18th International Workshop on Algorithms in Bioinformatics

WABI 2018, August 20–22, 2018, Helsinki, Finland

Edited by

Laxmi Parida

Esko Ukkonen

Editors

Laxmi Parida
IBM T. J. Watson Research Center
parida@us.ibm.com

Esko Ukkonen
University of Helsinki
esko.ukkonen@helsinki.fi

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 978-3-95977-082-8

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/978-3-95977-082-8.

Publication date
August, 2018

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://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.WABI.2018.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This proceedings volume contains papers presented at the 18th Workshop on Algorithms in Bioinformatics (WABI 2018), which was held at Aalto University, Helsinki, Finland, from August 20–22, 2018. WABI 2018 was one of seven conferences that were organized as part of ALGO 2018.

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 molecular biology, that are founded on sound models, that are computationally efficient, and that have been implemented and tested in simulations and on real datasets. The meeting's focus is on recent research results, including significant work-in-progress, as well as identifying and exploring directions of future research.

WABI 2018 is grateful for the support to the sponsors of ALGO 2018: Federation of Finnish Learned Societies, Helsinki Institute for Information Technology HIIT, City of Helsinki, and European Association for Theoretical Computer Science (EATCS).

In 2018, a total of 57 manuscripts were submitted to WABI from which 25 were selected for presentation at the conference and are included in this proceedings volume as full papers. Extended versions of selected papers will be invited for publication in a thematic series in the journal Algorithms for Molecular Biology (AMB), published by BioMed Central.

The 25 papers were selected based on a thorough peer review, involving at least three 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 studies, network analysis, sequence and genome analysis, topological data analysis and analysis of mass spectrometry data.

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, Mihai Pop, for his presentation. We also thank Niina Haiminen and Jarkko Toivonen as well as Päivi Lindeman and Pirjo Mulari for their help with editing the proceedings and setting up the poster session. Finally, we thank the ALGO Organizing Committee and Parinya Chalermsook, Petteri Kaski, and Jukka Suomela in particular for their hard work in making all of the local arrangements.

June 2018

Laxmi Parida
Esko Ukkonen

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Chairs

Laxmi Parida	IBM T. J. Watson Research Center
Esko Ukkonen	University of Helsinki

Program Committee

Tatsuya Akutsu	Kyoto University
Lars Arvestad	Stockholm University
Anne Bergeron	Universite du Quebec a Montreal
Paola Bonizzoni	Università di Milano-Bicocca
Alessandra Carbone	Université Pierre et Marie Curie
Benny Chor	Tel-Aviv Univ
Daniel Doerr	Bielefeld University
Nadia El-Mabrouk	University of Montreal
Liliana Florea	Johns Hopkins University
Anna Gambin	Institute of Informatics, Warsaw University
Niina Haiminen	IBM T. J. Watson Research Center
Iman Hajirasouliha	Cornell University
Bjarni Halldorsson	deCODE genetics and Reykjavik University
Katharina Huber	University of East Anglia
Carl Kingsford	Carnegie Mellon University
Gunnar W. Klau	Heinrich Heine University Düsseldorf
Harri Lähdesmäki	Aalto University
Timo Lassmann	Telethon Kids
Yu Lin	Australian National University
Zsuzsanna Liptak	University of Verona
Veli Mäkinen	University of Helsinki
Paul Medvedev	Pennsylvania State University
Bernard Moret	EPFL
Burkhard Morgenstern	University of Goettingen
Vincent Moulton	University of East Anglia
Robert Patro	Stony Brook
Christian N. Storm Pedersen	Aarhus University
Solon Pissis	King's College London
Cinzia Pizzi	Univ Padova
Alex Pothen	Purdue Univ
Teresa Przytycka	NIH
Sven Rahmann	University of Duisburg-Essen
Knut Reinert	FU Berlin
Eric Rivals	CNRS & Univ Montpellier
Simona Rombo	Univ Palermo
Marie-France Sagot	INRIA Grenoble Rhône-Alpes and Univ de Lyon 1
Yasubumi Sakakibara	Keio University
Mikaël Salson	Université Lille 1

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Michael Schatz	Cold Spring Harbor Laboratory
Marcel Schulz	Saarland University
Russell Schwartz	Carnegie Mellon University
Kana Shimizu	Waseda University
Rahul Siddharthan	Institute of Mathematical Sciences, Chennai
Alexandros Stamatakis	Heidelberg Instit. for Theoretical Studies
Jens Stoye	Bielefeld University
Hélène Touzet	University of Lille and INRIA
Lusheng Wang	City University of Hong Kong
Tandy Warnow	University of Illinois at Urbana-Champaign
Yufeng Wu	Univ Connecticut
Louxin Zhang	National University of Singapore
Shaojie Zhang	University of Central Florida
Michal Ziv-Ukelson	Ben Gurion University of the Negev

WABI Steering Committee

Bernard Moret	EPFL
Vincent Moulton	University of East Anglia
Jens Stoye	Bielefeld University
Tandy Warnow	University of Illinois at Urbana-Champaign

ALGO Organizing Committee Chairs

Parinya Chalermsook	Aalto University
Petteri Kaski	Aalto University
Jukka Suomela	Aalto University

Additional Reviewers

Said Sadique Adi
Ilan Ben-Bassat
Sebastian Böcker
Aritra Bose
Christina Boucher
Bastien Cazaux
Annie Chateau
Rayan Chikhi
Neo Christopher Chung
Simone Ciccolella
Leandro I. S. De Lima
Meleshko Dmitrii
Yoann Dufresne
Hannes P. Eggertsson
Jamal Elkhader
Giuditta Franco
Krzysztof Gogolewski
Mauricio Soto Gomez
Aldo Guzmán-Sáenz
Arnaldur Gylfason
Marteinn Hardarson
Lin He
Markus Heinonen
Shahidul Islam
Vasanthan Jayakumar
Dominik Kempa
Kiavash Kianfar
Kuo-Ching Liang
Simone Linz
Guillaume Marçais
Ardalan Naseri
Laurent Noé
Murray Patterson
Leonardo Pellegrina
Pierre Peterlongo
Christopher Pockrandt
Raffaella Rizzi
Giovanna Rosone
Yutaka Saito
Mingfu Shao
Blerina Sinaimeri
Jean-Stéphane Varré
Roland Wittler
Weronika Wronowska
Zohar Yakhini
Guangyu Yang
Hongyu Zheng

List of Authors

Chabrielle Allen
caa6@rice.edu
Rice University
United States

Mai Alzamel
mai.alzamel@kcl.ac.uk
King's College London
United Kingdom

Lorraine A. K. Ayad
lorraine.ayad@kcl.ac.uk
King's College London
United Kingdom

Saugata Basu
sbasu@purdue.edu
Purdue University
United States

Travis Benedict
trb6@rie.edu
Rice University
United States

Bonnie Berger
bab@csail.mit.edu
MIT
United States

Giulia Bernardini
giuliabernardini.morg@gmail.com
University of Milan-Bicocca
Italy

Sebastian Böcker
sebastian.boecker@uni-jena.de
Friedrich Schiller University Jena
Germany

Julian Booth
julius_booth@sfu.ca
Simon Fraser University
Canada

Christina Boucher
cboucher@cise.ufl.edu
University of Florida
United States

Bertrand Caron
b.caron@uq.edu.au
School of Chemistry & Molecular Biosciences,
The University of Queensland, St Lucia
Australia

Bastien Cazaux
bastien.cazaux@lirmm.fr
University of Helsinki
Finland

Cedric Chauve
cedric_chauve@sfu.ca
Simon Fraser University
Canada

Leonid Chindelevitch
leonid@sfu.ca
Simon Fraser University
Canada

Stéphane Chrétien
stephane.chretien@npl.co.uk
National Physical Laboratory
United Kingdom

