$\begin{aligned} \mathcal{NLL} &= - \sum_{i=1}^n \sum_{j=1}^m \left[(1 - Y_{ij}) \log \left[\text{Bin}_{ij}(\mu_0)\alpha + \text{Bin}_{ij}(\mu_1)(1 - \alpha) \right] \right. \\ &\quad \left. + Y_{ij} \log \left[\text{Bin}_{ij}(\mu_0)(1 - \beta) + \text{Bin}_{ij}(\mu_1)\beta \right] \right] \end{aligned} \quad (10)$$

Since this function is linear with respect to the variables Y_{ij} , we can solve this by providing it as the objective function to an ILP solver.

2.5 ILP Constraints Pertaining to Perfect Phylogeny

We represent the tumor phylogeny as a perfect phylogeny (PP), leaves of which represent the sampled n cells. A perfect phylogeny satisfies the “infinite sites assumption” (ISA), which states that each genomic locus in the dataset mutates at most once during the evolutionary history [15]. We aim to find a genotype matrix Y that provides a perfect phylogeny. In order for a binary matrix to be a PP, the three-gametes rule has to hold, i.e., for any given pair of columns (mutations) there must be no three rows (cells) with configuration (1, 0), (0, 1) and (1, 1). 0 and 1 represent the ancestral (unmutated) and mutated state respectively.

In order to enforce that matrix Y satisfies the perfect phylogeny criterion, we follow the ILP formulation introduced by Gusfield et al. [11]. which introduces variables $B(p, q, a, b) \in \{0, 1\}$ for each pair of mutation loci p and q and for each $(a, b) \in \{(0, 1), (1, 0), (1, 1)\}$. For each cell i , the following constraints are introduced on the variables $B(p, q, a, b)$, Y_{ip} , and Y_{iq} , $i \in \{1, \dots, n\}$ and $p, q \in \{1, \dots, m\}$:

$$B(p, q, 0, 1) \geq Y_{iq} - Y_{ip} \quad (11)$$

$$B(p, q, 1, 0) \geq Y_{ip} - Y_{iq} \quad (12)$$

$$B(p, q, 1, 1) \geq Y_{ip} + Y_{iq} - 1 \quad (13)$$

To enforce a perfect phylogeny, the following constraint must hold for all pairs of p and q :

$$2 \geq B(p, q, 1, 0) + B(p, q, 0, 1) + B(p, q, 1, 1) \quad (14)$$

To maximize the likelihood, we need to minimize the negative log-likelihood function defined in Eq. (10), subject to the constraints in Eqns. (11), (12), (13), and (14).

Our ILP formulation can be routinely solved through commercial tools such as CPLEX or Gurobi. In our analysis, we have used Gurobi that concurrently runs multiple deterministic solvers (e.g., dual simplex, primal simplex, etc.) on multiple threads and returns the one that finishes first. As a result, for the same input data and parameter values, Gurobi returns a deterministic solution. It is important to highlight, though, that there could be multiple optimal solutions.

2.6 Handling Missing Data

For the loci for which the read count values are missing, we do not consider the related Y_{ij} variables in the objective function. The values of Y_{ij} for such missing entries are imputed according to the other values and subject to the Perfect Phylogeny constraints.

2.7 Perfect-Phylogeny Supertree

Running ILP-solvers is expensive in terms of memory consumption, specially when the number of mutation loci is high. To overcome this limitation, we use a divide-and-conquer approach consisting of the following steps:

1. Partition the columns of $X_{n \times m}$, into k subsets of mutation loci, $\{C_1, \dots, C_k\}$, where each C_j is a set of columns in $X_{n \times m}$. We define the submatrices $\{X^{(1)}, \dots, X^{(k)}\}$ as follows:

$$X^{(i)} = X_{[1, \dots, n; c_1^{(i)}, \dots, c_s^{(i)}]} \quad c_j^{(i)} \in C_i \quad \text{and} \quad s = |C_i| \quad (15)$$

2. For each submatrix, $X^{(i)}$, optimize the objective function in Eq. 10 (subject to the constraints in Eqns 11, 12, 13, and 14) independently. This step results in genotyped submatrices, $\{G^{(1)}, \dots, G^{(k)}\}$.

- Concatenate the genotyped submatrices into $G_{n \times m}$ as follows:

$$G_{n \times m} = (G^{(1)} \mid G^{(2)} \mid \dots \mid G^{(k)})$$

- If $G_{n \times m}$ does not admit a Perfect Phylogeny, find the minimum number of changes to apply on the entries of $G_{n \times m}$ such that the modified matrix satisfies the Perfect Phylogeny model. We name this final step as PERFECT PHYLOGENY CORRECTION PROBLEM (PPC), which is described in the following section.

2.8 The Perfect Phylogeny Correction Problem

We define the PERFECT PHYLOGENY CORRECTION PROBLEM as inferring matrix \mathcal{G} from matrix G , which does not satisfy perfect phylogeny such that \mathcal{G} admits a perfect phylogeny. PPC consists of two steps:

- Find the minimum number of characters (mutation loci) which must be removed in order to make the matrix, G admit a perfect phylogeny. This problem is known as CHARACTER-REMOVAL PROBLEM (CRP) [11].
- Among the entries of the selected loci in the previous step, minimize the number of changes to apply such that final matrix \mathcal{G} is perfect phylogeny. We name this problem as ENTRY-CHANGE PROBLEM (ECP).

2.8.1 The Character-Removal Problem

Given a genotyped matrix G , which does not admit a PP, we aim to find the smallest set of mutation loci to remove such that the final result admits a perfect phylogeny. The three-gametes rule is violated by the pairs of columns (loci) whose corresponding rows (cells) contain all three configurations (0, 1), (1, 0), and (1, 1). As Gusfield *et al.* [11] showed, there is an efficient ILP formulation for solving this problem.

Let $\delta(i)$ be a binary variable used to indicate whether or not loci i will be removed. Then for each pair of mutation loci (p, q) , $p, q \in \{1, \dots, m\}$ which violate three-gametes rule, we add the inequality $\delta(p) + \delta(q) \geq 1$. Finally, we minimize the objective function $\sum_{i=1}^m \delta(i)$. An example of CRP is shown in Table 1.

■ **Table 1** Character-Removal algorithm selects two loci, 2 and 4 to remove from G .

	locus1	locus2	locus3	locus4	locus5
cell1	1	1	1	0	0
cell2	1	1	1	0	0
cell3	0	0	0	1	0
cell4	1	0	0	0	1
cell5	0	1	0	1	1

2.8.2 The Entry-Change Problem

We define the ENTRY-CHANGE PROBLEM as inferring a matrix \mathcal{G} from G by changing the entries of G such that \mathcal{G} is a PP. Here, the character-removal problem plays an important role by returning the set of loci to be removed from G in order to admit a PP. As a result, instead of the whole matrix G , ECP can be applied to this set of loci. This in turn reduces the search space for solving ECP. Let $R = \{r_1, \dots, r_l\}$ denote this set of mutations.

The ILP for ECP introduces variables $Change(i, j) \in \{0, 1\}$ for each cell $i \in \{1, \dots, n\}$ and $j \in R$ indicating whether or not the entry $G(i, j)$ will be changed. We also use the variables $B(p, q, a, b) \in \{0, 1\}$ described in section 2.5 with the same definition. The program next creates inequalities that force the binary variable $B(p, q, a, b)$ to 1 if changing (or not changing) either or both entries $G(i, p)$ and $G(i, q)$ produces the combination (a, b) . To explain the formulation in detail, consider the pair of columns shown in Table 2.

■ **Table 2** Five rows and two columns in the input G of ECP.

	locus1	locus2
cell1	1	1
cell2	1	1
cell3	0	0
cell4	1	0
cell5	0	1

The first row creates the combination (1,1) between locus 1 and 2 if the entry $G(1, 2)$ does not change. This condition can be formulated as the following inequality:

$$B(1, 2, 1, 1) \geq 1 - Change(1, 2) \quad (16)$$

And changing the entry $G(1, 2)$ creates the combination (1, 0):

$$B(1, 2, 1, 0) \geq Change(1, 2) \quad (17)$$

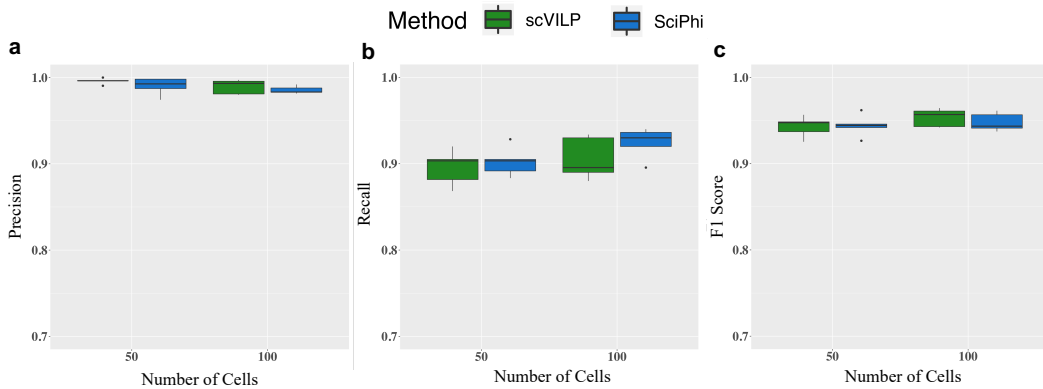
The constraints are built similarly, according to the other rows and columns. The constraint on variables $B(p, q, a, b)$ which guarantees PP is the same as the inequality 14. The objective function is to Minimize $\sum_{i=1}^n \sum_{j=1}^l Change(i, j)$ where n is the number of cells and l is the number of problematic loci.

While [3] presented an ILP solution for finding a minimum flip supertree admitting a PP, our combination of character removal and entry change problem does not guarantee a minimum flip supertree. Here our goal is to find a Perfect phylogeny supertree and our two-step approach guarantees to achieve that and at the same time restricting ECP to the sites selected by character removal results in reduction of runtime.

3 Results

3.1 Benchmarking on Simulated Data

To benchmark the performance of our approach and compare it with that of SCIΦ we first ran both of the methods on simulated datasets for which the ground truth is known. For generating simulated datasets, we used the single-cell read count simulator module introduced in [23]. The simulator first generates a model tumor tree, which starts from a progenitor cell without any mutation. Cells evolve along the branches of the tree and the mutations are acquired along these branches. The leaves of the tree represent the single cells. For simulating allelic dropout, mutations present in the cells at the leaves of the phylogeny are dropped out randomly with probability β ($\frac{\beta}{2}$ of the mutations become wild type and $\frac{\beta}{2}$ become homozygous). This simulator mimics the noisy MDA process [4] and accounts for the sequencing error whose corresponding probability values were set to 5×10^{-7} and 10^{-3} respectively. The MDA process was repeated 50 times as suggested by the authors of SCIΦ [23]. The mean and variance associated with the coverage distribution were set to 25 and 50 respectively. Finally, we also introduced 10% missing data.



■ **Figure 1** Performance of scVILP and SCI Φ on simulated data with different number of cells. The number of mutations is 100. Performance measured as (a) precision, (b) recall, and (c) F1 score.

We investigated how each method performs on datasets carrying different number of cells. The number of cells in the datasets, n , was varied as $n = 50$, and $n = 100$. The number of sites, m , was set at $m = 100$. We generated 5 datasets for each value of n . The rate of allelic dropout was chosen from the values $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. SCI Φ was run according to the authors' recommendation with an additional filter of requiring at least two cells to show variant read count of at least three. Our method scVILP was run with allelic dropout values given as input parameters to the program. In order to emulate the real settings where we do not know the exact values of the allelic dropouts, we added noise to their true values in the same way described in [16]. To add noise to the dropout values used as input to scVILP, a random value was sampled from a normal distribution with mean 1 and standard deviation 0.1 from the interval $(0.5, 2)$ and multiplied with the dropout values. The parameter μ_0 was empirically determined (by varying between $10^{-8} - 10^{-3}$) and set to 10^{-5} . The value of the parameter μ_1 was set to 0.5 assuming heterozygous somatic mutation. The results are shown in Fig. 1. Performance of each method was measured in terms of precision and recall. Precision is defined as $\frac{TP}{TP+FP}$, whereas recall is defined as $\frac{TP}{TP+FN}$ (TP: true positive, FP: false positive and FN: false negative). For each setting, scVILP achieved similar or better precision than SCI Φ . On the other hand, SCI Φ achieved a little better recall than scVILP for $n = 100$, but scVILP's recall was comparable to that of SCI Φ for $n = 50$. Overall, both methods achieved similar precision and recall. This can be observed when comparing the F1 measure, which is the harmonic mean of precision and recall. However, scVILP's median F1 score was marginally better than that of SCI Φ for each setting. These experiments show that both of these methods achieve similar accuracy in detecting the mutations. Since both of these methods utilize the underlying phylogenetic tree (assuming infinite sites assumption) in inferring the variants, both of them can identify mutations in a particular cell even if there is very little or missing variant support at a specific locus.

However, the major difference between these two methods lie in the use of the inference algorithm. While SCI Φ uses a Markov Chain Monte Carlo algorithm for sampling from the posterior distribution, scVILP solves the problem deterministically using ILP. Consequently, scVILP is much faster in finding the solution as can be seen in the runtime comparison (Fig. 2). For $n = 50$, scVILP was on average ~ 38 times faster than SCI Φ , while for $n = 100$, it was on average ~ 31 times faster than SCI Φ .

Allelic dropout is a major source of error in single-cell data [19]. We further assessed scVILP's performance under different values of allelic dropout rate. We generated 5 datasets for each value of allelic dropout in $\{0.2, 0.3\}$. For these datasets, we used $n = 50$ and $m = 100$.

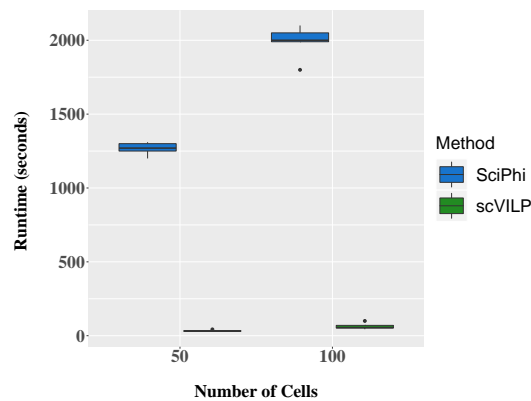


Figure 2 Runtime comparison for scVILP and SciPhi on simulated data with different number of cells. The number of mutations is 100.

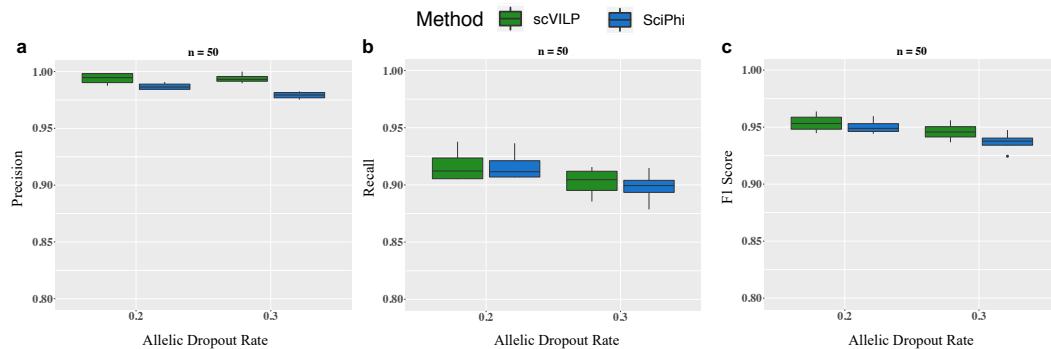
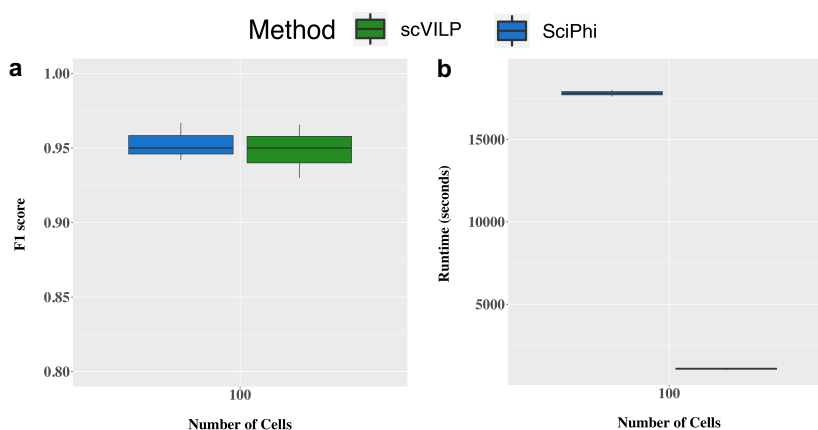


Figure 3 Performance of scVILP and SciPhi on simulated data with different allelic dropout rates. The number of mutations is 100 and number of cells is 50. Performance measured as (a) precision, (b) recall, and (c) F1 score.