Michał Aleksander Ciach
m_ciach@student.uw.edu.pl
Institute of Mathematics, Informatics and
Mechanics, University of Warsaw
Poland

Tamal K. Dey
dey.8@osu.edu
The Ohio State University
United States

Riccardo Dondi
riccardo.dondi@unibg.it
Università degli Studi di Bergamo
Italy

Kai Dührkop
Kai.Duehrkop@uni-jena.de
Friedrich-Schiller-University Jena
Germany

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Peter Dulworth
psd2@rice.edu
Rice University
United States

Richard Durbin
rd@sanger.ac.uk
Department of Genetics, University of Cambridge
United Kingdom

Lavinia Egidi
lavinia.egidi@mfn.unipmn.it
Computer Science Institute, University of Eastern Piedmont
Italy

Mohammed El-Kebir
melkebir@illinois.edu
University of Illinois at Urbana-Champaign
United States

Ryan A. Leo Elworth
r.a.leo.elworth@rice.edu
Rice University
United States

Martin S. Engler
martin.engler@cwi.nl
Life Sciences and Health Group, Centrum Wiskunde & Informaticas, Amsterdam
Netherlands

Travis Gagie
travis.gagie@gmail.com
Diego Portales University
Chile

Anna Gambin
aniag@mimuw.edu.pl
Institute of Mathematics, Informatics and Mechanics, University of Warsaw
Poland

Erik Garrison
eg10@sanger.ac.uk
Wellcome Sanger Institute
United Kingdom

Daan P. Geerke
d.p.geerke@vu.nl
AIMMS Division of Molecular and Computational Toxicology, Vrije Universiteit Amsterdam
Netherlands

Roberto Grossi
grossi@di.unipi.it
University of Pisa
Italy

Christophe Guyeux
christophe.guyeux@univ-fcomte.fr
University of Bourgogne Franche-Comté
France

Morteza Chalabi Hajkarim
morteza.hajkarim@bric.ku.dk
University of Copenhagen
Denmark

Till Hartmann
Till.Hartmann@uni-due.de
University of Duisburg-Essen
Germany

Costas S. Iliopoulos
c.iliopoulos@kcl.ac.uk
King's College London
United Kingdom

Nikolai Karpov
nkarpov@iu.edu
Indiana University Bloomington
United States

Gunnar W. Klau
gunnar.klau@hhu.de
Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf
Germany

Dmitry Kosolobov
dkosolobov@mail.ru
Ural Federal University
Russia

Alan Kuhnle
akuhnle418@gmail.com
University of Florida
United States

Manuel Lafond
mlafond2@uOttawa.ca
Université de Sherbrooke
Canada

Marie A. Lataretu
marie.lataretu@uni-jena.de
Friedrich-Schiller-University Jena
Germany

Felipe A. Louza
louza@usp.br
Department of Computing and Mathematics, University of São Paulo
Brazil

Szymon Majewski
smajewski@impan.pl
Institute of Mathematics, Polish Academy of Sciences
Poland

Veli Mäkinen
vmakinen@cs.helsinki.fi
University of Helsinki
Finland

Salem Malikic
smalikic@sfu.ca
Simon Fraser University
Canada

Sayan Mandal
mandal.25@osu.edu
The Ohio State University
United States

Mehrdad Mansouri
mehrdad_mansouri@sfu.ca
Simon Fraser University
Canada

Giovanni Manzini
giovanni.manzini@unipmn.it
Computer Science Institute, University of Eastern Piedmont
Italy

Alan E. Mark
a.e.mark@uq.edu.au
School of Chemistry & Molecular Biosciences, The University of Queensland, St Lucia
Australia

Błażej Miasojedow
bmia@mimuw.edu.pl
Institute of Mathematics, Informatics and Mechanics, University of Warsaw
Poland

Serge Moulin
serge.moulin@hotmail.fr
University of Bourgogne Franche-Comté
France

Martin D. Muggli
martin.muggli@colostate.edu
Colorado State University
United States

Luay Nakhleh
nakhleh@cs.rice.edu
Rice University
United States

Wanda Niemyska
w.niemyska@mimuw.edu.pl
Institute of Mathematics, Informatics and Mechanics, University of Warsaw
Poland

Tuukka Norri
tsnorri@iki.fi
University of Helsinki
Finland

Adam M. Novak
anovak@soe.ucsc.edu
University of California, Santa Cruz
United States

Laxmi Parida
parida@us.ibm.com
IBM T. J. Watson Research Center
United States

Benedict J. Paten
bpaten@ucsc.edu
University of California, Santa Cruz
United States

Nadia Pisanti
pisanti@di.unipi.it
Dipartimento di Informatica, Universita di Pisa, Italy & Erable Team, INRIA
Italy

Solon P. Pissis
solon.pissis@kcl.ac.uk
King's College London
United Kingdom

Nicola Prezza
nicola.prezza@di.unipi.it
University of Pisa
Italy

Simon J. Puglisi
simon.j.puglisi@gmail.com
University of Helsinki
Finland

Md. Khaledur Rahman
morahma@umail.iu.edu
Indiana University Bloomington
United States

Sven Rahmann
Sven.Rahmann@uni-due.de
University of Duisburg-Essen
Germany

Satish Rao
satishr@cs.berkeley.edu
University of California, Berkeley
United States

Pasi Rastas
pasi.rastas@helsinki.fi
University of Helsinki
Finland

Yohei M. Rosen
yohei@ucsc.edu
University of California, Santa Cruz
United States

Giovanna Rosone
giovanna.rosone@unipi.it
Dipartimento di Informatica - Università di Pisa
Italy

S. Cenk Sahinalp
cenksahi@indiana.edu
Indiana University Bloomington
United States

Leena Salmela
leena.salmela@cs.helsinki.fi
University of Helsinki
Finland

Marinella Sciortino
marinella.sciortino@unipa.it
University of Palermo
Italy

Celine Scornavacca
celine.scornavacca@umontpellier.fr
Institut des Sciences de l'Evolution (Université de Montpellier, CNRS, IRD, EPHE)
France

Roded Sharan
roded@post.tau.ac.il
Tel-Aviv University
Israel

Jouni Sirén
jouni.siren@iki.fi
University of California, Santa Cruz
United States

Michał Startek
mist@duch.mimuw.edu.pl
Institute of Mathematics, Informatics and Mechanics, University of Warsaw
Poland

Tamon Stephen
tamon@sfu.ca
Simon Fraser University
Canada

Barak Sternberg
barakolo@gmail.com
Tel-Aviv University
Israel

Guilherme P. Telles
gpt@ic.unicamp.br
Institute of Computing, University of Campinas
Brazil

Eli Upfal
eli@cs.brown.edu
Brown University
United States

Filippo Utro
futro@us.ibm.com
IBM T. J Watson Research Center
United States

Fabio Vandin
fabio.vandin@unipd.it
University of Padova
Italy

Lourens Veen
l.veen@esciencecenter.nl
Netherlands eScience Center, Amsterdam
Netherlands

Margaryta Vityaz
margaryta_vityaz@sfu.ca
Simon Fraser University
Canada

Riku Walve
riku.walve@helsinki.fi
University of Helsinki
Finland

Tandy Warnow
warnow@illinois.edu
University of Illinois at Urbana-Champaign
United States

W. Timothy J. White
tim.white@bihealth.de
Berlin Institute of Health
Germany

Hooman Zabeti
hzabeti@sfu.ca
Simon Fraser University
Canada

Qiuyi (Richard) Zhang
qiyi@math.berkeley.edu
University of California, Berkeley
United States

A Duality-Based Method for Identifying Elemental Balance Violations in Metabolic Network Models

Hooman Zabeti
School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada
hzabeti@sfu.ca

Tamon Stephen
Department of Mathematics, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada.
tamon@sfu.ca

Bonnie Berger
Department of Mathematics and CSAIL, Massachusetts Institute of Technology, Cambridge, MA, 02139, United States of America.
bab@csail.mit.edu

Leonid Chindelevitch
School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada
leonid@sfu.ca

Abstract

Elemental balance, the property of having the same number of each type of atom on both sides of the equation, is a fundamental feature of chemical reactions. In metabolic network models, this property is typically verified on a reaction-by-reaction basis. In this paper we show how violations of elemental balance can be efficiently detected in an entire network, without the need for specifying the chemical formula of each of the metabolites, which enhances a modeler's ability to automatically verify that their model satisfies elemental balance.

Our method makes use of duality theory, linear programming, and mixed integer linear programming, and runs efficiently on genome-scale metabolic networks (GSMNs). We detect elemental balance violations in 40 out of 84 metabolic network models in the BiGG database. We also identify a short list of reactions that are candidates for being elementally imbalanced. Out of these candidates, nearly half turn out to be truly imbalanced reactions, and the rest can be seen as witnesses of elemental balance violations elsewhere in the network. The majority of these violations involve a proton imbalance, a known challenge of metabolic network reconstruction.

Our approach is efficient, easy to use and powerful. It can be helpful to metabolic network modelers during model verification. Our methods are fully integrated into the MONGOOSE software suite and are available at `https://github.com/WGS-TB/MongooseGUI3`.

2012 ACM Subject Classification Applied computing → Biological networks

Keywords and phrases Metabolic network analysis, elemental imbalance, linear programming, model verification

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.1

Funding TS would like to acknowledge financial support from an NSERC Discovery Grant. LC would like to acknowledge financial support from a Sloan Foundation Fellowship and an NSERC Discovery Grant.

Acknowledgements The authors would like to thank Cedric Chauve, Michael Schnall-Levin, Kamyar Khodamoradi, Nafiseh Sedaghat and Reza Miraskarshahi for helpful discussions.

1 Introduction

Genome-scale metabolic network reconstructions (GSMNs) represent the collection of all metabolic reactions available to a specific organism, together with constraints on their direction. These GSMNs, coupled with the constraint-based analysis framework, have been successfully used for predicting the growth rates of various organisms under different environmental conditions [20], the minimal media necessary for growth [10], the essentiality and synthetic lethality of specific genes [2], as well as identifying promising intervention strategies to inhibit their growth [12].

A fundamental property of chemical reactions is elemental balance, the presence of the same number of each type of atom in the reactants (left-hand side of the reaction) and the products (right-hand side of the reaction). For instance, the ethanol combustion reaction,

$$C_2H_5OH + 3O_2 \rightarrow 3H_2O + 2CO_2,$$

is elementally balanced. Each reaction in a GSMN is expected to possess this property [18, 15]. Unfortunately, it turns out that elemental balance is frequently violated in published models [14]. Since the chemical formulas of the metabolites in a metabolic network are sometimes left unspecified, it is a challenge for modelers to use the facilities provided by platforms such as COBRA [18] to check elemental balance in their model directly.

In this paper we present a method for quickly and effectively identifying small groups of reactions in a metabolic network model such that at least one of them violates elemental balance, without the need to specify the elemental formulas of any metabolites. This method is based on the earlier observation that a set of elementally imbalanced reactions may be able to produce "something out of nothing" [14]. Our key theoretical result, first proved in Chindelevitch [7], shows that the ability to produce something out of nothing is mathematically equivalent to a violation of elemental balance; more specifically, if something cannot be produced out of nothing, then there is a set of chemical formulas that make all the reactions elementally balanced. Our method is fully integrated into the MONGOOSE software suite for metabolic network analysis in exact rational arithmetic [8, 13], and is available at `https://github.com/WGS-TB/MongooseGUI3`.

When we apply our method to the collection of existing GSMNs in the BiGG database [11], we find elemental balance violations in 40 out of 84 of them. When we compare these elemental balance violations to the formulas provided in some of the models, we observe that our tool directly identifies a subset of the elementally imbalanced reactions in 28 of the 40 GSMNs, and identifies small sets of reactions containing an imbalanced reaction, which we call "free lunches", in an additional 10 cases. We cannot ascertain which one of these options occurs in the remaining 2 cases since they do not have a formula specified for each metabolite. These results mean that our tool can help modelers efficiently identify elemental imbalances during the process of GSMN reconstruction, even in the absence of detailed chemical formulas for the metabolites.

Although previous authors have identified the presence of elemental imbalances in GSMNs [14], it has not been previously demonstrated, to the best of our knowledge, that this could be done without knowing the exact chemical formulas for the metabolites. Our results also show that our method fails to detect the elementally imbalanced reactions that are present in 32 out of 84 GSMNs in the BiGG database, so it does not guarantee that the network is elementally balanced. Nevertheless, with a detection rate exceeding 55%, this method can be an additional tool for model verification, and we expect it to be helpful to the community thanks to its versatility, transparency, and ease of use.

2 Methods

This section is organized as follows. We start by explaining the connection between elemental balance in a metabolic network and its ability to produce "something out of nothing". We then introduce our three-step method which consists of the following steps:

1. Verify whether the condition forbidding the production of something out of nothing is violated, and if so, identify any metabolites that can be produced out of nothing.
2. Identify a small set of reactions responsible for such a production being possible.
3. Find a lower bound on the number of imbalanced reactions in the network and identify reactions that are likely to be imbalanced.

All of these steps are carried out entirely from the stoichiometric matrix of the network, without any knowledge of the underlying chemical formulas of the metabolites. Steps 1 and 2 are carried out using linear programming, while step 3 uses integer linear programming.

Throughout this paper we use the standard convention that z can also denote the vector with all entries equal to z for any $z \in \mathbb{R}$, and that all vector inequalities (such as $x \geq y$) are interpreted component-wise except for $x \neq y$, which means that x and y are unequal vectors.

Elemental Balance and the No Free Lunch Condition

A GSMN is a collection of metabolic reactions available to an organism. If the GSMN has m metabolites and the n reactions, it is commonly represented by its *stoichiometric matrix* $S \in \mathbb{R}^{m \times n}$, which contains a row for each metabolite and a column for each reaction. The entry S_{ij} is the *stoichiometric coefficient* of metabolite i in reaction j, which is positive if reaction j produces metabolite i, negative if it consumes it, and zero otherwise.

The stoichiometric matrix S is *elementally balanced* with respect to some set of chemical elements (called a "closed system with atomic representation" in [16]) if and only if there exists a vector y with strictly positive components such that

$$y^T S = 0. \tag{1}$$

Indeed, one can think of y as containing the number of elements (atoms) in each metabolite, since every metabolite has a strictly positive (and integer) number of atoms, and the total number of atoms must be the same for the reactants and the products in any elementally balanced reaction.

Therefore, the non-existence of such a vector y implies the presence of an imbalanced reaction in the model. Theorem 2 shows that elemental balance is equivalent to the No *Free Lunch (FL)* condition, which postulates that something cannot be produced out of nothing by the net (overall) reaction of any linear combination of the reactions in the metabolic network. This condition can be written as

$$\mathcal{A} := \{v \mid Sv \geq 0 \text{ and } Sv \neq 0\} = \emptyset$$

The following theorem, known as Stiemke's Alternative [5], is one of several statements closely related to Farkas' Lemma.