Both scVILP and SciPhi were compared on these datasets in terms of precision and recall (Fig. 3). The F1 score for each method decreased with an increase in the allelic dropout rate. For each setting of allelic dropout rate, scVILP had slightly higher precision than that of SciPhi. Recalls for both methods were comparable; however, scVILP had slightly better recall for the datasets with higher allelic dropout rate. This experiment demonstrates that scVILP performs robustly for varying allelic dropout rate.

Due to the high memory consumption of ILP-solvers, straightforward application of scVILP might run into memory issues for very large number of mutations that can occur in single-cell WES datasets. For addressing such cases, we developed the Perfect Phylogeny supertree algorithm that should run without any memory issue, reducing the running time while maintaining tolerable tradeoff in accuracy. To analyze its performance, we generated three datasets containing 100 cells and 1000 mutation loci with allelic dropout chosen from $\{0, 0.15, 0.25\}$. We compared scVILP against SciPhi on these datasets (Fig. 4). While scVILP and SciPhi had similar F1 score for these datasets, scVILP had significantly shorter runtime compared to SciPhi. For these datasets, direct ILP formulation of scVILP failed to run on a machine with 32GB RAM, but the Perfect Phylogeny supertree algorithm, by dividing the original matrix into 5 submatrices, ran successfully with 11GB peak memory usage. Since SCS technologies are generating SNV data on hundreds of cells, but not yet on thousands of cells, we did not investigate the performance of scVILP and SciPhi by simulating larger number of cells.



■ **Figure 4** Performance of scVILP’s Perfect Phylogeny supertree algorithm compared to SCI Φ on simulated data with large number of mutations. The number of mutations is 1000 and the number of cells is 100. (a) comparison of F1 score, (b) runtime comparison.

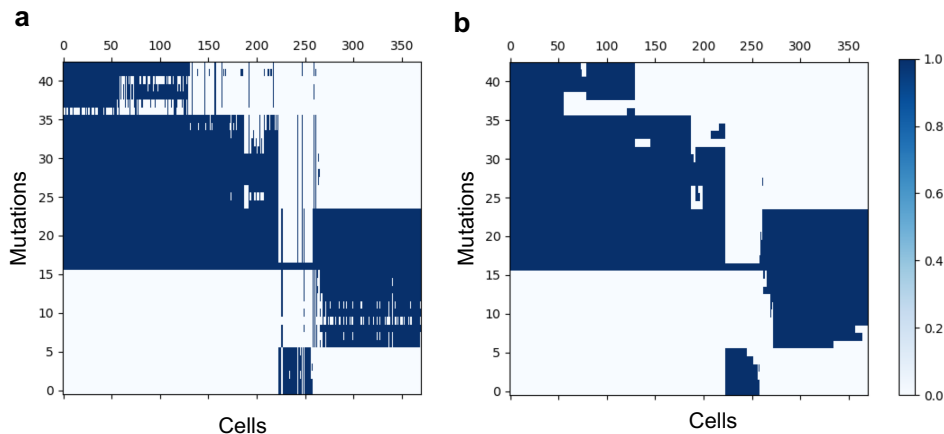
3.2 Application of scVILP on Real Data

Finally, we applied scVILP on a human cancer dataset. This dataset consists of 370 single cells and 43 mutation loci from a high-grade serous ovarian cancer patient described in [17]. For comparison, we also ran SCI Φ on this dataset and the mutation calls for both methods are shown in Fig. 5. Mutation calls for both these methods were mostly similar. The underlying phylogenetic trees inferred by these methods were also similar (Robinson-Foulds distance 0.123, computed by PAUP[24]) indicating that multiple perfect phylogenies fit the data. We further analyzed the mutation calls that were different between the methods. A large number of mutations that were genotyped as 1 by SCI Φ and 0 by scVILP had high reference read count and low variant read count indicating scVILP’s genotyping might have been more accurate for these sites. Both methods were run on a PC with 8GB RAM and 2 GHz Intel Core i5 processor. scVILP significantly outperformed SCI Φ in terms of runtime. The runtime for scVILP was around 30 minutes compared to the nearly 12 hours runtime of SCI Φ .

4 Conclusions

Here, we introduced scVILP, a novel combinatorial approach for jointly inferring the mutations in single cells and a perfect phylogeny that connects the cells. Using a novel Integer Linear Program, our method infers the mutations in individual cells assuming they evolve along a perfect phylogeny. scVILP probabilistically models the observed read counts but combinatorially solves the joint inference problem.

We compared scVILP to SCI Φ [23] on both simulated and real datasets. For the simulated data, both methods performed similarly in terms of precision and recall (as shown by their F1 scores). However, scVILP significantly outperformed SCI Φ in terms of runtime. Similarly for real human cancer dataset, scVILP achieved a solution much faster compared to SCI Φ . For the real dataset, SCI Φ produced different solutions in different runs because of its probabilistic nature. For the real dataset, even though mutation calls from scVILP and SCI Φ were similar, we also observed certain differences indicating that our deterministic algorithm and SCI Φ ’s probabilistic inference do not necessarily arrive at a unique solution in spite of the use of underlying perfect phylogenetic tree. This also indicates that potentially, multiple perfect phylogenies can fit the data well necessitating the development of methods that also assess uncertainty in the inferences.



■ **Figure 5** Summary of the mutation calls from SCIΦ and scVILP on a dataset consisting of 370 cells from a patient with high-grade serous ovarian cancer [17]. Deep blue and white entries in the heatmaps represent mutations states 1 and 0, respectively. (a) Mutation calls of SCIΦ. (b) Mutation calls of scVILP.

Our method could potentially be improved by including violations of the perfect phylogeny. Tumor phylogeny studies [30, 13, 26] show possible violations of ISA in single-cell sequencing datasets. More comprehensive evaluation of these violations could be performed by analyzing the read counts in conjunction with tumor phylogeny inference. Another improvement would be the inclusion of copy number data.

As single-cell sequencing becomes more high-throughput producing thousands of cells, scVILP will be very helpful for identifying the mutations from such larger datasets. scVILP is very fast and accurate in detecting the mutations as well as the phylogeny underlying the cells. Addressing both these problems in a single combinatorial framework will be very helpful for quick analysis of large datasets for understanding tumor heterogeneity.

References

- 1 Niko Beerenwinkel, Roland F Schwarz, Moritz Gerstung, and Florian Markowetz. Cancer evolution: mathematical models and computational inference. *Systematic Biology*, 64(1):e1–e25, 2014.
- 2 Rebecca A Burrell and Charles Swanton. Tumour heterogeneity and the evolution of polyclonal drug resistance. *Molecular Oncology*, 8(6):1095–1111, 2014.
- 3 Markus Chimani, Sven Rahmann, and Sebastian Böcker. Exact ILP solutions for phylogenetic minimum flip problems. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pages 147–153. ACM, 2010.
- 4 Frank B Dean, John R Nelson, Theresa L Giesler, and Roger S Lasken. Rapid amplification of plasmid and phage DNA using phi29 DNA polymerase and multiply-primed rolling circle amplification. *Genome Research*, 11(6):1095–1099, 2001.
- 5 Mark A. DePristo, Eric Banks, Ryan Poplin, Kiran V. Garimella, Jared R. Maguire, Christopher Hartl, Anthony A. Philippakis, Guillermo del Angel, Manuel A. Rivas, Matt Hanna, Aaron McKenna, Tim J. Fennell, Andrew M. Kernytzsky, Andrey Y. Sivachenko, Kristian Cibulskis, Stacey B. Gabriel, David Altshuler, and Mark J. Daly. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, 43(5):491–498, 2011.

- 6 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.
- 7 Xiao Dong, Lei Zhang, Brandon Milholland, Moonsook Lee, Alexander Y Maslov, Tao Wang, and Jan Vijg. Accurate identification of single-nucleotide variants in whole-genome-amplified single cells. *Nature Methods*, 14(5):491, 2017.
- 8 Charles Gawad, Winston Koh, and Stephen R. Quake. Dissecting the clonal origins of childhood acute lymphoblastic leukemia by single-cell genomics. *Proceedings of the National Academy of Sciences*, 111(50):17947–17952, 2014.
- 9 Moritz Gerstung, Christian Beisel, Markus Rechsteiner, Peter Wild, Peter Schraml, Holger Moch, and Niko Beerenwinkel. Reliable detection of subclonal single-nucleotide variants in tumour cell populations. *Nature Communications*, 3:811, 2012.
- 10 Robert J. Gillies, Daniel Verduzco, and Robert A. Gatenby. Evolutionary dynamics of carcinogenesis and why targeted therapy does not work. *Nat Rev Cancer*, 12(7):487–493, July 2012.
- 11 Dan Gusfield, Yelena Frid, and Dan Brown. Integer programming formulations and computations solving phylogenetic and population genetic problems with missing or genotypic data. In *International Computing and Combinatorics Conference*, pages 51–64. Springer, 2007.
- 12 Katharina Jahn, Jack Kuipers, and Niko Beerenwinkel. Tree inference for single-cell data. *Genome Biology*, 17(1):1–17, 2016.
- 13 Jack Kuipers, Katharina Jahn, Benjamin J Raphael, and Niko Beerenwinkel. Single-cell sequencing data reveal widespread recurrence and loss of mutational hits in the life histories of tumors. *Genome research*, 27(11):1885–1894, 2017.
- 14 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.
- 15 Jian Ma, Aakrosh Ratan, Brian J. Raney, Bernard B. Suh, Webb Miller, and David Haussler. The infinite sites model of genome evolution. *Proceedings of the National Academy of Sciences*, 105(38):14254–14261, 2008.
- 16 Salem Malikic, Simone Ciccolella, Farid Rashidi Mehrabadi, Camir Ricketts, Md Khaledur Rahman, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S Cenk Sahinalp. PHISCS-a combinatorial approach for sub-perfect tumor phylogeny reconstruction via integrative use of single cell and bulk sequencing data. *BioRxiv*, page 376996, 2018.
- 17 Andrew McPherson, Andrew Roth, Emma Laks, Tehmina Masud, Ali Bashashati, Allen W Zhang, Gavin Ha, Justina Biele, Damian Yap, Adrian Wan, et al. Divergent modes of clonal spread and intraperitoneal mixing in high-grade serous ovarian cancer. *Nature Genetics*, 48(7):758, 2016.
- 18 Lauren M.F. Merlo, John W. Pepper, Brian J. Reid, and Carlo C. Maley. Cancer as an evolutionary and ecological process. *Nat Rev Cancer*, 6(12):924–935, December 2006.
- 19 Nicholas Navin. Cancer genomics: one cell at a time. *Genome Biology*, 15(8):452–465, 2014.
- 20 Nicholas E. Navin. The first five years of single-cell cancer genomics and beyond. *Genome Research*, 25(10):1499–1507, October 2015.
- 21 PC Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976.
- 22 Andrew Roth, Jaswinder Khattra, Damian Yap, Adrian Wan, Emma Laks, Justina Biele, Gavin Ha, Samuel Aparicio, Alexandre Bouchard-Cote, and Sohrab P. Shah. PyClone: statistical inference of clonal population structure in cancer. *Nat Meth*, 11(4):396–398, April 2014.
- 23 Jochen Singer, Jack Kuipers, Katharina Jahn, and Niko Beerenwinkel. Single-cell mutation identification via phylogenetic inference. *Nature Communications*, 9(1):5144, 2018.
- 24 DL Swafford. PAUP*. Phylogenetic analysis using parsimony (* and other methods). *Vol. Sinauer Associates, Sunderland, MA*, 2002.

- 25 Charles Swanton. Intratumor heterogeneity: evolution through space and time. *Cancer research*, 72(19):4875–4882, 2012.
- 26 Lili Wang, Jean Fan, Joshua M Francis, George Georghiou, Sarah Hergert, Shuqiang Li, Rutendo Gambe, Chensheng W Zhou, Chunxiao Yang, Sheng Xiao, et al. Integrated single-cell genetic and transcriptional analysis suggests novel drivers of chronic lymphocytic leukemia. *Genome research*, 27(8):1300–1311, 2017.
- 27 Yong Wang and Nicholas E. Navin. Advances and applications of single-cell sequencing technologies. *Molecular Cell*, 58(4):598–609, 2015.
- 28 Hamim Zafar, Nicholas Navin, Ken Chen, and Luay Nakhleh. SiCloneFit: Bayesian inference of population structure, genotype, and phylogeny of tumor clones from single-cell genome sequencing data. *bioRxiv*, page 394262, 2018.
- 29 Hamim Zafar, Nicholas Navin, Luay Nakhleh, and Ken Chen. Computational approaches for inferring tumor evolution from single-cell genomic data. *Current Opinion in Systems Biology*, 7:16–25, 2018.
- 30 Hamim Zafar, Anthony Tzen, Nicholas Navin, Ken Chen, and Luay Nakhleh. SiFit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome Biology*, 18(1):178, 2017.
- 31 Hamim Zafar, Yong Wang, Luay Nakhleh, Nicholas Navin, and Ken Chen. Monovar: single-nucleotide variant detection in single cells. *Nature Methods*, 13(6):505–507, June 2016.

Topological Data Analysis Reveals Principles of Chromosome Structure in Cellular Differentiation

Natalie Sauerwald

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, 15213, USA
nsauerwald@cmu.edu

Yihang Shen

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, 15213, USA
yihangs@andrew.cmu.edu

Carl Kingsford¹

Computational Biology Department, School of Computer Science,
Carnegie Mellon University, Pittsburgh, 15213, USA
carlk@cs.cmu.edu

Abstract

Topological data analysis (TDA) is a mathematically well-founded set of methods to derive robust information about the structure and topology of data. It has been applied successfully in several biological contexts. Derived primarily from algebraic topology, TDA rigorously identifies persistent features in complex data, making it well-suited to better understand the key features of three-dimensional chromosome structure. Chromosome structure has a significant influence in many diverse genomic processes and has recently been shown to relate to cellular differentiation. While there exist many methods to study specific substructures of chromosomes, we are still missing a global view of all geometric features of chromosomes. By applying TDA to the study of chromosome structure through differentiation across three cell lines, we provide insight into principles of chromosome folding and looping. We identify persistent connected components and one-dimensional topological features of chromosomes and characterize them across cell types and stages of differentiation.

Availability: Scripts to reproduce the results from this study can be found at <https://github.com/Kingsford-Group/hictda>

2012 ACM Subject Classification Applied computing → Computational biology; Applied computing

Keywords and phrases topological data analysis, chromosome structure, Hi-C, topologically associating domains

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.23

Funding This work has been supported in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through Grant GBMF4554 to C.K., and by the US National Institutes of Health (R01HG007104 and R01GM122935). Research reported in this publication was supported by the NIGMS of the NIH under award number P41GM103712 and by the Richard K. Mellon Presidential Fellowship in Life Sciences to N.S. This work was partially funded by The Shurl and Kay Curci Foundation.

Acknowledgements The authors would like to thank Alessandro Bertero and William S. Noble for useful information about their data, and Guillaume Marçais for comments on the manuscript.

¹ Corresponding author



1 Introduction

The three-dimensional shape of chromosomes has significant influence in many critical cellular processes, including gene expression and regulation [7, 14, 27], replication timing [25, 2], and overall nuclear organization [10]. The wide range of processes related to chromosome structure suggests that understanding this component is crucial to a broader explanation of many genomic mechanisms, yet it remains a challenge to study this complex system.