▶ **Theorem 1** (Stiemke's Alternative). *Let $A \in \mathbb{R}^{m \times n}$. Then exactly one of the following is true.*

1. $\exists\ x \in \mathbb{R}^n$ *such that* $Ax \geq 0$ *and* $Ax \neq 0$
2. $\exists\ y \in \mathbb{R}^m$ *with* $y > 0$ *such that* $y^T A = 0$

▶ **Theorem 2.** *[7] Let S be a stoichiometric matrix of a metabolic network. Then S is elementally balanced with respect to some set of elements if and only if no linear combination of the reactions in S can result in the production of one or more metabolites out of no reagents.*

Proof. Assume that S is elementally balanced. Then there exists a vector y with strictly positive components such that

$$y^T S = 0.$$

In this case, according to Theorem 1, there is no vector $v \in \mathbb{R}^n$ such that

$$Sv \geq 0 \text{ and } Sv \neq 0.$$

Therefore, the No FL condition holds.
Similarly, if the No FL condition holds for the matrix S, by Theorem 1, we can conclude that there exists a strictly positive vector y such that

$$y^T S = 0.$$

Hence, the matrix S is elementally balanced. ◀

As a result of Theorem 2, we can decide whether a given model violates elemental balance by testing the No FL condition for it. That is, we can check whether there exists a non-negative non-zero vector in the column space of S (i.e. $\mathcal{A} \neq \emptyset$) [7].

Suppose the No FL condition is not satisfied for a stoichiometric matrix S (i.e. $\mathcal{A} \neq \emptyset$). Then there exists a linear combination of the reactions in S that can produce metabolites out of nothing, individually or in combination. For the definitions below we recall that the *support* of a vector $v \in \mathbb{R}^n$ is the set of its non-zero components, $supp(v) := \{i \in \{1, \ldots, n\} \mid v_i \neq 0\}$.

▶ **Definition 3** (Free Lunch)**.** Given a stoichiometric matrix S, we call a subset F of reactions a **free lunch** if there exists a non-zero vector $v \in \mathcal{A}$ with $supp(v) = F$. We call such a v a **vector corresponding** to the free lunch F.

▶ **Definition 4** (Free Lunch Metabolite)**.** For a free lunch F with a corresponding vector $v \in \mathcal{A}$, we call the metabolite t a **free lunch metabolite** if $t \in supp(Sv)$.

Step 1: No FL and all possible FL metabolites

In the first step of our method we introduce a test to verify the No FL condition for a given stoichiometric matrix S and identify the set of all possible FL metabolites. The verification of the No FL condition is performed by checking the feasibility of the linear program

$$Sv = w, w \geq 0, 1^T w \geq 1, \text{ where } v \in \mathbb{R}^n, w \in \mathbb{R}^m. \tag{2}$$

In addition, recall that $\|x\|_0 := |\{i \mid x_i \neq 0\}|$ denotes the size of the support of a vector x and t is a FL metabolite if $t \in supp(Sv)$ for some $v \in \mathcal{A}$. In the following lemma we show that the support of Sv for the vector v solving the non-linear program

$$\max_v \|Sv\|_0 \text{ subject to } Sv \geq 0, \ v \in \mathbb{R}^n \tag{3}$$

is the set of all possible FL metabolites for the matrix S. Note that if the optimum value of (3) is greater than zero, then (2) is feasible and the No FL Condition is not satisfied.

▶ **Lemma 5.** *Given a stoichiometric matrix $S \in \mathbb{R}^{m\times n}$, the support of Sv where v is a solution of the non-linear program in (3) corresponds to the set of all possible FL metabolites for the given matrix S.*

Proof. Let $\mathcal{T}_S$ be the set of all possible FL metabolites for S and $\hat{v}$ be a maximizer of (3). By definition of an FL metabolite we can observe that

$$supp(S\hat{v}) \subseteq \mathcal{T}_S.$$

Now, assume that $\mathcal{T}_S - supp(S\hat{v}) \neq \emptyset$ and let $t \in \mathcal{T}_S - supp(S\hat{v})$. Since t is a FL metabolite, there exists a vector $u \in \mathcal{A}$ such that $t \in supp(Su)$. Note that in this case $supp(S\hat{v}) \subsetneq supp(S(\hat{v}+u))$, which contradicts the maximality of $\|S\hat{v}\|_0$. Therefore, $\mathcal{T}_S = supp(\arg\max_v \|Sv\|_0)$. ◀

Although (3) is a non-linear program, we can in fact solve it with the following linear program.

▶ **Lemma 6.** *Given a matrix $S \in \mathbb{R}^{m\times n}$, the support of w, where w is a solution of*

$$\max_w (1^T w), \text{ subject to } Sv - w \geq 0,\ 0 \leq w \leq 1,\ (v,w) \in \mathbb{R}^{n+m} \tag{4}$$

corresponds to the set of all possible FL metabolites for the given matrix S.

Proof. Let (v,w) be a solution for (4). Also let p be a maximizer of (3) and define $q = Sp$. We show that

$$1^T w = \|q\|_0.$$

Note that since $0 \leq w \leq 1$, we have $1^T w = \|w\|_1 \leq \|q\|_0$. Now assume that $1^T w < \|q\|_0$. Let

$$\tilde{v} = \frac{v+p}{\min\{\min_{i\in supp(w)} w_i, \min_{j\in supp(q)} q_j\}}.$$

Also, define $\tilde{w} \in \mathbb{R}^m$ such that

$$\tilde{w}_i = \begin{cases} 1 & \text{if } \max\{w_i, q_i\} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Since the non-zero components of vector $S\tilde{v}$ are greater than or equal to 1, we found $(\tilde{v},\tilde{w}) \in \mathbb{R}^{n+m}$ that is in the feasible region of (4) and

$$1^T\tilde{w} \geq \|q\|_0 > 1^T w$$

which contradicts the maximality of $1^T w$. With a similar argument we can show that the solution of (4) is unique. Therefore, by Lemma 5, we are done. ◀

Step 2: Minimal Free Lunches

Theorem 2 shows that violation of the No FL condition implies violation of elemental balance in at least one reaction in the model. In the following section we continue the process in order to identify minimal subsets of reactions that contain such imbalanced reactions. We also show that, regardless of the chemical formulas assigned to the metabolites participating in the model, such a subset always contains an imbalanced reaction.

▶ **Definition 7** (Minimal Free Lunch)**.** The Free Lunch F is called a **Minimal Free Lunch**, if no proper subset of F is a free lunch.

A minimal FL can be computed with respect to any desired subset of FL metabolites. Let $\mathcal{T}$ be a non-empty subset of all FL metabolites in a model, and let us define the vector $w \in \mathbb{R}^m$ to be an indicator vector such that $w_i = 1$ if metabolite $i \in \mathcal{T}$ and $w_i = 0$ otherwise, for $i \in \{1, \ldots, m\}$. Then the smallest subset of reactions producing all FL metabolites in $\mathcal{T}$ can be computed by solving the following non-linear optimization problem:

$$\arg\min_{v} \|v\|_0 \text{ subject to } Sv - w \geq 0, v \in \mathbb{R}^n, w \in \mathbb{R}^m \tag{5}$$

This problem is NP-hard and challenging in practice [4], so we use the following steps to identify the minimal subsets instead. First we use the iterative reweighted ℓ_1 minimization algorithm, presented by Candès, Wakin and Boyd [6], to find an approximation for the non-convex minimization (5). We then reduce the (possibly non-minimal) FL found to a minimal FL by removing each reaction in turn and checking if the resulting combination is still a FL; if it is not, the reaction is restored. This procedure produces a minimal FL [7].