In particular, the process of differentiation, by which a cell changes to a new cell type, is critical to all multi-cellular life, but the mechanisms behind this process remain an active field of research with recent work suggesting a role for chromosome structure in this process. Structural changes have been observed in chromosomes through lineage specification, both across several stages of human cardiogenesis [16] as well as across human embryonic stem cells (ESCs) and four human ES-cell-derived lineages [12]. Fields et al. (2017) [16] identified both global and local structural dynamics, observing transitions from repressive to active compartments around cardiac-specific genes as they are upregulated through differentiation. Dixon et al. (2015) [12] also noted structural dynamics across hierarchical scales during development, with some corresponding gene expression changes.

Chromosome structure can be measured by a number of variants of the chromosome conformation capture protocol [11], including Hi-C [21] which permits genome-wide measurements of the chromosomal architectures of a population of cells. Hi-C quantifies physical proximity by counting cross-linkage frequencies between genomic segments. Because of the dependence on cross-linking, which is likely to induce both false positive and false negative contacts, the heterogeneity within cell populations, and the large scale and complexity of the system, Hi-C can be very challenging to analyze. Many methods have focused on identifying local structures [17], however it has proven challenging to study large-scale structures across the entire genome.

A class of techniques called “Topological Data Analysis” (TDA) has gained prominence recently as a generalized, mathematically grounded set of methods for identifying and analyzing the topological and geometric structures underlying data. Emerging from work in applied topology and computational geometry, TDA aims to infer information about the robust structures of complex data sets [9]. These methods have already been applied to various biological contexts [3], including in studies of gene expression at the single cell level [28], viral reassortment [8], horizontal evolution [4], cancer genomics [24, 1], and other complex diseases [20, 18]. Similar methods have also been used in tools to enable large-scale biological database searching [32]. The two main methods of TDA are Mapper, a dimensionality reduction framework and visualization method, and persistent homology, an algorithm for extracting geometric and topological structures which describe the underlying data.

Given its rigorous mathematical foundation and ability to identify important topological structures, TDA is very well-suited to the analysis of Hi-C data. Emmett et al. (2015) [15] first applied these methods to human Hi-C data, though computational limitations at the time restricted this analysis to only one chromosome at 1Mb resolution. More recently, the Mapper method of TDA was used to analyze the similarities between single-cell Hi-C maps [6]. Carriere and Rabadan (2018) [6] first computed pairwise distances between all single-cell Hi-C contact matrices, and applied TDA to the distance matrix between single cells rather than applying TDA directly to the Hi-C data, analyzing the results with Mapper. In this paper, we use persistent homology to identify geometric structures in human chromosomes directly from Hi-C data, and study how they change throughout lineage specification and differentiation.

This work presents the first use of TDA to study the chromosome structures of all 22 human autosomal chromosomes, providing insight into the structural changes involved in cellular differentiation. We identify hundreds of thousands of zero- and one-dimensional features of chromosome structure across 14 cell types, and describe the patterns underlying geometric structures of Hi-C data. We characterize the one-dimensional hole structures identified by TDA, noting that many of the patterns we observe can be explained by the linearity of the chromosome. Additionally, we compare the topologies of 14 cell types representing various stages of differentiation and various cell lines, and note that the topological similarity is largely dictated by cell line rather than differentiation stage.

2 Methods

2.1 Overview of TDA

TDA is based on the premise that data points are sampled from an unknown continuous geometric structure that can be described by topological properties preserved under continuous deformations of the space. These properties can include the number and size of connected components, loops or holes the structure contains. TDA approximates a continuous geometry by building a *simplicial complex*, or a network of edges and triangles, from the nodes of the given data points. Due to the computational complexity of TDA and the challenges of interpreting larger-dimensional features, we focus only on 0- and 1-dimensional features here, which requires generating simplices with maximum dimension 2, i.e., nodes, edges, and triangles.

A Vietoris-Rips (VR) complex is then generated, which is a set of simplices produced by adding edges between all nodes with distance less than a given α , and a triangle between all sets of three nodes for which each pair is no more than α apart. Together these components describe a structure built from the data, from which the topological features of the underlying space can be described and quantified through a process called *persistent homology*.

► **Definition (Vietoris-Rips complex).** *Given a set of points X in a metric space (M, d) and a real number $\alpha \geq 0$, The Vietoris-Rips complex is the set of simplices $\{[x_0, \dots, x_k]\}$ such that $d(x_i, x_j) \leq \alpha$ for all (i, j) , with k less than or equal to a given maximum dimension [9].*

The analysis of simplicial complexes and their topological properties is based in homology theory, which defines the topological properties of any given dimension of a space. These properties can be represented by homology groups $H_0(X), H_1(X), H_2(X), \dots, H_n(X)$. A homology group H_k represents k -dimensional “holes”. For example, H_0 represents the connected components of the VR complex, H_1 represents one-dimensional loops, and H_2 represents two-dimensional voids [30].

Given a set of data points X , we build VR complexes for different values of parameter α . The basis of persistent homology is the idea that features that persist in the VR complexes across values of α are the key topological features of the space generated by the data. Conversely, features that only appear for few values of α are often filtered out or ignored as methodological artifacts. A feature from persistent homology is therefore described by an interval $[b, d]$, where b represents the birth time of the feature, or the smallest value of α at which the feature is found, and d , called death time, the smallest value of α at which the feature no longer exists. These features are visualized in two ways: *persistence diagrams* and *barcode plots*. Persistence diagrams are sets of (b, d) points in the Euclidean half-plane above the diagonal (see Figure 1(a) for an example). Barcode plots display the same information, but with each homology group shown as an interval $[b, d]$. These barcode lines are plotted at

different heights, often ordered by death time d for visualization purposes, but the y values are arbitrary (see Figure 1(b) for an example). For more technical details on TDA and persistent homology, see for example Carlsson (2014) [5] or Wasserman (2018) [31].

2.2 Applying TDA to Hi-C

The methods of TDA use a distance matrix that describes the distances between all data points. Although Hi-C data is interpreted as describing the 3D distances between chromosomal segments, the values of a Hi-C matrix are contact counts rather than distance values, where a high contact count implies a low distance. We use the following transformation to convert a normalized Hi-C matrix M to a distance matrix K :

$$K_{i,j} = 1 - \begin{cases} 1 & i = j \\ \frac{1}{m} \log(M_{i,j} + 1) & i \neq j \end{cases}$$

where $m = 1.01 \max_{i,j \leq D} (\log(M_{i,j}) + 1)$, and D is the number of rows in the contact matrix. A pseudo-count of 1 is added to all off-diagonal values in the Hi-C matrix to avoid taking a logarithm of zero, and the factor of 1.01 is included to ensure that all distances where $i \neq j$ are nonzero. Three other distance transformations were tested on a subset of our data, and the patterns observed were qualitatively similar across all four.

Note that this transformation does not return a metric, as Hi-C values themselves do not satisfy the triangle inequality. A short study of the practical effect of violating the metric assumption in TDA can be found in the appendix, which suggests that at least in simple cases, persistent homologies are somewhat robust to distances that do not satisfy the triangle inequality.

We use GUDHI [22], a Python library for TDA, to compute persistent homology from these distance matrices at 100kb resolution.

2.3 Loop trace-back algorithm

The H_1 structures identified by TDA represent “loops” in the input data, or one-dimensional holes in the simplicial complex. We will use the term loop interchangeably with H_1 structure but it is important to note that these are not loops in the traditional sense of chromatin loops. The loops identified by TDA may be surrounded by non-consecutive genomic segments, unlike the continuous loops generally studied in chromatin.

TDA does not identify the data points involved in each H_k , as each element corresponds to a homology class rather than an individual structure. Each H_1 class represents the set of loops around a one-dimensional topological hole, not a specific loop structure in the data. In order to locate a representative loop for each homology class, we first define an “optimal loop” as the loop in the homology class with the shortest total distance. In the Hi-C case, loop edges are connections between two chromosome bins i and j , and their edge weights are given by $K_{i,j}$ in the distance matrix. We therefore look for the loop that minimizes the sum of edge weights over all loops in the homology class.

In order to identify the optimal loop of each homology class H_1 we use the following loop trace-back algorithm, based on identifying a shortest path through the distance matrix K , similar to the method used by Emmett et al. (2015) [15]. One major difference in our algorithm is that we do not perturb the Hi-C matrices, ensuring that we preserve all spatial relationships among genomic loci. We additionally provide a proof of the correctness of our loop trace-back algorithm (see Appendix, section A1). The formal loop trace-back algorithm is described as follows:

- Step 1: For each H_1 persistent structure, identify the unique edge which forms at the birth time and generates the persistent homology. We will refer to the two nodes of this edge as V_1 and V_2 .
- Step 2: Construct a weighted graph G from the distance matrix, where nodes are chromosome bins and edges are weighted by the distance $K_{i,j}$, including only edges where $K_{i,j}$ is less than the distance between V_1 and V_2 .
- Step 3: Find the shortest (lowest weight) path from V_1 to V_2 on the graph using Dijkstra's algorithm, and return this path plus the edge between V_1 and V_2 .

This algorithm is based on one key assumption: there is only one edge with weight equal to the birth time of each H_1 persistent structure. In the Hi-C case, this is not a strong assumption as only about 2% of the persistent structures were formed at values with more than one edge. In all of these cases, there was only one other edge with the same value. The TDA software used here reports which edge formed each persistent structure, so we take that edge as our starting point for the algorithm in these ambiguous cases. We still guarantee that the loop returned will be a member of the correct homology class, however there exist edge cases in which this loop may not represent the minimum weight path. For applications that violate this assumption more frequently, this algorithm may be unsuitable.

2.4 Null models

In order to understand the TDA loop structures, three separate null models of distance matrices representing various properties of the Hi-C data were generated and analyzed with TDA. The three null models are defined as follows:

- Random permutation: all Hi-C distance values are permuted randomly, preserving only the symmetry of the distance matrix and the values themselves.
- Edge permutation: the distance values along each row of the distance matrix were permuted randomly, preserving both the degree of and set of distances for each node but randomly changing their assignments. The corresponding columns were permuted in the same way to preserve symmetry.
- Linear dependence: a new distance matrix is created, in which each diagonal beyond the main diagonal preserves the same mean and standard deviation of the original data, with Gaussian noise added. The dominant pattern of the Hi-C distance matrices is a decrease in distance as the difference between the row index and column index increases. This model represents this same pattern, but does not include any additional structural features of chromosomes.

The properties of barcode plots and persistence diagrams of these null models were compared with those of true Hi-C data.

2.5 Bottleneck distance to compare persistence diagrams

In order to derive stability results for TDA, Carlsson (2014) [5] proposed a metric called the *bottleneck distance* that quantifies the difference between two persistence diagrams. The bottleneck distance is based on a perfect bipartite matching g between two persistence diagrams dgm_1 and dgm_2 , where points in either persistence diagram can also be matched to any point along the diagonal. The formula for computing the bottleneck distance d_B is:

$$d_B(\text{dgm}_1, \text{dgm}_2) = \inf_{\text{matching } g} \left[\max_{(x,y) \in g} \|x - y\|_\infty \right].$$

■ **Table 1** All Hi-C samples used for this study.

Sample name	Description	SRA Accessions	Study
RUES2 ESC	embryonic stem cell	SRX3375347, SRX3375348	[16]
RUES2 MES	mesoderm	SRX3375349, SRX3375350	[16]
RUES2 CP	cardiac progenitor	SRX3375351, SRX3375352	[16]
RUES2 CM	cardiac myocyte	SRX3375353, SRX3375354	[16]
RUES2 FetalHeart	fetal heart tissue	SRX3375355, SRX3375356	[16]
WTC11 PSC	pluripotent stem cell	SRX4958481, SRX4958482	[16]
WTC11 MES	mesoderm	SRX4958483, SRX4958484	[16]
WTC11 CP	cardiac progenitor	SRX4958485, SRX4958486	[16]
WTC11 CM	cardiac myocyte	SRX4958487, SRX4958488	[16]
H1 ESC	embryonic stem cell	SRX378271, SRX378272	[12]
H1 ME	mesendoderm	SRX378273, SRX378274	[12]
H1 MS	mesenchymal stem cell	SRX378275, SRX378276	[12]
H1 NP	neural progenitor	SRX378277, SRX378278	[12]
H1 TB	trophoblast-like cells	SRX378279, SRX378280	[12]

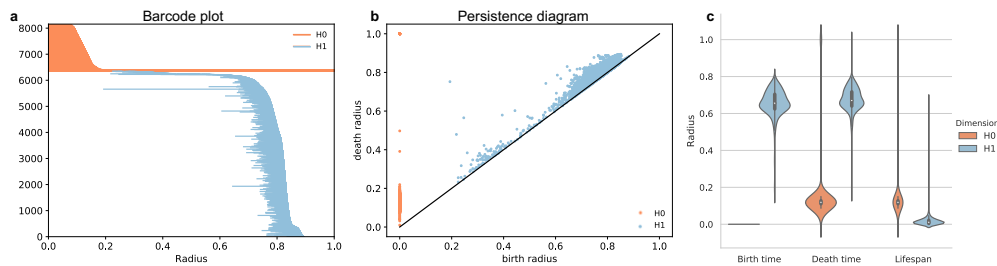
The bottleneck distance quantifies the similarity between two persistence diagrams by the maximum distance between two points in the best matching.

3 Results

3.1 Data

We analyzed Hi-C samples from 14 conditions, representing several differentiation lineages from two different studies [16, 12]. Two of these lineages represent paths through human cardiogenesis, starting with stem cells and continuing through the mesoderm (MES), cardiac progenitor (CP), and cardiac myocyte (CM) stages. One line, from RUES2 cells, begins with embryonic stem cells (ESC), and also includes a fetal heart tissue sample. The study authors also collected data from WTC11 cells, beginning with a human induced pluripotent stem cell (PSC), then collecting data at the same stages as the RUES2 cells: MES, CP and CM [16]. The third Hi-C data set represents still another differentiation starting point, using H1 ESC cells to generate four human ES-cell-derived lineages: mesendoderm (ME), mesenchymal stem (MS) cells, neural progenitor (NP) cells, and trophoblast-like (TB) cells [12]. All data is described in Table 1, including accession codes. Samples from all 14 conditions included two replicates each. All of the Hi-C data was processed from raw reads to normalized contact matrices at 100kb using the HiC-Pro pipeline [29] and iterative correction and eigenvector decomposition (ICE) normalization [19]. In order to maximize coverage, we combined all of the reads from replicates to produce one Hi-C matrix per sample.

We generated ten of each of the null models for every RUES2 cell type and each chromosome from 13 to 22. The null models on longer chromosomes proved not to be computationally feasible, but the patterns across the chromosomes that we were able to model were remarkably consistent (see Figure 7), suggesting that the additional data from all three null models on chromosomes 1 through 12 would follow similar patterns.



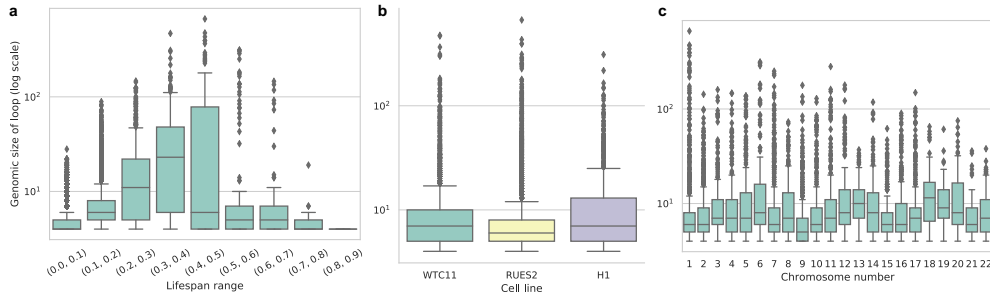
■ **Figure 1** Representative example of a barcode plot (a) and persistence diagram (b) from Hi-C data. These figures were created from chromosome 5 of WTC11 cardiac progenitor cells, and summarize the output from the persistent homology computation. Each point or bar represents one structure, defined by the radius at which the structure can first be seen (birth radius), and the last radius before the structure no longer exists (death radius). These figures are two ways to represent the same persistent homology information. (c) Distributions of birth, death, and lifespan values of all 14 cell types on all 22 chromosomes.