▶ **Lemma 8.** *Suppose $\mathcal{M}$ is a minimal FL. Then, for every assignment of chemical formulas to the metabolites involved in $\mathcal{M}$ there exists an elementally imbalanced reaction in $\mathcal{M}$. Furthermore, for each reaction $r \in \mathcal{M}$ there exists an assignment of chemical formulas to these metabolites that makes r imbalanced while making the reactions in $\mathcal{M} - \{r\}$ balanced.*

Proof. Assume $\mathcal{M}$ is a minimal FL. Let S' be the submatrix of the stoichiometric matrix S corresponding to the reactions in $\mathcal{M}$. Since $\mathcal{M}$ is a FL, by Theorem 2, there exits no strictly positive vector y for which (1) holds. Therefore, if we think of the vector y as containing the number of elements in each metabolite, regardless of the chemical formulas assigned to the metabolites in S', there exists at least one imbalanced reaction in $\mathcal{M}$.

In addition, since $\mathcal{M}$ is minimal, for each $r \in \mathcal{M}$, $\mathcal{M} - \{r\}$ cannot be a FL. Let S'' be the submatrix of S corresponding to the reactions in $\mathcal{M} - \{r\}$. Therefore, by Theorem 2, there exits a strictly positive vector y such that

$$y^T S'' = 0.$$

We can assume that the coefficients of y are rational, since S has rational entries. Let $\hat{y} = Ny$ be the vector obtained by scaling y by the least common multiple N of the denominators in y. Then $\hat{y}^T S'' = 0$ as well, and $\hat{y}$ contains positive integers. Thus we can set the formula of metabolite j to $C_{\hat{y}_j}$, and observe that (1) holds for $\mathcal{M} - \{r\}$ with these one-atom formulas. ◀

We solve the linear programs in Steps 1 and 2 using the QSOpt_ex solver [3], which checks that the solutions are correct in exact rational arithmetic. Some of our previous work [8, 13] has argued that this is necessary in order to ensure accuracy and reproducibility of the solutions. We use the QSOpt_ex API for Python [19] created by Jon Steffensen [17] to streamline the process.

Step 3: Free Lunch Witnesses

In the current section we aim to find the smallest number of reactions that are guaranteed to be involved in at least one FL each. We observe that the optimum of the minimization problem

$$\min_{y} \|y^T S\|_0, \text{ subject to } y > 0 \tag{6}$$

represents a lower bound on the minimum number of FL's over all possible metabolite formulas, and the minimizer represents a smallest set of reactions involved in FL's. Moreover,

the optimum value of (6) also gives us a lower bound on the number of imbalanced reactions in the model for an arbitrary or a specific set of metabolite formulas. In other words, if the optimum value of (6) is k, we know that at least k distinct reactions are elementally imbalanced, no matter what metabolite formulas we choose, and in particular, the metabolic network cannot be elementally balanced unless $k = 0$. This motivates the following

▶ **Definition 9** (Free Lunch Witness)**.** The reaction r is called a **Free Lunch Witness**, if $r \in supp(y^T S)$ for some $y \in \text{argmin}_{y>0} \|y^T S\|_0$.

Even though we cannot guarantee this, it turns out to be more likely than not in practice that a FL witness is elementally imbalanced for a given set of metabolite formulas, as we discuss in more detail in the Results section.

In order to solve (6), we can use the following mixed integer linear program:

$$\min_x \ 1^T x \text{ subject to } x \geq -y^T S, \ x \geq y^T S, y \geq \epsilon, \ x \in \{0,1\}^n, \ y \in \mathbb{R}^m \tag{7}$$

where ϵ is a small positive scalar (in practice, we use $\epsilon = 10^{-4}$). The lower bound of ϵ ensures that the vector $y = 0$ is not an admissible solution of (6), so that any y that is optimal for (7) can be scaled to a $\hat{y}$ that is optimal for (6) (with the same objective value), and vice versa.

We point out that the solution vector to the optimization problem (7) is in general not unique (even though the optimal value is), and may change based on the solver as well as the order of the reactions or metabolites in the stoichiometric network S. We used CPLEX 12.8.0 [1] via its API for Python [19] to solve the integer optimization problem (7).

3 Results

We analyzed a collection of 84 previously published metabolic network models in the BiGG database [11]. From our analysis we excluded any pseudo-reactions, namely, the biomass reactions (those containing the word "biomass" or "growth" in their name) as well as import and export reactions (those that have only positive or only negative stoichiometric coefficients, i.e. produce something out of nothing or reduce something to nothing, respectively). This ensures that all the reactions included in a stoichiometric matrix S were *bona fide* biochemical reactions, with both reactant and product sides non-empty. We expected all those reactions to be elementally balanced, as required in the model construction protocol [18].

Nevertheless, we found instances of elemental balance violation in 72 of the 84 network models by using the provided chemical formulas. An additional 7 of the models had no instance of elemental balance violation and 5 of them had no specified chemical formulas for the metabolites. Note that we only consider the following *bona fide* atoms in our decision about whether a reaction is imbalanced: C, Fe, H, Mg, Na, N, O, P, S, Z (the latter represents a photon, which can for instance be used by photosynthetic organisms). We do not consider symbols for functional groups or other moieties such as I, K, M, R, U, X, Y; in other words, if all the *bona fide* atoms were balanced, the reaction was considered to be balanced no matter whether these additional symbols were present on both sides of the reaction or only one.

The pie chart in Figure 1 shows the relative frequency of the atoms that are not balanced. There are a total of 810 imbalanced reactions, and 1492 atomic imbalances, so an average imbalanced reaction has just under two imbalanced atoms. Almost half of all atomic imbalances are proton or hydrogen imbalances, involving the H atom.

We applied the first step of our method to find instances of FL metabolites that result from elemental balance violations. We found that 40 out of 84 models include at least one FL metabolite. The number of FL metabolites ranged from 3 to 1791, with a mean just

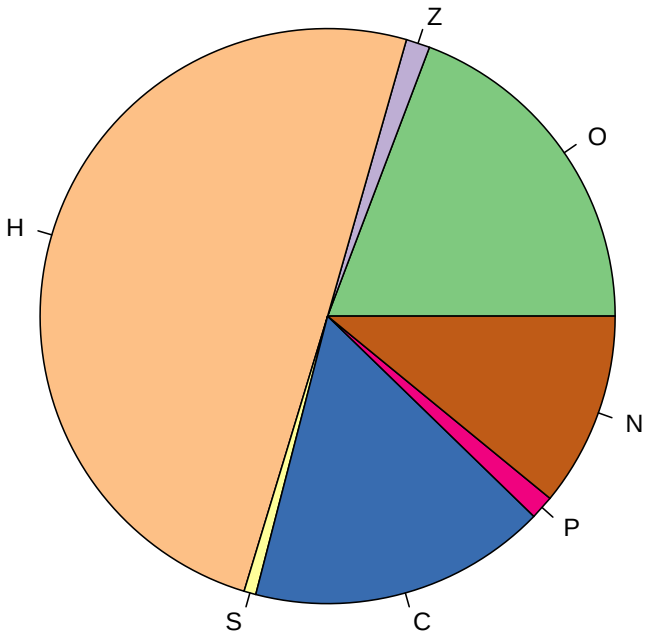


Figure 1 Frequency of imbalanced reactions in the BiGG networks by imbalanced atom.

under 139. There are a total of 5552 FL metabolites, of which 2912 are distinct; more than 90% of them happen to be in four of the models, namely, those for *Homo sapiens* (RECON1), *Mus musculus* (iMM1415), *Escherichia coli* (iECIAI1_1343), and *Salmonella enterica* (STM_v1_0). The vast majority of the other models only have 3 FL metabolites, which are predominantly protons (H) in three different compartments, namely, [c] (cytosol), [p] (periplasm), and [e] (extracellular space).

The pie chart in Figure 2 shows the relative frequency of the FL metabolites that are found in more than 5 of the metabolic networks we investigated. There are a total of 9 such metabolites, which all happen to be currency metabolites - H (protons), H_2O (water), H_2O_2 (hydrogen peroxide), O_2 (oxygen), and SO_2 (sulfur dioxide) - in the compartments listed above.

We also applied the second step to find a minimal FL with respect to one of the FL metabolites[1] in each of these 40 models, and their sizes ranged from 2 to 71, with a mean of 5.8. By using the specified chemical formulas, we were able to verify the existence of at least one imbalanced reaction in each minimal FL in 38 out of 40 models, in accordance with our theoretical results (the remaining 2 models, namely, iMM1415 and iECIAI1_1343, did not have any chemical formulas specified for their metabolites).

Finally, by applying the third step of our method, we found small sets of FL witnesses in all 40 models, with sizes ranging from 1 to 12, and a mean of 2.2. Just as for the second step, we compared the set of detected FL witnesses to the set of imbalanced reactions based on the chemical formulas provided in 38 of the models. We observed that, in 28 out of 38 models, at least one of the FL witnesses identified by our method was an imbalanced reaction. Furthermore, just over half (41 out of 79) of the FL witnesses were imbalanced reactions.

[1] We chose the FL metabolite in following way. First we found an optimum solution $(v, w) \in \mathbb{R}^{n+m}$ of $\min_v \|v\|_1$ subject to: $Sv - w \geq 0, w \geq 0, 1^T w \geq 1$ to roughly approximate the subset of FL metabolites for which we may have a small FL. We then chose the first index in $supp(w)$ as the desired FL metabolite.

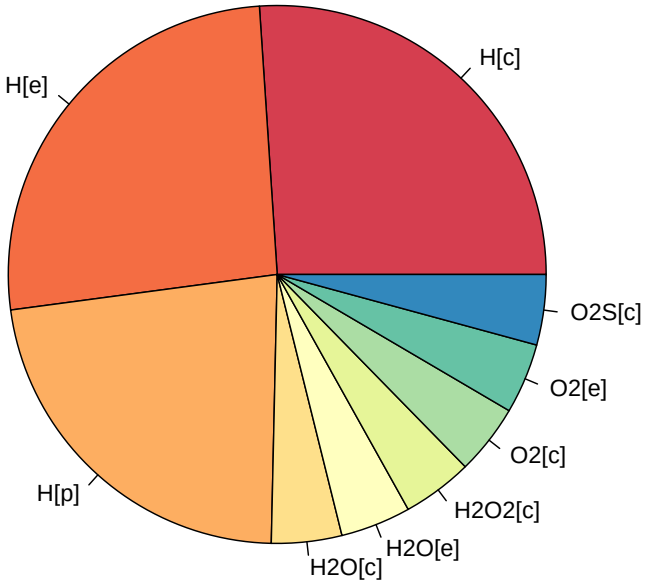


Figure 2 Relative frequency of the most common free lunch metabolites in the BiGG networks.

This suggests that FL witnesses can be good candidates for imbalanced reactions in the absence of chemical formulas for the metabolites in a network.

A detailed summary of the 40 models with at least one FL is presented in Table 1. We now discuss in detail the results of applying our algorithms to four particular models, chosen for illustration purposes. They are arranged in increasing order of complexity.

The first example is iECB_1328, a model for *Escherichia coli* B strain REL606 (the first model in Table 1). For this model, our method identifies 3 FL metabolites, which happen to be protons, H, in three different compartments. We also find a minimal FL that consists of two reactions - OPET decarboxylase and 5-carboxy-2-oxohept-3-enedioate decarboxylation:

$$C_8H_5O_7[c] \rightarrow CO_2[c] + C_7H_6O_5[c] \tag{8a}$$

$$H[c] + C_8H_5O_7[c] \rightarrow CO_2[c] + C_7H_6O_5[c] \tag{8b}$$

We can see that by subtracting reaction (8b) from reaction (8a), we get a net overall reaction that produces the FL metabolite H[c] out of nothing, while all other metabolites cancel out. The OPET decarboxylase reaction is also detected as the single FL witness in this model, and in fact, it is easy to see that the H atom is not balanced in this reaction (there are 5 H's on the reagent side and 6 on the product side). Thus, in this model, the FL witness is an imbalanced reaction.

The second example is iBWG_1329, another model for *Escherichia coli*, but a different strain, BW2952. Similarly to the first example, our method identifies 3 FL metabolites, which happen to be protons, H, in three different compartments. This time, the minimal FL consists of three reactions - nucleoside triphosphate pyrophosphorylase (atp), phenylacetate-CoA ligase, and phenylacetyl CoA thioesterase:

$$C_{10}H_{12}N_5O_{13}P_3[c] + H_2O[c] \rightarrow C_{10}H_{12}N_5O_7P[c] + H[c] + HO_7P_2[c] \tag{9a}$$

$$C_{10}H_{12}N_5O_{13}P_3[c] + C_{21}H_{32}N_7O_{16}P_3S[c] + C_8H_7O_2[c] \rightarrow$$
$$\rightarrow C_{10}H_{12}N_5O_7P[c] + C_{29}H_{38}N_7O_{17}P_3S[c] + HO_7P_2[c] \tag{9b}$$

$$H_2O[c] + C_{29}H_{38}N_7O_{17}P_3S[c] \rightarrow C_{21}H_{32}N_7O_{16}P_3S[c] + C_8H_7O_2[c] \tag{9c}$$

We can see that subtracting (9a) from the sum of (9b) and (9c) results in the production of the FL metabolite H[c] out of nothing. We can also see that the H atoms are not balanced in (9c), with 40 on the reagent side and 39 on the product side, while (9a) and (9b) are balanced. In this model, our method detects reaction (9b) as the unique FL witness. Thus, in this model, the FL witness is itself a balanced reaction, but is part of a FL of size 3.

The third example is iJN678, a model for *Synechocystis sp.* PCC 6803, a cyanobacterium that can grow by oxygenic photosynthesis [9]. For this model, our method once again identifies 3 FL metabolites, which this time are photons (Z) in three compartments. The minimal FL contains five reactions - ATP synthetase(u), cyanophycinase, cyanophycin sinthetase, cytochrome oxidase bd (plastocianine-8 2 protons) (lumen), and photosystem II, which we denote by R_1 through R_5 for convenience:

$$R_1 : 3C_{10}H_{12}N_5O_{10}P_2[c] + 3HO_4P[c] + 14H[u] \rightarrow \\ \rightarrow 3C_{10}H_{12}N_5O_{13}P_3[c] + 11H[c] + 3H_2O[c] \tag{10a}$$

$$R_2 : 2H_2O[c] + C_{10}H_{17}N_5O_4R[c] \rightarrow C_6H_{15}N_4O_2[c] + C_4H_6NO_4[c] + R[c] \tag{10b}$$

$$R_3 : C_6H_{15}N_4O_2[c] + C_4H_6NO_4[c] + 2C_{10}H_{12}N_5O_{13}P_3[c] + R[c] \rightarrow \\ \rightarrow 2C_{10}H_{12}N_5O_{10}P_2[c] + 2H[c] + 2HO_4P[c] + C_{10}H_{17}N_5O_4R[c] \tag{10c}$$

$$R_4 : C_8H_{10}O_2[u] + \frac{1}{2}O_2[u] \rightarrow C_8H_8O_2[u] + H_2O[u] \tag{10d}$$

$$R_5 : 2H[c] + C_8H_8O_2[u] + H_2O[u] + 2Z[c] \rightarrow 2H[u] + C_8H_{10}O_2[u] + \frac{1}{2}O_2[u] \tag{10e}$$

It is easy to check (but far from obvious to guess!) that $2R_1 + 3R_2 + 3R_3 + 14R_4 + 14R_5$ would result in the consumption of 28 units of the FL metabolite Z[c], while all other metabolites cancel out. Also, note that Z is only present on the reagent side of (10e). In this example, our method detects reaction (10e) as one of the 3 FL witnesses, and in this case, it is the one imbalanced FL witness.