3.2 Persistent homology in Hi-C data

We visualize the persistent homology groups in the two ways described previously, persistence diagrams and barcode plots. We focus here on H_0 and H_1 structures and observe a very distinctive pattern in both structure classes across chromosomes and cell types in all of our Hi-C data (Figure 1c). The majority of H_0 structures disappear at a radius of somewhere between $\alpha = 0.1$ and $\alpha = 0.2$, suggesting that many new edges are formed near these values, and relatively few H_0 structures persist after this. TDA therefore quickly recovers the linear structure of the chromosome. The H_1 structures tend to have short lifespans (they are close to the diagonal in the persistence diagrams), and most are born at $\alpha \sim 0.6 - 0.8$, though there are consistently a few loops born earlier. A representative barcode plot and persistence diagram can be seen in Figure 1. Birth and death values to plot all barcode plots and persistence diagrams on all cell types and all chromosomes studied here is available at <https://github.com/Kingsford-Group/hictda>.

3.3 Characterization of loops

Using the loop trace-back algorithm described in Methods, we locate the representative genomic bins for each H_1 homology class identified and characterize their distributions across cell type, chromosomes, and differentiation stages. Recall that TDA “loops” are not traditional chromosome loops in the sense that they may be made up of non-consecutive genomic segments; across all of our data only 1868 of the 850188 total loops found are made up of consecutive genomic segments with mean length of only $\sim 625\text{kb}$. Given that most of the loops are not made of consecutive genomic segments, we will refer to their size as the number of bins involved in the loop rather than the genomic distance between the first and last bin. These loop structures are generally very small (mean $\sim 446.5\text{kb}$, or 4.47 bins), but this mean is largely dominated by very short-lived loops; 98.98% of all H_1 structures identified by TDA have a lifespan (difference between birth and death radius) of less than 0.1, which can be seen in both the barcode plot, where the majority of blue lines are very short, and the persistence diagram, where most blue dots are very close to the diagonal (Figure 1). We therefore look at these loop size distributions as a function of their lifespan (Figure 2a), noting that while the average loop is quite short, there are many much longer loops (the longest loop involves 2492 bins, equivalent to 24.92Mb) and the distributions



■ **Figure 2** Distributions of genomic loop sizes across loop lifespans, cell types, and chromosomes. For all three figures, the log-scaled y-axis represents the total number of genomic bins returned by the loop traceback method. (a) Loop distributions separated by their lifespans in the VR complex. Larger lifespans indicate loops that are present for a large range of α values. (b) Loop size distributions by cell type, excluding all loops with lifespans less than 0.1, which are likely to be artifacts. (c) Loop size distributions by chromosome, again excluding all loops with lifespans less than 0.1.

vary significantly between lifespan ranges. Removing the loops with lifespan less than 0.1 which are likely artifacts, the distributions of loop size do not vary much between cell types (Figure 2b), though there is some variation between chromosomes (Figure 2c). We did not observe any consistent trends in loop characteristics through differentiation across the three cell lines (Figure 6).

3.4 Loop patterns are largely dictated by the linearity of chromosomes

In order to further understand the H_1 structures, the patterns observed in real Hi-C data were compared to our null models. As a representative example, barcode plots of all loop structures of chromosome 14 in RUES2 MES cells can be seen in Figure 3a, along with the null models from this data.

One of the most striking patterns in comparing these null models to the true data is that the loops in Hi-C tend to be born at a much higher value of α , and survive only a short time, as noted previously. The model that most closely resembles this pattern is the linear dependence model, but the distribution of birth times shows that the linear dependence model has a long tail towards the earlier birth times which is absent in real Hi-C data (Figure 3b). The short life span, which will generally correspond to smaller loops, is also much more consistent with the linear dependence model, although somewhat more pronounced in Hi-C (Figure 3c). The fact that the linear dependence null model shows these same patterns as Hi-C data suggests that the linear property (nodes that are close together in index, or linear distance, are also close in 3D space) is sufficient to explain the short loops and birth times concentrated at high values of α . Long loops appear to be created when nodes with a large difference in their indices (large linear distance) are close together, as demonstrated by the loops with very long life spans in the two permutation models. This appears to be very rare in Hi-C data; the majority of loop interactions we observe are relatively short-lived and involve few genomic bins, consistent with findings from more traditional chromosome loop structures which have been shown to be almost exclusively between loci less than 2Mb apart [26].

The greatest difference between the linear dependence model and Hi-C data can be seen in the number of loops identified (Figure 3d). Although the linear dependence model is again the most similar to real data in this regard, it typically still contains over twice the number of loops as the corresponding Hi-C matrix. This observation could be explained by the

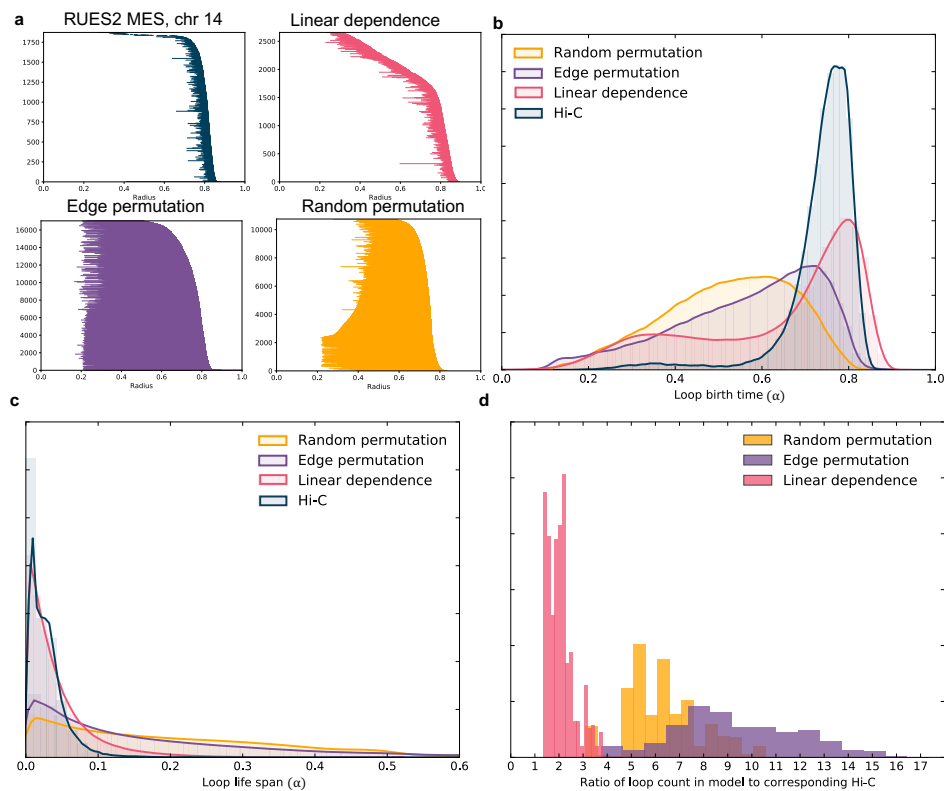
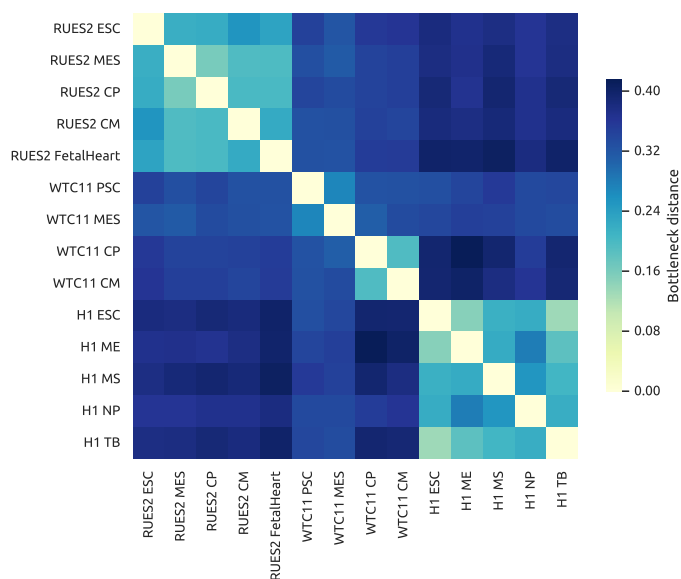


Figure 3 Loop analysis through multiple null models. (a) Barcode plots (showing only loop structures) from chromosome 14 of RUES2 MES cells, along with the corresponding barcode plots from the three null models. (b) Normalized histogram of the birth times of all loops in RUES2 data from true Hi-C and each null model. (c) Normalized histogram of loop life spans (length of a barcode line) shows that the loops from real Hi-C data are very small, similar to the linear dependence model. (d) Normalized histogram of the ratios of numbers of loops in each null model to the number of loops in the corresponding Hi-C matrix.

existence of topologically associating domains (TADs) or compartments in real chromosomes, which are absent from the linear dependence model but a defining characteristic of Hi-C. These structures serve to isolate chromosome segments from each other, which likely prevents the formation of loops between them. If loops can largely only be formed within a TAD or compartment, this would significantly restrict the total number of feasible loops which could explain the pattern we observe in the persistence diagrams of Hi-C data.

3.5 Comparing topologies across differentiation

Although there are some evident changes in topological structures measured by TDA through differentiation, the larger differences exist between the three main cell lines (RUES2, WTC11, and H1, see Figure 4). Interestingly, there does not seem to be any more similarity, measured by bottleneck distance averaged over all chromosomes, between cells at the same stages of differentiation across the three cell lines than cells at different stages of differentiation. For example, the distance value between the two cardiac progenitor samples from RUES2 and WTC11 appears no lower than the value between RUES2 CP and WTC11 CM cells, and H1 ESC and RUES2 ESC seem no more similar to each other than H1 ESC and RUES2



■ **Figure 4** Comparison of all 14 samples studied. This heatmap represents the bottleneck distances between each pair of samples in our data, averaged over all 22 chromosomes. With the exceptions of the high distances between the two later stages and two earlier stages of WTC11 differentiation, the pattern of low distance within one cell line dominates these comparisons. This pattern can be seen by the lighter blocks along the diagonal with limits corresponding to the changes in cell line.

MES or RUES2 ESC and H1 ME cells. Global topological features identified by TDA at the genome-wide scale therefore seem to be determined more by cell line than differentiation stage, suggesting that the structural changes involved in differentiation may be more localized or on a smaller scale than we have studied here.

4 Discussion

One of the major challenges in the application of TDA to Hi-C data is the computational complexity of the methods combined with the scale of Hi-C. For a more in-depth discussion of TDA complexity, see for example [23]. Due to computational limitations, the structures studied here are fairly large-scale; the Hi-C data analyzed is at a relatively low 100kb resolution, and our study only includes 14 samples. By studying smaller sections of the genome, perhaps near a gene of interest or within a particular structure of interest, higher resolution Hi-C or 5C could be used with TDA to identify small-scale topological structures. We were also only able to study intra-chromosomal matrices, but the topology of inter-chromosomal interactions would likely yield interesting insights as well though it would further increase the computational complexity of the problem. Another consequence of the computational complexity of TDA is our focus on only 0- and 1- dimensional features. Emmett et al. (2015) [15] speculate that the 2-dimensional voids may represent interesting biological features such as transcription factories. Future improvements in the data structures and algorithms underlying TDA would significantly improve our ability to study more aspects of the topology of the full genome.

The nature of Hi-C, as an experiment based on cross-linking over a full population, does not permit a transformation from the Hi-C counts to a true distance metric. While this did not seem to affect the results in two toy examples tested (see Section S2), the complexity of Hi-C may result in more unpredictable outcomes from violating the triangle inequality on

a much larger scale. One possibility to improve this concern is to use any of the methods that estimate a 3D structure from Hi-C, or select only the Hi-C values that satisfy a metric definition [13], and infer distances which would be geometrically consistent. However, this induces another source of error, and it is unclear whether the results would be more reliable.

We have presented the first application of TDA to study the topology of all 22 human autosomal chromosomes. By studying clusters and loops of 14 samples from various cell lines and stages of differentiation, we identify generative principles of chromosome structure. We describe the characteristics of loop structures in chromosomes, the majority of which are very short genomically but with a small number of extremely large outliers suggesting the presence of very long-distance chromosome interactions. Our models suggest that the linearity of the chromosome is sufficient to explain the short lifespan of its loops, but additional structural features specific to Hi-C likely lead to the relatively small number of loops and their late birth times. We also show that topological structure seems to be largely determined by cell line rather than stage of differentiation. TDA shows promise for further analysis of Hi-C data, especially as computational limitations are overcome permitting analysis of higher dimensional features at higher resolution.

Financial disclosure. C. K. is a co-founder of Ocean Genomics, Inc.

References

- 1 Javier Arsuaga, Tyler Borrman, Raymond Cavalcante, Georgina Gonzalez, and Catherine Park. Identification of copy number aberrations in breast cancer subtypes using persistence topology. *Microarrays*, 4(3):339–369, 2015.
- 2 Ferhat Ay, Timothy L Bailey, and William Stafford Noble. Statistical confidence estimation for Hi-C data reveals regulatory chromatin contacts. *Genome Research*, 24(6):999–1011, 2014.
- 3 Pablo G Cámara. Topological methods for genomics: present and future directions. *Current Opinion in Systems Biology*, 1:95–101, 2017.
- 4 Pablo G Camara, Daniel IS Rosenbloom, Kevin J Emmett, Arnold J Levine, and Raul Rabadan. Topological data analysis generates high-resolution, genome-wide maps of human recombination. *Cell Systems*, 3(1):83–94, 2016.
- 5 Gunnar Carlsson. Topological pattern recognition for point cloud data. *Acta Numerica*, 23:289–368, 2014.
- 6 Mathieu Carriere and Raul Rabadan. Topological Data Analysis of Single-cell Hi-C Contact Maps. *arXiv*, 2018. [arXiv:1812.01360](https://arxiv.org/abs/1812.01360).
- 7 Giacomo Cavalli and Tom Misteli. Functional implications of genome topology. *Nature Structural and Molecular Biology*, 20(3):290–299, 2013.
- 8 Joseph Minhow Chan, Gunnar Carlsson, and Raul Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, pages 18566—18571, 2013.
- 9 Frédéric Chazal and Bertrand Michel. An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists. *arXiv*, 2017. [arXiv:1710.04019](https://arxiv.org/abs/1710.04019).
- 10 Yu Chen, Yang Zhang, Yuchuan Wang, Liguang Zhang, Eva K Brinkman, Stephen A Adam, Robert Goldman, Bas van Steensel, Jian Ma, and Andrew S Belmont. Mapping 3D genome organization relative to nuclear compartments using TSA-Seq as a cytological ruler. *The Journal of Cell Biology*, 217(11):4025–4048, 2018.
- 11 Job Dekker, Karsten Rippe, Martijn Dekker, and Nancy Kleckner. Capturing chromosome conformation. *Science*, 295(5558):1306–1311, 2002.
- 12 Jesse R Dixon, Inkyung Jung, Siddarth Selvaraj, Yin Shen, Jessica E Antosiewicz-Bourget, Ah Young Lee, Zhen Ye, Audrey Kim, Nisha Rajagopal, Wei Xie, et al. Chromatin architecture reorganization during stem cell differentiation. *Nature*, 518(7539):331–336, 2015.
- 13 Geet Duggal, Rob Patro, Emre Sefer, Hao Wang, Darya Filippova, Samir Khuller, and Carl Kingsford. Resolving spatial inconsistencies in chromosome conformation measurements. *Algorithms for Molecular Biology*, 8:8, 2013.