For the final example we chose a model that does not have any chemical formulas assigned to its metabolites, namely, model iECIAI1_1343, which represents *Escherichia coli.* IAI1. For this model our method identifies 1279 FL metabolites and a minimal FL which contains two reactions - OPET decarboxylase and 5-carboxy-2-oxohept-3-enedioate decarboxylation:

$$\text{5-Carboxy-2-oxohept-3-enedioate}[c] \rightarrow \text{2-Hydroxyhepta-2-4-dienedioate}[c] + CO_2[c] \tag{11a}$$

$$H[c] + \text{5-Carboxy-2-oxohept-3-enedioate}[c] \rightarrow \text{2-Hydroxyhepta-2-4-dienedioate}[c] + CO_2[c] \tag{11b}$$

We can see that by subtracting reaction (11b) from reaction (11a), we get the FL metabolite H[c] out of nothing, while all other metabolites cancel out. Note that this is the same FL as the one identified in equations (8a) and (8b), although in this model they are not specified via chemical formulas. Moreover, our method identifies 4 FL witnesses; as a result of the absence of chemical formulas, we are not able to verify whether they are elementally balanced.

4 Conclusion

Our approach represents the first attempt to automatically verify elemental balance in a GSMN without access to the chemical formulas for the metabolites. It quickly and efficiently identifies small sets of reactions which must contain at least one imbalanced reaction, or determines that no such set exists. In addition, it can provide a lower bound on the number

Table 1 The summary of our results. n and m denote the number of reactions and metabolites, respectively. m_{FL} and n_{FL} is the number of FL metabolites and the size of a minimal FL we found, respectively. n_I and n_{FLW} are the numbers of imbalanced reactions and FL witness reactions, respectively. Finally, n_{IFLW} if the number of FL witness reactions that are imbalanced.

Model	Organism	n	m	m_{FL}	n_I	n_{FL}	n_{FLW}	n_{IFLW}
iECB_1328	E. coli B str. REL606	2748	1951	3	10	2	1	1
iNJ661	M. tuberculosis H37Rv	1025	825	192	10	4	4	1
iY75_1357	E. coli str. K-12	2759	1953	3	9	3	1	0
iMM1415	Mus musculus	3726	2775	1767	N/A	12	6	N/A
iECO103_1326	E. coli O26:H11 str. 11368	2758	1958	3	17	2	2	2
iECUMN_1333	E. coli UMN026	2740	1935	3	7	2	1	1
iEcolC_1368	E. coli ATCC 8739	2768	1969	3	16	2	2	1
RECON1	Homo sapiens	3741	2766	1791	5	40	4	1
iECO111_1330	E. coli O111:H- str. 11128	2760	1959	3	19	2	2	2
iSBO_1134	S. boydii Sb227	2591	1908	3	7	2	1	1
iRC1080	C. reinhardtii	2191	1706	21	61	6	12	0
iJB785	Synechococcus elongatus	849	768	16	11	14	4	0
iS_1188	S. flexneri 2a str. 2457T	2619	1914	3	4	2	1	1
STM_v1_0	S. enterica subsp. enterica	2545	1802	344	1	71	2	0
iECBD_1354	E. coli BL21	2748	1952	3	10	2	1	1
iECO26_1355	E. coli O26:H11 str. 11368	2780	1965	3	17	2	2	2
iWFL_1372	E. coli W	2782	1973	3	21	2	2	2
iECSE_1348	E. coli SE11	2768	1957	3	18	2	2	2
iECD_1391	E. coli BL21	2741	1943	3	10	2	1	1
iEcDH1_1363	E. coli DH1	2750	1949	3	8	3	1	1
iECDH1ME8569_1439	E. coli DH1	2755	1950	3	9	3	1	0
iECDH10B_1368	E. coli str. K-12	2742	1947	3	9	3	1	1
iB21_1397	E. coli BL21	2741	1943	3	10	2	1	1
iEcE24377_1341	E. coli O139:H28 str. E24377A	2763	1972	3	17	3	1	1
iUMNK88_1353	E. coli UMNK88	2777	1969	3	18	3	1	0
iETEC_1333	E. coli ETEC H10407	2756	1962	3	14	3	1	0
iECW_1372	E. coli W	2782	1973	3	21	2	2	2
iSF_1195	S. flexneri 2a str. 301	2630	1917	3	7	2	1	1
iSbBS512_1146	S. boydii CDC 3083-94	2591	1910	3	7	2	1	0
iEKO11_1354	E. coli KO11FL	2778	1972	3	21	2	2	2
iCHOv1	Cricetulus griseus	6663	4456	46	92	4	9	4
iSSON_1240	S. sonnei Ss046	2693	1936	3	9	2	1	1
iJN678	Synechocystis sp. PCC 6803	863	795	3	9	5	3	3
iSFxv_1172	S. flexneri 2002017	2638	1918	3	7	2	1	1
iBWG_1329	E. coli BW2952	2741	1949	3	9	3	1	0
iECIAI1_1343	E. coli IAI1	2765	1968	1279	N/A	2	4	N/A
iSFV_1184	S. flexneri 5 str. 8401	2621	1917	3	1	2	1	0
iLB1027_lipid	P. tricornutum	4456	2172	3	6	6	2	1
iEcHS_1320	E. coli HS	2753	1963	3	21	2	2	2
iEC55989_1330	E. coli 55989	2756	1953	3	14	3	1	1
Average				138.8	14.0	5.8	2.2	1

of imbalanced reactions present in the model. Lastly, it identifies all the metabolites that can be produced out of nothing by the overall reaction of a combination of model reactions.

The difference in the number of elementally imbalanced models examined with (72) and without (40) chemical formulas (out of a total of 84) can be addressed by examining equation (1). Since the vector y in (1) is not unique, different y vectors can represent different sets of metabolites. Therefore, although a model with stoichiometric matrix S may contain

imbalanced reactions for a specific set of chemical formulas, the same stoichiometric matrix S may be elementally balanced for a different set of formulas. Thus, our method can guarantee a violation of elemental balance by finding one or more FL metabolites, but the absence of such metabolites does not guarantee that the model is elementally balanced.

Nevertheless, we believe that our approach provides a complimentary way of ascertaining a model's soundness and point out potential issues, as part of the model verification process. It can - and should - be accompanied by a verification of the elemental balance of individual reactions whenever possible, i.e. when the chemical formulas are specified for its metabolites. We thus hope that our method is a useful contribution to metabolic network model verification.

References

1 CPLEX optimizer. URL: `www-01.ibm.com/software/integration/optimization/cplex-optimizer`.

2 V Acuña, F Chierichetti, V Lacroix, A Marchetti-Spaccamela, M-F Sagot, and L Stougie. Modes and cuts in metabolic networks: complexity and algorithms. *BioSystems*, 95:51–60, 2009.

3 D Applegate, W Cook, S Dash, and D Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35:693–699, 2007.

4 D Bertsimas, A King, and R Mazumder. Best subset selection via a modern optimization lens. *Ann. Statist.*, 44(2):813–852, 2016.