- 14 Geet Duggal, Hao Wang, and Carl Kingsford. Higher-order chromatin domains link eQTLs with the expression of far-away genes. *Nucleic Acids Research*, 42(1):87–96, 2014.
- 15 Kevin Emmett, Benjamin Schweinhart, and Raul Rabadan. Multiscale topology of chromatin folding. *arXiv*, 2015. [arXiv:1511.01426](https://arxiv.org/abs/1511.01426).
- 16 Paul A Fields, Vijay Ramani, Giancarlo Bonora, Galip Gurkan Yardimci, Alessandro Bertero, Hans Reinecke, Lil Pabon, William S Noble, Jay Shendure, and Charles Murry. Dynamic reorganization of nuclear architecture during human cardiogenesis. *bioRxiv*, page 222877, 2017.
- 17 Mattia Forcato, Chiara Nicoletti, Koustav Pal, Carmen Maria Livi, Francesco Ferrari, and Silvio Bicciato. Comparison of computational methods for Hi-C data analysis. *Nature Methods*, 14(7):679–685, 2017.
- 18 Timothy SC Hinks, Tom Brown, Laurie CK Lau, Hitasha Rupani, Clair Barber, Scott Elliott, Jon A Ward, Junya Ono, Shoichiro Ohta, Kenji Izuhara, et al. Multidimensional endotyping in patients with severe asthma reveals inflammatory heterogeneity in matrix metalloproteinases and chitinase 3-like protein 1. *Journal of Allergy and Clinical Immunology*, 138(1):61–75, 2016.
- 19 Maxim Imakaev, Geoffrey Fudenberg, Rachel Patton McCord, Natalia Naumova, Anton Goloborodko, Bryan R Lajoie, Job Dekker, and Leonid A Mirny. Iterative correction of Hi-C data reveals hallmarks of chromosome organization. *Nature Methods*, 9(10):999–1003, 2012.
- 20 Li Li, Wei-Yi Cheng, Benjamin S Glicksberg, Omri Gottesman, Ronald Tamler, Rong Chen, Erwin P Bottinger, and Joel T Dudley. Identification of type 2 diabetes subgroups through topological analysis of patient similarity. *Science Translational Medicine*, 7(311):311ra174, 2015.
- 21 Erez Lieberman-Aiden, Nynke L van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragozy, Agnes Telling, Ido Amit, Bryan R Lajoie, Peter J Sabo, Michael O Dorschner, Richard Sandstrom, Bradley Bernstein, M A Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A Mirny, Eric S Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 2009.
- 22 Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The Gudhi library: Simplicial complexes and persistent homology. In *International Congress on Mathematical Software*, pages 167–174. Springer, 2014.
- 23 Anindya Moitra, Nicholas O Malott, and Philip A Wilsey. Cluster-based Data Reduction for Persistent Homology. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 327–334. IEEE, 2018.
- 24 Monica Nicolau, Arnold J Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, pages 7265–7270, 2011.
- 25 Benjamin D Pope, Tyrone Ryba, Vishnu Dileep, Feng Yue, Weisheng Wu, Olgert Denas, Daniel L Vera, Yanli Wang, R Scott Hansen, Theresa K Canfield, et al. Topologically associating domains are stable units of replication-timing regulation. *Nature*, 515(7527):402–405, 2014.
- 26 Suhas S P Rao, Miriam H Huntley, Neva C Durand, Elena K Stamenova, Ivan D Bochkov, James T Robinson, Adrian L Sanborn, Ido Machol, Arina D Omer, Eric S Lander, and Erez Lieberman Aiden. A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1880, December 2014.
- 27 Sarah Rennie, Maria Dalby, Lucas van Duin, and Robin Andersson. Transcriptional decomposition reveals active chromatin architectures and cell specific regulatory interactions. *Nature Communications*, 9(1):487, 2018.
- 28 Abbas H Rizvi, Pablo G Camara, Elena K Kandror, Thomas J Roberts, Ira Schieren, Tom Maniatis, and Raul Rabadan. Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, 35(6):551, 2017.

- 29 Nicolas Servant, Nelle Varoquaux, Bryan R Lajoie, Eric Viara, Chong-Jian Chen, Jean-Philippe Vert, Edith Heard, Job Dekker, and Emmanuel Barillot. HiC-Pro: an optimized and flexible pipeline for Hi-C data processing. *Genome Biology*, 16(1):259, 2015.
- 30 Edwin H Spanier. Algebraic topology. *MacGraw-Hill, New York*, 1966.
- 31 Larry Wasserman. Topological data analysis. *Annual Review of Statistics and Its Application*, 5:501–532, 2018.
- 32 Y William Yu, Noah M Daniels, David Christian Danko, and Bonnie Berger. Entropy-scaling search of massive biological data. *Cell Systems*, 1(2):130–140, 2015.

A Proof of correctness of the loop trace-back algorithm

In this section, we will prove that the loop found by shortest path based algorithm is in fact the optimal loop of a particular persistent structure.

As described in Methods, the persistent homology algorithm involves building a series of VR complexes by varying the parameter α , creating different homology classes that appear and disappear at different values of α . For a given parameter α_0 , we construct a weighted graph $(G(\alpha_0), V, E)$ where vertices are all points in the given data set, and an edge, with weight equal to the distance between the vertices, exists if the weight is not larger than α_0 . From the perspective of algebraic topology, the graph $G(\alpha_0)$ can be regarded as a simplicial complex, therefore, definitions such as homology group H_1 can be used on $G(\alpha_0)$ as well. We will also need to define the group of 1-boundaries of a simplicial complex G , $B_1(G)$. Conceptually $B_1(G)$ is the set of all “trivial” cycles in G ; the cycles composed of boundaries of triangles.

The following describes useful notations for the proof.

► **Definition (Cycle weight).** *Given a weighted graph (G, V, E) , $\forall e \in E$, denote $w(e)$ as the weight of e , then the weight of a cycle l in the graph is defined as: $w(l) = \sum_{e \in l} w(e)$.*

► **Definition (Cycle birth time).** *Given a weighted graph (G, V, E) , the birth time of a cycle in the graph is defined as: $T(l) = \max_{e \in l} w(e)$.*

We will refer to an element of a homology group $H_1(G)$ as a homology class, which is an equivalence class over cycles. We define the birth time of a homology class as follows.

► **Definition (Homology class birth time).** *Given a homology group $H_1(G)$, the birth time of a homology class $h \in H_1(G)$ is defined as: $T(h) = \min_{l \in h} T(l)$, where $l \in h$ are cycles.*

We denote $[l] = \{l + s \mid s \in B_1(G)\}$ as the corresponding homology class of the cycle l . The sum of two cycles used here is similar to the sum of a vector space over field \mathbf{Z}_2 , specifically, given two cycles $l_1 = \{e_{11}, \dots, e_{1m}\}$ and $l_2 = \{e_{21}, \dots, e_{2n}\}$, their sum is $l_1 + l_2 = \{e'_1, \dots, e'_j\}$, $e'_i \in ((\{e_{11}, \dots, e_{1m}\} \cup \{e_{21}, \dots, e_{2n}\}) - (\{e_{11}, \dots, e_{1m}\} \cap \{e_{21}, \dots, e_{2n}\}))$. Given a homology class $h \in H_1(G)$, then $\forall l \in h$, $h = [l]$. For the purposes of our algorithm, we assume that at each time α_0 at which the dimension of H_1 increases through the filtration, there is one unique edge e with $w(e) = \alpha_0$. Under this assumption, we look for the optimal cycle L satisfying the following:

$$\begin{aligned}
 L &= \operatorname{argmin} w(l) = \sum_{e_i \in l} w(e_i) \\
 &\text{s.t.} \\
 &l \notin B_1(G(\alpha_0)) \\
 &T([l]) = \alpha_0
 \end{aligned}$$

Intuitively, L is the smallest (in terms of total weight) non-trivial cycle among all the cycles in homology classes born at α_0 .

We will denote the unique edge with weight α_0 as $[a, b]$, and let p_{short} be the shortest path through $G(\alpha_0) \setminus [a, b]$ from a to b . This leads to the main result.

► **Theorem 1.** *If there is only one edge with weight α_0 for α_0 at which the dimension of $H_1(G)$ increases by 1, then the $p_{short} + [a, b]$ is the optimal cycle L .*

The proof of the theorem can be divided into two parts: first we will show that any cycle satisfying two conditions above should contain the edge $[a, b]$, and second we will show that the cycle $p_{short} + [a, b]$ satisfies these two conditions. Since $p_{short} + [a, b]$ is the shortest cycle among all the cycles having the edge $[a, b]$, we can easily get theorem 1.

We have the following three lemmas:

► **Lemma 1.** *If all the edges have unique weights, for any α_0 when the dimension of $H_1(G(\alpha_0))$ increases by one, any cycle l on $G(\alpha_0)$ satisfying the two conditions $l \notin B_1(G(\alpha_0))$ and $T([l]) = \alpha_0$ contains the edge $[a, b]$, where $w([a, b]) = \alpha_0$.*

Proof. Assume for contradiction that there exists a cycle $l = \{e_1, e_2, \dots, e_p\}$ on $G(\alpha_0)$, $l \notin B_1(G(\alpha_0))$ and $T([l]) = \alpha_0$, which does not include the edge $[a, b]$. We assume there is only one edge with weight α_0 , so $w(e_i) < \alpha_0$ for all $e_i \in \{e_1, \dots, e_p\}$. Then $\alpha_0 = T([l]) \leq T(l) = \max_{e \in \{e_1, \dots, e_p\}} w(e) < \alpha_0$, which contradicts the first statement. ◀

► **Lemma 2.** *If all the edges have unique weights, for any α_0 when the dimension of $H_1(G(\alpha_0))$ increases by one, any cycle l which belongs to a homology class h satisfying the conditions $h \in H_1(G(\alpha_0))$ and $T(h) = \alpha_0$ contains the edge $[a, b]$ with $w([a, b]) = \alpha_0$.*

Proof. Assume for contradiction that there exists a cycle $l \in h$, $l = \{e_1, e_2, \dots, e_p\}$, that does not include $[a, b]$, and $h \in H_1(\alpha_0)$ and $T(h) = \alpha_0$. There is only one edge with the weight α_0 , so $w(e_i) < \alpha_0 \forall e_i \in \{e_1, \dots, e_p\}$. Then $\alpha_0 = T(h) \leq T(l) = \max_{e \in \{e_1, \dots, e_p\}} w(e) < \alpha_0$, which is a clear contradiction. ◀

► **Lemma 3.** *If all the edges have the unique weights, for any α_0 when the dimension number of $H_1(G(\alpha_0))$ increases by one, any cycle l on $G(\alpha_0)$ which contains the edge $[a, b]$, $w([a, b]) = \alpha_0$, does not belong to $B_1(G(\alpha_0))$.*

Proof. Assume $\exists l = \{e_1, e_2, \dots, e_p\} + [a, b]$, $e_i \in E$, such that $l \in B_1(G(\alpha_0))$. Since the dimension increases by one at α_0 , $\exists h \in H_1(G(\alpha_0))$ such that $T(h) = \alpha_0$. Choose one cycle $l' \in h$, from Lemma 2, we know that l' contains the edge $[a, b]$, then we can write l' as $l' = \{e'_1, e'_2, \dots, e'_q\} + [a, b]$. Since $l \in B_1(G(\alpha_0))$, $l + l' = \{e_1, e_2, \dots, e_p\} + [a, b] + \{e'_1, e'_2, \dots, e'_q\} + [a, b] \subseteq \{e_1, e_2, \dots, e_p, e'_1, e'_2, \dots, e'_q\}$ still belongs to the homology class h . Then $\alpha_0 = T(h) \leq T(l + l') \leq \max_{e \in \{e_1, e_2, \dots, e_p, e'_1, e'_2, \dots, e'_q\}} w(e) < \alpha_0$, which is again a clear contradiction. ◀

Lemma 3 tells us that $p_{short} + [a, b] \notin B_1(G(\alpha_0))$, and also any cycle $l = p_{short} + [a, b] + s$, $s \in B_1(G(\alpha_0))$ contains the edge $[a, b]$, therefore $T(p_{short} + [a, b] + s) \geq \alpha_0$, $\forall s \in B_1(G(\alpha_0))$, then $\alpha_0 \geq T([p_{short} + [a, b]]) = \min_{s \in B_1(G(\alpha_0))} T(p_{short} + [a, b] + s) \geq \alpha_0$ which indicates that $T([p_{short} + [a, b]]) = \alpha_0$.

Now we have proved that the cycle $p_{short} + [a, b]$ satisfies two conditions, and from Lemma 1 we have also proved that any cycle satisfying these two conditions must contain $[a, b]$, therefore $p_{short} + [a, b]$ is definitely the one with the smallest weight. We have finished the proof of theorem 1.

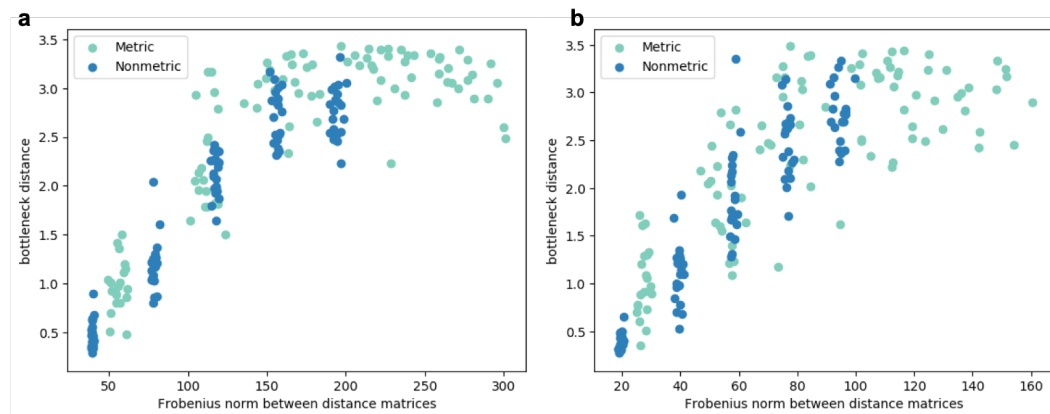


Figure 5 Study of robustness to violating the triangle inequality. In both figures, the “metric” values came from perturbing the original data and generating a distance matrix satisfying the triangle inequality, and the “nonmetric” points are the result of perturbing the distance matrix to violate the triangle inequality. The x-axis is the Frobenius norm between the perturbed distance matrix and the original, and the y-axis represents the bottleneck distance between the resulting persistent homology computations. (a) Underlying geometry: one two-dimensional circle. (b) Underlying geometry: Two adjacent two-dimensional circles.

B Study of robustness to triangle inequality failure

Much of the theory of TDA is built on the assumption that the distances being analyzed satisfy the definition of a metric. Unfortunately Hi-C data is not a direct measurement of distance, and does not satisfy the triangle inequality. The transformation used here to convert Hi-C values to a “distance” similarly does not satisfy the triangle inequality and is therefore not a metric. In order to determine whether the violation of the metric assumption strongly influences the results of TDA, we ran a small study on two toy examples. One example is a simple two-dimensional circle, and the other is two adjacent two-dimensional circles. First, we sampled n points (40 points for the first example and 80 for the other) from each underlying geometry, and generated a distance matrix from these points. We then performed two perturbations of the sampled data: (1) preserving symmetry and non-negativity, we applied Gaussian noise of increasing variance to the distance matrix itself, creating a matrix which violates the triangle inequality, and (2) we applied Gaussian noise of increasing variance to the sampled points, and computed a distance matrix from the perturbed points, thereby preserving the triangle inequality. We ran 20 perturbations for each variance (0.5, 1, 1.5, 2, and 2.5), and then ran TDA on all matrices from the two perturbations as well as the original sampled data, and computed the bottleneck distance between each perturbation and the original data. We note that the bottleneck distances for perturbations that violate the metric definition do not differ clearly from those that obey it, suggesting that in these toy examples, the violations of the triangle inequality did not meaningfully influence the output of the persistent homology computations (Figure 5).

23:16 Topological Data Analysis for Hi-C

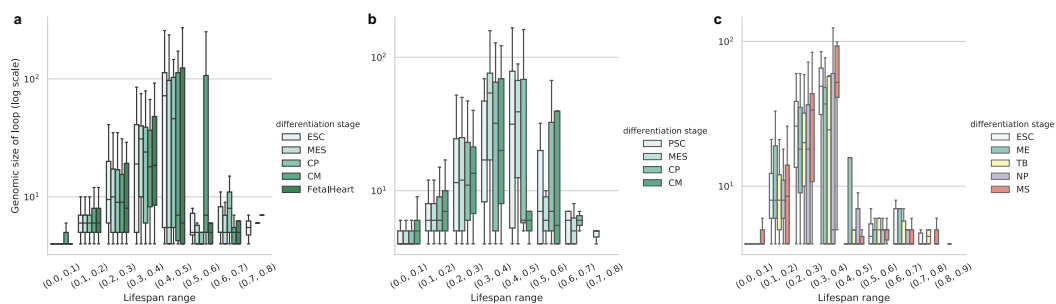


Figure 6 Distributions of genomic loop sizes across loop lifespans, separated by differentiation stage. The log-scaled y-axis again represents the total number of genomic bins returned by the loop traceback method. (a) RUES2 cell line (b) WTC11, and (c) H1. Note that because the H1 data is not a linear trajectory through the differentiation process like the RUES2 and WTC11 data, this is represented by a different color scheme.

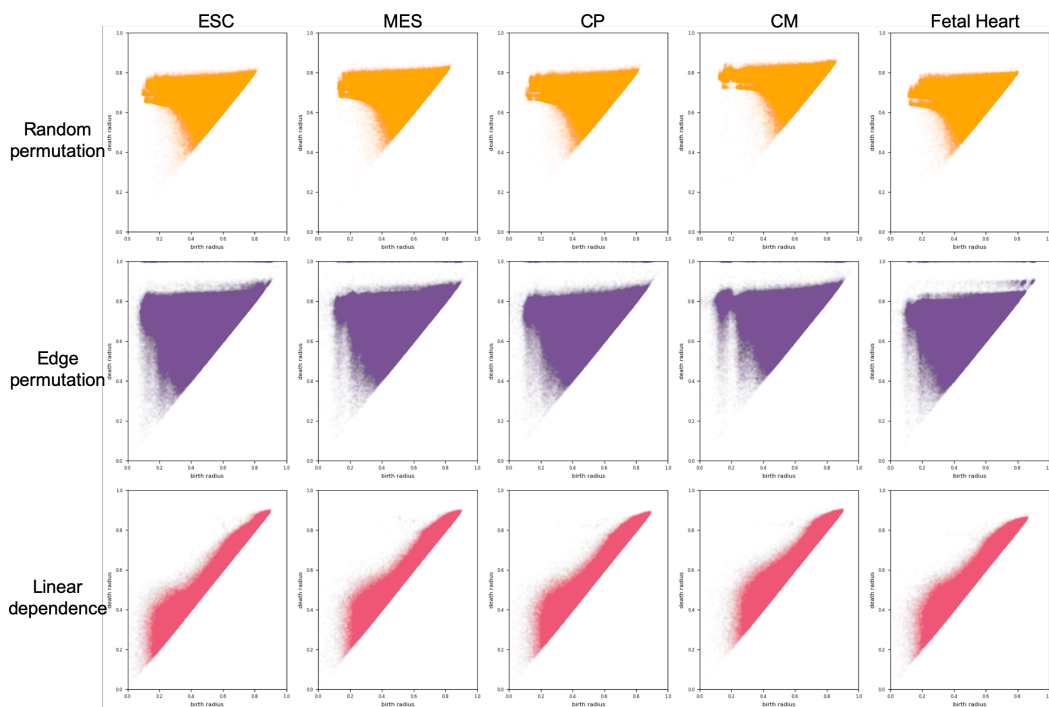



Figure 7 Persistence diagrams for all null models. Ten null models were generated for each of the RUES2 cell types on each chromosome from 13-22. All computed models for each cell type are plotted on top of each other here to demonstrate that each null model shows a very characteristic shape, which is very consistent across all models. Lighter-colored areas represent regions of variance between chromosomes or models.

Synteny Paths for Assembly Graphs Comparison

Evgeny Polevikov

Bioinformatics Institute, Saint Petersburg, Russia
polevikovea@gmail.com

Mikhail Kolmogorov¹ 

Department of Computer Science and Engineering, University of California, San Diego, CA, USA
mkolmogo@ucsd.edu

Abstract

Despite the recent developments of long-read sequencing technologies, it is still difficult to produce complete assemblies of eukaryotic genomes in an automated fashion. Genome assembly software typically output assembled fragments (contigs) along with assembly graphs, that encode all possible layouts of these contigs. Graph representation of the assembled genome can be useful for gene discovery, haplotyping, structural variations analysis and other applications. To facilitate the development of new graph-based approaches, it is important to develop algorithms for comparison and evaluation of assembly graphs produced by different software. In this work, we introduce *synteny paths*: maximal paths of homologous sequence between the compared assembly graphs. We describe Asgan – an algorithm for efficient synteny paths decomposition, and use it to evaluate assembly graphs of various bacterial assemblies produced by different approaches. We then apply Asgan to discover structural variations between the assemblies of 15 *Drosophila* genomes, and show that synteny paths are robust to contig fragmentation. The Asgan tool is freely available at: <https://github.com/epolevikov/Asgan>.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Assembly graphs, Genome assembly, Synteny blocks, Comparative Genomics

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.24

Supplement Material The Asgan tool is freely available at: <https://github.com/epolevikov/Asgan>.

Funding The authors received no specific funding for this work.

Acknowledgements We are grateful to Dmitry Antipov, Pavel Avdeyev, Pavel Pevzner and Giulia Guidi for their helpful comments.

1 Introduction

Genome assembly is the problem of reconstructing a DNA sequence from sequencing reads - short, overlapping substrings of the original DNA. This reconstruction is challenging because of the presence of genomic repeats - multiple copies (precise or imprecise) of the same sequence within the genome. Since the read length is limited, even simple bacterial assemblies from short Illumina reads may contain hundreds of unresolved repeats, which results in fragmented assemblies [22]. The increased read length of the Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) sequencers significantly improved the contiguity of many de novo assemblies [27]. However, other genomes are still incomplete due to very long unresolved repeats, such as segmental duplications in human genomes [34].

¹ Corresponding author



To reduce the assembly fragmentation and optimally resolve repeats, genome assemblers construct various assembly graphs from input reads. One popular representation is the overlap graph [23], where each input read corresponds to a node, and directed edges represent overlaps between the corresponding reads. A limitation of this approach is that it does not reveal precise boundaries of genomic repeats as they might be hidden inside the nodes [10]. Alternatively, de Bruijn graphs [26] proved to be a useful framework that represents read information in a compact form and reveals repeat boundaries [25]. A recently introduced Flye algorithm [12] utilizes repeat graphs, which are similar to de Bruijn graphs, but are built using approximate, rather than exact sequence matches.

A typical genome assembly workflow includes building and simplifying an assembly graph, afterwards contigs are generated as unambiguous paths in this graph [32, 4]. Many studies focus only on the resulting contig fragments for downstream analysis, however it was shown that incorporating the adjacency information from assembly graphs can be useful for gene discovery [33], structural variation analysis [9], hybrid assembly [2, 35], haplotyping [30], segmental duplication analysis [12] and other applications.

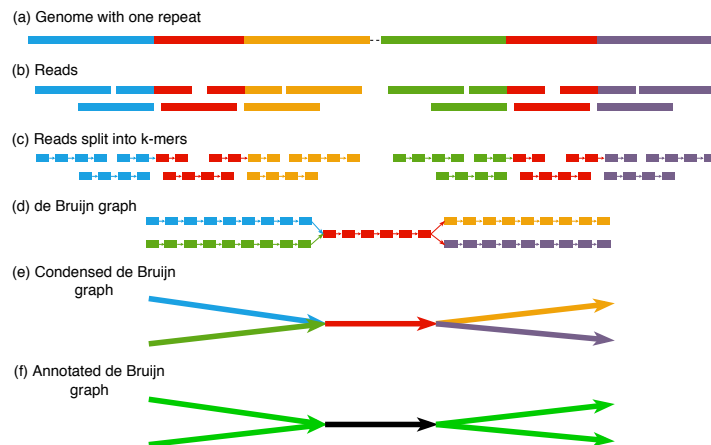
To facilitate the development of new assembly graph-based approaches, it is important to develop algorithms for comparison and evaluation of various assembly graphs produced by different approaches. Recently, a GTED distance metric [5] has been introduced as an attempt to generalize the string edit distance [14] to sequence graphs. While theoretically sound, this approach has only been applied to small viral genomes due to the computational constraints. Additionally, Earth mover's distance (EMD) was proposed as a probabilistic distance measure between de Bruijn graphs of metagenomic assemblies [18]. The authors have shown that incorporating the connectivity information from de Bruijn graphs improves the accuracy of metagenomic sample classification, comparing to the read mapping approaches. However, the described algorithm works only with de Bruijn graphs built with small values of k (less than 10), which makes it unsuitable to analyse the structure of the assembly graphs produced by the most genome assemblers. Finally, various visualization tools, such as Bandage [36] or AGB [19] allow for visual inspection of assembly graphs, but do not support automatic graphs comparison.

In this work, we propose a new method for assembly graphs analysis and comparison, which could be used for benchmarking various assembly algorithms, and for improving comparative genomics capabilities of fragmented assemblies. We introduce *synteny paths* - maximal paths of homologous sequence between the compared assembly graphs, which are inspired by synteny blocks that are commonly used for structural comparison of complete genomes [24, 8]. We formulate the *minimum synteny paths* decomposition problem, prove that its exact solution is NP-hard and provide an efficient heuristic algorithm. Then, we illustrate the application of synteny paths for evaluating assembly graphs of various bacterial genomes produced by different assemblers. Finally, we apply synteny paths to compare 15 *Drosophila* assemblies and show that our approach is robust to contig fragmentation, and reveals structural divergences between the compared genomes.

2 Background

De Bruijn graphs. Given a set of reads R and a parameter k , de Bruijn graph $DBG(R, k)$ is constructed by first representing each read of length L as a set of $L - k + 1$ overlapping k -mers (substrings of length k). Each unique k -mer k_1 from the constructed set is translated into a node v_{k_1} in the de Bruijn graph. Two nodes v_{k_1} and v_{k_2} are connected by a directed edge, if the corresponding k -mers k_1 and k_2 are adjacent in at least one of the input reads

(Figure 1a-d). Note that it also means: (i) k_1 and k_2 overlap by $k - 1$ nucleotides and (ii) edge (v_{k_1}, v_{k_2}) spells a $(k + 1)$ -mer, constructed from two overlapping k -mers. Since each read could also be represented as a sequence of consecutive $(k + 1)$ -mers: (k_1, k_2, \dots, k_n) , the corresponding edges in DBG form a path. Similarly, because every read is a substring of the original genome G , there is a path in DBG , that spells G (assuming no sequencing errors and uniform read coverage).



■ **Figure 1** De Bruijn graph construction and annotation. (a) Genome with one repeat of multiplicity two (shown in red). (b) Reads sampled from the genome. (c) Each read of length L represented as sequences of $L - k + 1$ k -mers. (d) De Bruijn graph is built by gluing k -mers that spell the same sequence. The repeat copies are collapsed into a path, with first and last nodes of this path revealing the repeat boundaries. (e) Condensed de Bruijn graph is constructed by collapsing non-branching paths into single edges. (f) Annotated de Bruijn graph, with unique edges shown in green and a repetitive edge shown in black.

For simplicity, we assume that the genome consists of one circular chromosome, thus every node in DBG has at least one incoming and outgoing edge. Because some k -mers appear multiple times in the genome, the corresponding DBG nodes might have multiple incoming or outgoing edges. We call a node *non-branching*, if it has one incoming and one outgoing edge (and *branching* otherwise). A *maximal non-branching path* is a path in which two terminal nodes are branching, and all intermediate nodes are non-branching. Many assemblers simplify the described de Bruijn graph by collapsing every maximal non-branching path into a single edge, labelled with a sequence spelled by the original path. This transformation results in a condensed de Bruijn graph (Figure 1e). Further in text, we assume that de Bruijn graphs are condensed, meaning that edge sequences might be longer than k .

Double-stranded de Bruijn graphs. De novo assembly algorithms assume that reads might originate from either forward or reverse DNA strand, therefore it is convenient to represent both strands of the assembled genome in a double-stranded de Bruijn graph. For any edge e that spells sequence S in such graph, there exists a single complementary edge e' that spells S' (reverse-complement of S). Further, a double-stranded DBG is symmetric with respect to the complement operation: for any path $P = (e_1, e_2, e_3, \dots, e_{l-1}, e_l)$ of size l that spells sequence S , there exists a complementary path $P' = (e'_l, e'_{l-1}, \dots, e'_3, e'_2, e'_1)$ of size l that spells S' .

3 Methods

Annotated de Bruijn graph (ADBG). If all k -mers in a genome G were unique, $DBG(G)$ would consist of a single non-branching cyclic path, however repetitive k -mers complicate the graph structure. If a k -mer originates from a genomic repeat (of length $\geq k$), all copies of this k -mer in the genome will be collapsed into a single node on DBG . Similarly, multiple consecutive repetitive k -mers from the genome will be collapsed into a non-branching path in the graph (or a single edge in case of the condensed DBG).

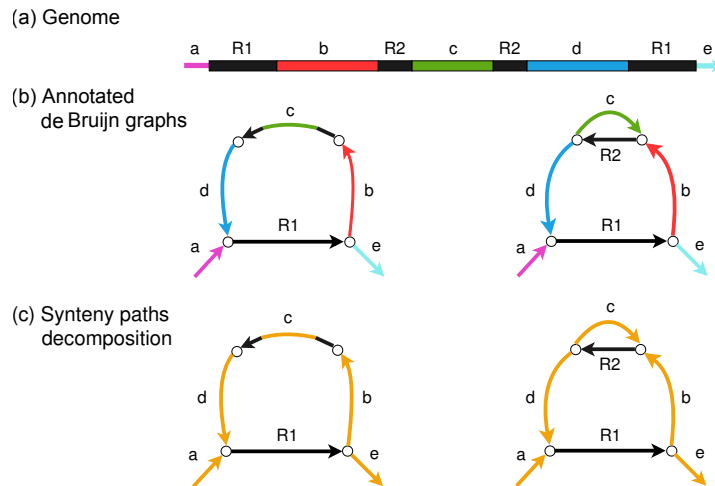
We define *annotated de Bruijn graph (ADBG)* as a de Bruijn graph with each edge labeled as either unique or repetitive. If the complete genome sequence G is available, one can classify each edge of $ADBG(G)$ as either unique or repetitive by checking how many times the corresponding sequence appears in G . If the genome sequence is unknown (as in de novo assembly), edges could be classified based on read coverage, edge length distribution and other criteria (we provide an implementation below).

Analyzing different ADBGs derived from the same genome. First, we consider a case when two ADBGs AG_1 and AG_2 are constructed from the same genome G , but with different values of k ($k_1 < k_2$). Assuming uniform read coverage and no sequencing errors, both AG_1 and AG_2 contain a path that spells G . However, AG_1 might have more repetitive edges than AG_2 corresponding to repeats varying in length from k_1 to k_2 . To reveal similarities between two graphs, we focus on their unique edges, because mapping of repetitive edges could be ambiguous. For each unique edge e^1 in AG_1 , there is a unique edge e^2 in AG_2 that has a substring that is identical to the string spelled by e^1 . We call this pair of edges e^1, e^2 *syntenic*. In the case of condensed DBG , a unique edge e^2 from AG_2 might correspond to multiple unique edges $\{e_1^1, e_2^1, \dots, e_k^1\}$ from AG_1 due to higher fragmentation level. In this case, we split e^2 into a path of syntenic edges $(e_1^2, e_2^2, \dots, e_k^2)$ that spells the original sequence of e^2 . This defines unique syntenic edge pairs between AG_1 and AG_2 . (Figure 2a-b).

Synteny paths. We say that two unique edges u and v are *compatible* in an ADBG AG if either (i) u and v are adjacent or (ii) AG contains a path between u and v such that all intermediate edges in this path are repetitive. Intuitively, we allow arbitrary insertions of repetitive sequences between the compatible unique edges, but prohibit unique sequence to be rearranged. Given two ADBGs AG_1 and AG_2 with two pairs of syntenic edges $U = \{u^1, u^2\}$ and $V = \{v^1, v^2\}$, we call pairs U and V *colinear* if u^1 and v^1 are compatible in AG_1 , and u^2 and v^2 are compatible in AG_2 . Synteny path is defined as a sequence of syntenic edge pairs $P = (E_1, E_2, \dots, E_k)$, in which every two consecutive pairs (E_{i-1}, E_i) are colinear (Figure 2c). A single syntenic edge pair is also considered a trivial synteny path.

Minimum synteny paths (MSP) decomposition. Each synteny path reveals local similarities between edges of ADBGs. To compare two graphs globally, we formulate the following decomposition problem. Given two ADBGs AG_1 and AG_2 with unique edges decomposed into the set of syntenic edge pairs $E = \{(u^1, u^2), (v^1, v^2), (w^1, w^2), \dots\}$, find the minimum number of synteny paths that cover all edge pairs from E . Intuitively, we want to find a minimal set of sequences that traverse all unique edges in AG_1 and AG_2 in the same order, while allowing arbitrary repeat insertions.

If AG_1 and AG_2 are built from the same unichromosomal genome G , they share a synteny path that spells G (with repeats removed) and solves MSP. However, because multiple MSP solutions may exist, a single path that solves MSP might not correspond to the original genome. Because each syntenic edge pair is considered a trivial synteny path, the maximum number of synteny paths in MSP equals to the number of syntenic edge pairs.

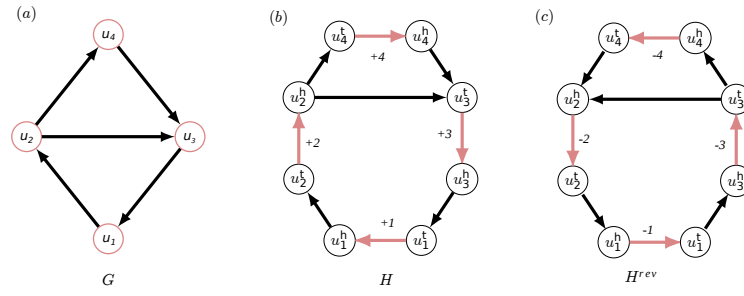


■ **Figure 2** An example of synteny paths decomposition. (a) A genome with two repeats of lengths $|R_1| > |R_2|$ shown in black. Unique sequences between repeats is shown in color. Each repeat is represented in two exact copies. (b) Two ADBGs built with different values of k . The left graph is built with $|R_2| < k < |R_1|$, so that only R_1 is collapsed in the graph. The graph on the right is built for $k < |R_2| < |R_1|$, thus both R_1 and R_2 are collapsed in the graph. Colors denote edge pairs that spell the same genome subsequences (syntenic edges). (c) Syntenic edges are joined into a single synteny path (shown in orange). The path traverses unique edges in the same order (a, b, c, d, e) and reveals the similarity between two graphs.

In case of double-stranded ADBG, we require the resulting set of synteny paths to be symmetric with respect to the complement operation: for any path P from the MSP solution, P' also belongs to the solution. Further in text we assume the symmetric version of the MSP decomposition.

MSP decomposition is NP-hard. Following we prove that the MSP decomposition is NP-hard by reducing the Hamiltonian path problem in directed graphs to the MSP decomposition. Let G be an arbitrary directed graph. We construct a new ADBG H from G , by translating nodes from G into a unique edges in H , and edges from G into repetitive edges in H . For each node u from G , we create two nodes in H : u^t and u^h (*tail* and *head* nodes, respectively), which are connected with a unique (directed) edge (u^t, u^h) . Next, for each edge (u, v) from G , we connect the nodes u^h and v^t of H with a repetitive edge (u^h, v^t) . Note that unique edges in H always start at tail nodes and end at head nodes, while repetitive edges start at head nodes and end at tail nodes (Figure 3a-b). Afterwards, we build a complementary graph H^{rev} as a copy of H with all edges reversed. Each unique edge (u_i^t, u_i^h) from H and its complement (u_i^h, u_i^t) from H^{rev} are labelled as $+i$ and $-i$, respectively. The described transformation could be performed in $O(|V| + |E|)$ time. For simplicity, we first prove that single-stranded MSP version is NP-hard using H , and then extend it to the double-stranded case with $H_{ds} = H \cup H^{rev}$ (Figure 3c). We also assume that indegree and outdegree of nodes in H is unlimited – below we show how to extend our proof to ADBGs over the DNA alphabet, in which indegree and outdegree of each node is at most four.

► **Lemma 1.** *If there is a Hamiltonian path in the initial graph G , it corresponds to a path in H that covers all unique edges.*



■ **Figure 3** Reducing the Hamiltonian path problem to the MSP problem. (a-b) Each node in the initial graph G is translated to a unique edge (shown in pink) in the new graph H , while the edges of G are translated into repetitive edges (shown in black) in H . (c) A complementary graph H^{rev} is constructed by inverting the directions of all edges in H . A union of two graphs $H_{ds} = H \cup H^{rev}$ is then used to solve the double-stranded MSP decomposition.

Proof. Let $p_G = (u_1, u_2, \dots, u_n)$ be a Hamiltonian path in G . By construction, it corresponds to the following path in H : $p_H = (u_1^t, u_1^h, u_2^t, u_2^h, \dots, u_n^t, u_n^h)$. Since each node u_i of G is covered by p_G , each unique edge (u_i^t, u_i^h) of H is covered by p_H . ◀

► **Lemma 2.** *If there is a path in H that covers all unique edges, it corresponds to a Hamiltonian path in G .*

Proof. Let $p_H = (u_1^t, u_1^h, u_2^t, u_2^h, \dots, u_n^t, u_n^h)$ be a path of length n in H that covers all unique edges (u_i^t, u_i^h) . By construction, consecutive pairs of nodes (u_i^h, u_{i+1}^t) correspond to repetitive edges in H . Thus, p_H is an alternating sequence of unique and repetitive edges in H , and is translated to the following node path of length n in G : $p_G = (u_1, u_2, \dots, u_n)$, which traverses all nodes in G . ◀

► **Theorem 3.** *Minimum synteny paths decomposition is NP-hard.*

Proof. Note that Lemmas 1-2 were formulated for paths in a single graph H . The lemmas could be trivially extended to pairwise synteny paths by duplicating H into H^{copy} and defining the duplicated unique edges as syntenic. Likewise, we can trivially extend the single-stranded version of the MSP decomposition to the double-stranded case by considering $H_{ds} = H \cup H^{rev}$, since H and H^{rev} are symmetric and independent.

Finally, to overcome the maximum node degree limit in ADBGs constructed over the DNA alphabet, we apply the following transformation to H . Each node with indegree or outdegree more than four is split into multiple nodes, and the original edges are distributed among the new nodes so as to satisfy the degree limit. The new nodes are also connected via additional repetitive edges in both directions. This forms *supernodes*, which are equivalent to the original nodes in H with respect to the connectivity of the unique edges.

The extended versions of Lemmas 1-2 prove the theorem. ◀

A heuristic algorithm for the MSP decomposition. Since the exact solution for the MSP decomposition is NP-hard, we propose an efficient heuristic algorithm. Given two double-stranded ADBGs AG_1 and AG_2 with unique edges decomposed into $2n$ syntenic pairs $\{\pm 1, \pm 2, \dots, \pm n\}$ (complementary edges have opposite sign), we construct a breakpoint graph [24, 3] as described below.

For each syntenic pair $+i$, we create two nodes in the breakpoint graph: i^t and i^h . A complementary syntenic pair $-i$ corresponds to the same pair of nodes, but in reversed order: i^h and i^t . Then, for each pair of colinear syntenic edges (u, v) , we put an undirected

edge between u^h and v^t (denoted as *adjacency* edge). Note that the complementary colinear syntenic pair $(-v, -u)$ will correspond to the same connection on the breakpoint graph, hence providing a convenient way to represent two strands of a de Bruijn graph [16] (Figure 4a-b).

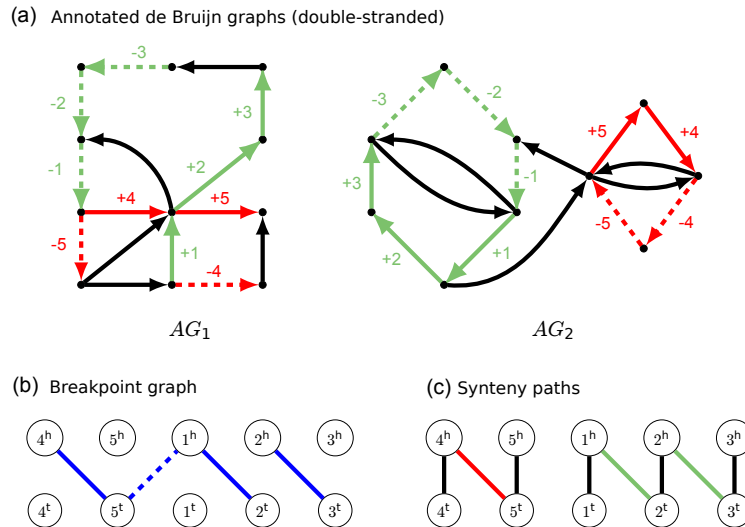


Figure 4 A heuristic algorithm for minimum syntenic paths problem. (a) Two double-stranded ADBGs AG_1 and AG_2 with edges decomposed into syntenic pairs. Repetitive edges shown in black, and unique edges are colored into green and red, highlighting two syntenic paths in the graphs. Dashed edges represent complementary edges. (b) Breakpoint graph constructed from AG_1 and AG_2 . The nodes correspond to the ends of syntenic blocks (head or tail) and edges represent the corresponding colinear blocks. Solid edges show the selected maximum matching (dashed edge was not selected). (c) Syntenic paths are revealed by removing edges that are not part of the matching, and adding trivial edges (shown in black).

The resulting breakpoint graph encodes all possible colinear syntenic pairs, therefore a matching on this graph defines a set of syntenic paths. To minimize the number of syntenic paths, we find a maximum matching on the breakpoint graph using the Blossom algorithm [7] (Figure 4b). Given the maximum matching, we reconstruct syntenic paths from the breakpoint graph by adding edges that connect nodes $\{i^h, i^t\}, i = 1, \dots, n$ (trivial edges). The resulting set of paths on the breakpoint graph defines a set of syntenic paths between AG_1 and AG_2 (Figure 4c). Note that because the complementary colinear connections are represented by the same adjacency edges in the breakpoint graph, the constructed MSP solution will automatically be symmetric.

In some cases, the reconstructed matching might not yield to the optimal MSP solution. Consider an initial breakpoint graph with only trivial edges present. When a new adjacency edge is added into the graph it either (i) connects two paths into one (reducing the number of syntenic paths by one), or (ii) transforms an existing path into a cycle (the number of paths is not reduced). After the initial maximal matching is reconstructed, we modify it using the following heuristic to minimize the number of edges of the second type and improve the MSP solution. The algorithm finds pairs of cycles in the graph that could be merged by exchanging two adjacency edges with two another adjacency edges (similarly to the 2-break operation [3]). Such pairs of cycles are then merged in a greedy manner. In practice, the described heuristic was not triggered for the most of the real datasets we describe below.

Analyzing ADBGs of two closely related genomes. For the sake of simplicity, we previously were assuming that de Bruijn graphs are built from the same genome using different values of k . Here we extend the proposed algorithm to the comparison of two closely related genomes. This comparison is more challenging as the genomes might contain small mutations or structural variations. Comparative genomics studies typically decompose genomes into sets of coarse synteny blocks to mask small-scale sequence polymorphisms [24, 8]. We apply a similar principle to decompose the assembly graph edges into syntenic segments.

Given two ADBGs AG_1 and AG_2 , we perform local pairwise alignment of all unique edges from AG_1 to all unique edges of AG_2 using minimap2 [15]. We discard alignments that are shorter than 50 Kbp to focus on large-scale structure. Afterwards, colinear alignments (that appear in the same order and orientation in both genomes) are chained into synteny blocks [11]. Similarly to the described above, if an edge contains multiple synteny blocks, we split this edge into a path of new edges, each corresponding to a single synteny block. Since each unique edge now contains one synteny block, this defines the syntenic edge pairs. It is possible that some unique edges might not be aligned because the corresponding sequence is missing in the other genome. Such edges are flagged as *inserted*, and are logically equivalent to repetitive edges in the MSP decomposition. To be robust to possible chimeric connections in the graphs, we also require nucleotide distance between the chained alignments as well as the distance between the colinear synteny blocks to be less than 1 Mb.

Annotated repeat graphs. Recently, it has been shown how to apply repeat graphs for long-read assembly [12]. Similarly to de Bruijn graphs, repeat graphs reveal the repeat structure of the genome, which makes them suitable for the synteny paths analysis. Instead of relying on exact k -mer matches, repeat graphs are built from local sequence alignments and thus are more robust to high error rate of long reads. By design, repeat graphs can hide small sequence polymorphisms, and are well suited for the analysis of large structural variations.

The described algorithms have been implemented into a tool named Asgan (**A**ssembly **G**raphs **A**nalyzer). Asgan takes two annotated de Bruijn or repeat graphs as input (in the GFA format), and outputs synteny paths visualized using Graphviz along with various statistics. The Flye assembler [12] produces annotated repeat graphs as a part of its output. For assemblers based on the overlap graph approach, such as Canu [13], we construct repeat graphs by running the Flye graph construction algorithm on the assembled contigs (that contain flanking repeat sequences). The implementation is freely available at: <https://github.com/epolevikov/Asgan>.

4 Results

Comparing Flye and Canu assembly graphs using bacterial datasets from the NCTC collection. First, we illustrate the application of synteny paths for comparison of graphs produced by different assembly methods. We focus on the assembly graphs reconstructed from long-read sequencing data because they are typically less tangled and easier to visualize, comparing to short Illumina assemblies. We used two assemblers, Canu [13] and Flye [12] for the comparison, because they represent two different approaches for repeat resolution. We did not attempt to evaluate the GTED and EMD implementations since they were not designed for overlap / repeat graphs input (as discussed above).

We used Flye and Canu to assemble 21 bacterial genomes from the National Collection of Type Cultures (<https://www.sanger.ac.uk/resources/downloads/bacteria/nctc/>). When running Flye, we turned off the Trestle module (that resolves extra unbridged repeats) to make Flye graphs potentially more tangled. Each dataset contains P5C3 PacBio reads

■ **Table 1** Comparison of the assembly graphs produced by Flye and Canu using 21 bacterial genomes from the NCTC collection. Edges shorter than 50 Kb were ignored. Only one orientation (forward or reverse) of each edge and connected components was counted. Fourteen concordant datasets (where each connected component is covered by a single syntenic path) are marked with the * symbol.

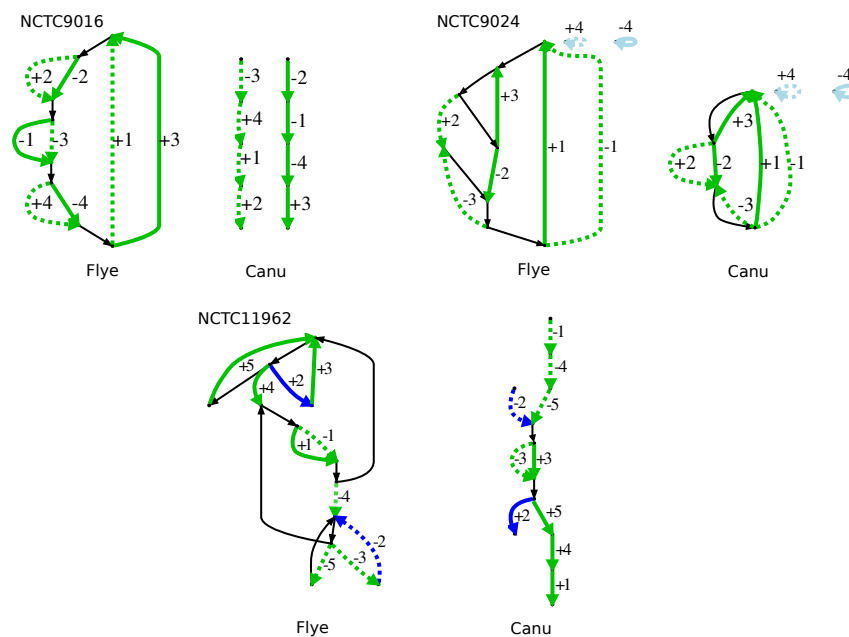
ID	Flye		Canu		Flye & Canu	
	Unique edges	Connected components	Unique edges	Connected components	Unique edges	Connected components
4450	7	1	8	6	12	8
6134	1	1	2	2	3	2
8333	8	1	3	1	8	2
8781	10	1	4	1	11	2
9657	6	2	6	6	9	7
11692	6	1	4	3	6	3
11962	4	1	3	1	5	2
5052*	7	2	5	2	7	2
7921*	3	1	1	1	4	1
9002*	1	1	1	1	2	1
9006*	7	1	2	1	7	1
9007*	2	1	2	1	2	1
9012*	4	1	2	1	4	1
9016*	4	1	1	1	4	1
9024*	4	2	4	2	4	2
9103*	3	3	3	3	3	3
9964*	4	1	1	1	4	1
10864*	6	1	1	1	7	1
11022*	4	1	1	1	4	1
12158*	3	2	2	2	3	2
12993*	2	2	2	2	2	2

with coverage varying from $70\times$ to $271\times$. For each assembly, the number of unique edges (longer than 50 Kb) ranged from 1 to 11 for Flye, and 1 to 8 for Canu. Since the assembly graphs were double-stranded, we only counted one orientation (forward or reverse) of each edge. Similarly, we counted only one orientation of each connected component, which might represent a bacterial chromosome, a plasmid, or a mixture of them.

The results are shown in the Table 1. We call two assembly graphs *concordant*, if each connected component in both graphs is covered by a single syntenic path. Fourteen out of 21 datasets were concordant, which is expected for assemblies generated from the same set of reads. In 8 out of 14 concordant datasets, Flye had more unique edges than Canu. This suggests that in these 8 datasets Canu resolved more repeats than Flye (as expected due to not using Trestle). Figure 5 shows examples of concordant and discordant assembly graphs.

In five out of seven datasets that were not concordant, Canu graphs, but not Flye graphs contained “dead end” edges (disconnected from the rest of the graph from either beginning or end). These missing connections might have contributed to the increased number of syntenic paths. In two remaining cases, both assemblers produced tangled graphs with complex unresolved repeats.

Asgan took less than a minute of wall clock time and less than 100 Mb of RAM to process each bacterial dataset. The running time was dominated by the minimap2 alignments (using three threads).



■ **Figure 5** (Top left) Comparison of bacterial assembly graphs produced by Flye and Canu for the NCTC9016 dataset. Synteny paths are shown in green. Complement paths are dashed. In the Canu graph, two strands of one connected component are separated, while in the Flye graph both strands are merged into one component through the unresolved repeats. Although the assembly produced by Flye is more fragmented, a single synteny path $(-2, -1, -4, +3)$ reveals the structural similarity. (Top right) Synteny paths decomposition for the NCTC9024 dataset. Both assemblies resulted into tangled repeat graphs with different number of unresolved repeats. Two synteny paths cover each connected component in full: $(+1, +3, -2)$ and $(+4)$. (Bottom) Two synteny paths (green and blue) in the NCTC11962 dataset reveal structural inconsistencies between the Flye and Canu graphs.

Comparing assembly graphs of 15 *Drosophila* genomes. Synteny paths decomposition could also be applied to structurally compare assemblies of different genomes. We used Flye to assemble 15 different *Drosophila* species from ONT data [20]. Read coverage was varying from 23x to 44x, read length N50 was varying from 7 Kb to 28 Kb. Flye produced contiguous assemblies with N50 over 1Mb for 14 out of 15 datasets (Table 2).

First, we applied synteny paths to compare each assembly against the high-quality *Drosophila melanogaster* reference genome. Table 2 shows various assembly statistics, as well as the number of synteny blocks and synteny paths for each genome. The most distant species exhibit more than 10% nucleotide divergence, and pairwise alignments were problematic to compute. Instead, we estimated the evolutionary divergence of each genome from *D. melanogaster* as the number of matched 15-mers within the synteny blocks (reported by minimap2). K -mer survival rate could be estimated from point mutation rate d as: $(1 - d)^k$. For example, 0.87 k -mer match rate of the *D. melanogaster* assembly corresponds to approximately 1% base difference, which is typical for the current ONT assemblies [12]. Synteny block coverage (total length of all synteny blocks divided by the assembly size) was varying from 94% for the *D. melanogaster* assembly to low 2% for the *D. virilis* assembly, which highlights the challenge of recovering synteny blocks between diverged genomes.

As expected, we observed an increase in the number of synteny blocks, as the distance between an assembly and the reference increases, excluding the three most distant genome which had reduced the number of blocks due to the difficulties in alignment (Table 2). On average, the number of synteny paths was 18% smaller than the number of synteny blocks within each genome (varying from 0% to 56%). Under the assumption that the breakpoint

reuse rate is low, the length of large synteny blocks should be exponentially distributed [24]. Thus, N50 contiguity metric could be used as a complement to the number of synteny blocks or paths (defined as the largest possible number L , such that all blocks/paths of length L or longer cover at least 50% of total blocks/paths length). Table 2 shows that synteny paths N50 correlates with the evolutionary distance between the compared genomes and also robust to the decreased synteny blocks coverage.

■ **Table 2** *Drosophila* assembly graphs statistics and comparison against the *D. melanogaster* reference. Genomes are sorted according to the k-mer identity, computed as the number of matching 15-mers against the *D. melanogaster* reference. Synteny blocks coverage was calculated as the total length of the blocks divided by the total assembly length. Only the sequences that are contained in a component with at least one synteny block were considered.

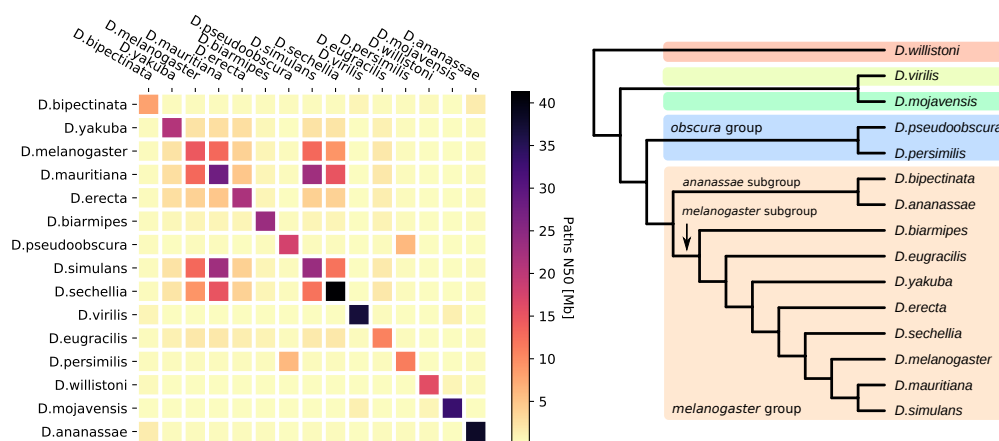
Genome	Asm. size, Mb	Asm. N50, Mb	Blocks, No.	Paths, No.	Blocks N50, Mb	Paths N50, Mb	Blocks cov.	K-mer identity
<i>D. melanogaster</i>	137.0	11.7	78	39	11.7	14.8	94%	0.87
<i>D. mauritiana</i>	128.1	13.9	32	16	13.5	20.8	88%	0.50
<i>D. simulans</i>	124.8	21.3	16	13	20.9	20.9	90%	0.48
<i>D. sechellia</i>	139.6	20.0	30	27	12.1	12.1	81%	0.47
<i>D. yakuba</i>	146.1	16.5	77	72	2.4	2.4	77%	0.30
<i>D. erecta</i>	131.8	12.4	46	39	4.3	4.8	86%	0.30
<i>D. eugracilis</i>	159.5	2.1	159	103	1.3	2.0	69%	0.19
<i>D. biarmipes</i>	168.8	4.9	194	181	0.8	0.9	64%	0.15
<i>D. ananassae</i>	176.8	3.1	286	281	0.4	0.42	40%	0.09
<i>D. bipectinata</i>	179.2	0.8	317	266	0.28	0.39	35%	0.09
<i>D. pseudoobscura</i>	151.8	2.7	238	230	0.24	0.28	28%	0.07
<i>D. persimilis</i>	156.4	2.3	237	229	0.26	0.28	26%	0.07
<i>D. willistoni</i>	187.1	2.9	27	26	0.16	0.18	2%	0.06
<i>D. mojavensis</i>	162.5	7.6	19	19	0.08	0.08	1%	0.06
<i>D. virilis</i>	161.7	10.8	82	36	0.08	0.08	2%	0.06

To further illustrate that synteny paths reveal structural variations and are robust to contig fragmentation, we computed synteny paths between all pairs of assemblies. On average, the number of synteny paths in each dataset was 20.4% smaller than the number of synteny blocks. We defined the similarity between two assemblies as $S = 1 - N50_{syn}/N50_{max}$, where $N50_{syn}$ corresponds to the synteny paths N50, and $N50_{max}$ is the maximum synteny paths N50 among all assembly pairs. Given the similarity matrix, we used Neighbor-Joining algorithm [31] to infer the phylogenetic tree of all assemblies (Figure 6). The reconstructed tree was structurally consistent with the tree reconstructed based on genomic mutation distances [6].

The running time of Asgan was less than six minutes of wall clock time for each pair of *Drosophila* assemblies. The typical RAM usage was varying from 2 Gb to 6 Gb.

5 Discussion

In this work, we presented Asgan – an algorithm for comparison and evaluation of assembly graphs produced by various assembly approaches. We introduced the concept of synteny paths, which are similar to the synteny blocks abstraction, but take advantage of assembly graph structure. The result of the minimum synteny paths decomposition problem returns the minimal set of synteny paths that traverse all unique edges in two graphs in the same order. We proved that the exact solution of the MSP decomposition is NP-hard and provided a heuristic algorithm that scales to eukaryotic assembly graphs.



■ **Figure 6** (Left) Heatmap showing synteny paths N50 between all pairs of assemblies. (Right). Phylogenetic tree reconstructed based on normalized synteny paths N50 using the Neighbor-Joining method.

We used synteny paths to compare the assembly graphs produced by Canu and Flye from the 21 bacterial datasets. In 14 out of 21 cases, each graph component was covered by a single synteny path, suggesting that both assemblers produced valid assembly graphs with no missing connections. In the remaining seven cases, some graph components were covered by multiple synteny paths, which is not expected for a unichromosomal genome. This reveals possible missing or erroneous connections in the assembly graphs, where the problematic regions are highlighted by synteny path breakpoints. Note that this type of assembly error could not be captured by the alignment of contigs to a reference genome. Thus, the proposed analysis could be useful for validation and debugging of assembly graphs produced by different approaches.

Synteny analysis is a powerful technique for comparative genomics, and a number of tools for synteny blocks decomposition has been developed [29, 28, 21, 8]. However, most of these tools were designed for comparing complete genomes, and it was recently shown [17] that their performance deteriorates when analysing fragmented assemblies. In contrast, synteny paths are taking advantage of the assembly graph structure, and are more robust to the contig fragmentation. We have demonstrated this by analysing the assemblies of 15 *Drosophila* species. As expected, the structure of the reconstructed synteny paths was correlated with mutation distances between the genomes. Using the synteny paths length distribution as a similarity measure, we reconstructed the phylogenetic tree of the analysed species, which was in agreement with the accepted *Drosophila* taxonomy.

It should be possible to extend the synteny paths approach to the comparison of multiple assembly graphs, which could be useful for assembly reconciliation [37, 1]. Intuitively, if one can prove that given assembly graphs share only one optimal MSP solution, this solution will likely correspond to the correct genomic path. However, it is currently unknown how to enumerate all optimal / suboptimal MSP solutions efficiently.

References

- 1 Sergey S Aganezov and Max A Alekseyev. CAMSA: a tool for comparative analysis and merging of scaffold assemblies. *BMC bioinformatics*, 18(15):496, 2017.
- 2 Dmitry Antipov, Anton Korobeynikov, Jeffrey S McLean, and Pavel A Pevzner. hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32(7):1009–1015, 2015.

- 3 Pavel Avdeyev, Shuai Jiang, Sergey Aganezov, Fei Hu, and Max A Alekseyev. Reconstruction of ancestral genomes in presence of gene gain and loss. *Journal of Computational Biology*, 23(3):150–164, 2016.
- 4 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.
- 5 Ali Ebrahimpour Boroojeny, Akash Shrestha, Ali Sharifi-Zarchi, Suzanne Renick Gallagher, S Cenk Sahinalp, and Hamidreza Chitsaz. GTED: Graph traversal edit distance. In *Research in Computational Molecular Biology*, pages 37–53. Springer, 2018.
- 6 Drosophila 12 Genomes Consortium et al. Evolution of genes and genomes on the Drosophila phylogeny. *Nature*, 450(7167):203, 2007.
- 7 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.
- 8 Cristina G Ghiurcuta and Bernard ME Moret. Evaluating synteny for improved comparative studies. *Bioinformatics*, 30(12):i9–i18, 2014.
- 9 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, 2012.
- 10 Govinda M Kamath, Ilan Shomorony, Fei Xia, Thomas A Courtade, and N Tse David. HINGE: long-read assembly achieves optimal repeat resolution. *Genome research*, 27(5):747–756, 2017.
- 11 W James Kent, Robert Baertsch, Angie Hinrichs, Webb Miller, and David Haussler. Evolution’s cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proceedings of the National Academy of Sciences*, 100(20):11484–11489, 2003.
- 12 Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540, 2019.
- 13 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.
- 14 Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10 (8), pages 707–710, 1966.
- 15 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.
- 16 Yu Lin, Sergey Nurk, and Pavel A Pevzner. What is the difference between the breakpoint graph and the de Bruijn graph? *BMC genomics*, 15(6):S6, 2014.
- 17 Dang Liu, Martin Hunt, and Isheng J Tsai. Inferring synteny between genome assemblies: a systematic evaluation. *BMC bioinformatics*, 19(1):26, 2018.
- 18 Serghei Mangul and David Koslicki. Reference-free comparison of microbial communities via de Bruijn graphs. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 68–77. ACM, 2016.
- 19 Alla Mikheenko and Mikhail Kolmogorov. Assembly Graph Browser: interactive visualization of assembly graphs. *Bioinformatics*, 2019.
- 20 Danny E Miller, Cynthia Staber, Julia Zeitlinger, and R Scott Hawley. Highly contiguous genome assemblies of 15 Drosophila species generated using nanopore sequencing. *G3: Genes, Genomes, Genetics*, 8(10):3131–3141, 2018.
- 21 Ilya Minkin, Anand Patel, Mikhail Kolmogorov, Nikolay Vyahhi, and Son Pham. Sibelia: a scalable and comprehensive synteny block generation tool for closely related microbial genomes. In *International Workshop on Algorithms in Bioinformatics*, pages 215–229. Springer, 2013.
- 22 Supratim Mukherjee, Dimitri Stamatis, Jon Bertsch, Galina Ovchinnikova, Hema Y Katta, Alejandro Mojica, I-Min A Chen, Nikos C Kyrpides, and TBK Reddy. Genomes OnLine database (GOLD) v. 7: updates and new features. *Nucleic acids research*, 47(D1):D649–D659, 2018.

- 23 Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington, et al. A whole-genome assembly of *Drosophila*. *Science*, 287(5461):2196–2204, 2000.
- 24 Pavel Pevzner and Glenn Tesler. Transforming men into mice: the Nadeau-Taylor chromosomal breakage model revisited. In *Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 247–256. ACM, 2003.
- 25 Pavel A Pevzner, Haixu Tang, and Glenn Tesler. De novo repeat classification and fragment assembly. *Genome research*, 14(9):1786–1796, 2004.
- 26 Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the national academy of sciences*, 98(17):9748–9753, 2001.
- 27 Adam M Phillippy. New advances in sequence assembly. *Genome Research*, 27:xi–xiii, 2017.
- 28 Sebastian Proost, Jan Fostier, Dieter De Witte, Bart Dhoedt, Piet Demeester, Yves Van de Peer, and Klaas Vandepoele. i-ADHoRe 3.0—fast and sensitive detection of genomic homology in extremely large data sets. *Nucleic acids research*, 40(2):e11–e11, 2011.
- 29 Christian Rödelsperger and Christoph Dieterich. CYNTENATOR: progressive gene order alignment of 17 vertebrate genomes. *PloS one*, 5(1):e8861, 2010.
- 30 Yana Safonova, Anton Bankevich, and Pavel A Pevzner. dipSPAdes: assembler for highly polymorphic diploid genomes. *Journal of Computational Biology*, 22(6):528–545, 2015.
- 31 Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.
- 32 Michael C Schatz, Arthur L Delcher, and Steven L Salzberg. Assembly of large genomes using second-generation sequencing. *Genome research*, 20(9):1165–1173, 2010.
- 33 Alex Shlemov and Anton Korobeynikov. PathRacer: racing profile HMM paths on assembly graph. *BioRxiv*, page 562579, 2019.
- 34 Mitchell R Vollger, Philip C Dishuck, Melanie Sorensen, AnneMarie E Welch, Vy Dang, Max L Dougherty, Tina A Graves-Lindsay, Richard K Wilson, Mark JP Chaisson, and Evan E Eichler. Long-read sequence and assembly of segmental duplications. *Nature methods*, 16(1):88, 2019.
- 35 Ryan R Wick, Louise M Judd, Claire L Gorrie, and Kathryn E Holt. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS computational biology*, 13(6):e1005595, 2017.
- 36 Ryan R Wick, Mark B Schultz, Justin Zobel, and Kathryn E Holt. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20):3350–3352, 2015.
- 37 Aleksey V Zimin, Douglas R Smith, Granger Sutton, and James A Yorke. Assembly reconciliation. *Bioinformatics*, 24(1):42–45, 2007.