5 KC Border. Alternative linear inequalities. *Cal Tech Lecture Notes*, 2013.

6 EJ Candès, MB Wakin, and SP Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier analysis and applications*, 14(5-6):877–905, 2008.

7 L Chindelevitch. *Extracting Information from Biological Networks*. PhD thesis, MIT, 2010.

8 L Chindelevitch, J Trigg, A Regev, and B Berger. An exact arithmetic toolbox for a consistent and reproducible structural analysis of metabolic network models. *Nature Communications*, 5, 2014.

9 T Heidorn, D Camsund, H Huang, P Lindberg, P Oliveria, K Stensjo, and P Lindblad. Synthetic biology in cyanobacteria: Engineering and analyzing novel functions. In *Methods in Enzymology*, volume 497, pages 539–579. Academic Press, 2011.

10 M Imieliński, C Belta, H Rubin, and Á Halász. Systematic analysis of conservation relations in escherichia coli genome-scale metabolic network reveals novel growth media. *Biophysical Journal*, 90(8):2659–2672, 2006.

11 ZA King, J Lu, A Dräger, P Miller, S Federowicz, JA Lerman, A Ebrahim, BO Palsson, and NE Lewis. BiGG Models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Research*, 44:D515–D522, 2015.

12 S Klamt and E Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20:226–234, 2004.

13 C Le and L Chindelevitch. The MONGOOSE rational arithmetic toolbox. In Marco Fondi, editor, *Metabolic Network Reconstruction and Modeling*, volume 1716 of *Methods in Molecular Biology*, pages 77–99. Humana Press, New York, NY, 2018.

14 J Monk, J Nogales, and B Palsson. Optimizing genome-scale network reconstructions. *Nature Biotechnology*, 32:447–452, 2014.

15 A Ravikrishnan and K Raman. Critical assessment of genome-scale metabolic networks: the need for a unified standard. *Briefings in Bioinformatics*, 16(6):1057–1068, 2015.

16 S Schuster and T Höfer. Determining all extreme semi-positive conservation relations in chemical reaction systems: a test criterion for conservativity. *Journal of the Chemical Society, Faraday Transactions*, 87:2561–2566, 1991.

17 JL Steffensen, K Dufault-Thompson, and Y Zhang. Psamm: A portable system for the analysis of metabolic models. *PLOS Computational Biology*, 12(2):e1004732, 2016.

18 I Thiele and B Palsson. A protocol for generating a high-quality genome-scale metabolic reconstruction. *Nature Protocols*, 5:93–121, 2010.

19 G van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1995.

20 A. Varma and B. Palsson. Stoichiometric flux balance models quantitatively predict growth and metabolic by-product secretion in wild-type *Escherichia coli* w3110. *Appl. Environ. Microbiol.*, 60:3724–3731, 1994.

Prefix-Free Parsing for Building Big BWTs

Christina Boucher[1]
CISE, University of Florida
Gainesville, FL, USA
https://orcid.org/0000-0001-9509-9725

Travis Gagie[2]
EIT, Diego Portales University
Santiago, Chile
1exCeBiB
Santiago, Chile
https://orcid.org/0000-0003-3689-327X

Alan Kuhnle[3]
CISE, University of Florida
Gainesville, FL, USA
https://orcid.org/0000-0001-6506-1902

Giovanni Manzini[4]
University of Eastern Piedmont
Alessandria, Italy
1exIIT, CNR
Pisa, Italy
https://orcid.org/0000-0002-5047-0196

Abstract

High-throughput sequencing technologies have led to explosive growth of genomic databases; one of which will soon reach hundreds of terabytes. For many applications we want to build and store indexes of these databases but constructing such indexes is a challenge. Fortunately, many of these genomic databases are highly-repetitive – a characteristic that can be exploited and enable the computation of the Burrows-Wheeler Transform (BWT), which underlies many popular indexes. In this paper, we introduce a preprocessing algorithm, referred to as *prefix-free parsing*, that takes a text T as input, and in one-pass generates a dictionary D and a parse P of T with the property that the BWT of T can be constructed from D and P using workspace proportional to their total size and $O(|T|)$-time. Our experiments show that D and P are significantly smaller than T in practice, and thus, can fit in a reasonable internal memory even when T is very large. Therefore, prefix-free parsing eases BWT construction, which is pertinent to many bioinformatics applications.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Burrows-Wheeler Transform, prefix-free parsing, compression-aware algorithms, genomic databases

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.2

[1] Partially supported by National Science Foundation grant 1618814.
[2] Partially supported by FONDECYT grant 1171058.
[3] Partially supported by National Science Foundation grant 1618814 and a post-doctoral fellowship from the University of Florida Informatics Institute.
[4] Partially supported by PRIN grant 201534HNXC

Supplement Material Source code: `https://gitlab.com/manzai/Big-BWT`

Acknowledgements The authors thank Risto Järvinen for the insight they gained from his project on `rsync` in the Data Compression course at Aalto University.

1 Introduction

The money and time needed to sequence a genome have shrunk shockingly quickly and researchers' ambitions have grown almost as quickly: the Human Genome Project cost billions of dollars and took a decade but now we can sequence a genome for about a thousand dollars in about a day. The 1000 Genomes Project [21] was announced in 2008 and completed in 2015, and now the 100,000 Genomes Project is well under way [22]. With no compression 100,000 human genomes occupy roughly 300 terabytes of space, and genomic databases will have grown even more by the time a standard research machine has that much RAM. At the same time, other initiatives have began to study how microbial species behave and thrive in environments. These initiatives are generating public datasets which are just are as equally challenging from a size perspective as the 100,000 Genomes Project. For example, in recent years, there has been an initiative to move toward using whole genome sequencing to accurately identify and track foodborne pathogens (e.g. antibiotic resistant bacteria) [5]. This led to the existence of GenomeTrakr, which is a large public effort to use genome sequencing for surveillance and detection of outbreaks of foodborne illnesses. Currently, the GenomeTrakr effort includes over 100,000 samples, spanning several species available through this initiative – a number that continues to rise as datasets are continually added [19]. Unfortunately, analysis of this data is limited due to their size, even though the similarity between genomes of individuals of the same species means the data is highly compressible.

These public databases are used in various applications – e.g., to detect genetic variation within individuals, determine evolutionary history within a population, and assemble the genomes of novel (microbial) species or genes. Pattern matching within these large databases is fundamental to all these applications, yet repeatedly scanning these – even compressed – databases is infeasible. Thus for these and many other applications, we want to build and use indexes from the database. Since these indexes should also fit in RAM and cannot rely on word boundaries, there are only a few candidates. Many of the popular indexes in bioinformatics are based on the Burrows-Wheeler Transform (BWT) [4] and there have been a number of papers about building BWTs for genomic databases; see, e.g., [18] and references therein. However, it is difficult to process anything more than a few terabytes of raw data per day with current techniques and technology because of the difficulty of working in external memory.

Since genomic databases are often highly repetitive, we revisit the idea of applying a simple compression scheme and then computing the BWT from the resulting encoding in internal memory. This is far from being a novel idea – e.g., Ferragina, Gagie and Manzini's `bwtdisk` software [7] could already in 2010 take advantage of its input being given compressed, and Policriti and Prezza [17] recently showed how to compute the BWT from the LZ77 parse of the input using $O(n(\log r + \log z))$-time and $O(r + z)$-space, where n is the length of the uncompressed input, r is the number of runs in the BWT and z is the number of phrases in the LZ77 parse – but we think the preprocessing step we describe here, *prefix-free parsing*, stands out because of its simplicity and flexibility. Specifically, the parsing algorithm itself is straightforward and it can either be made to work using a single pass over the data on disk or it can be parallelized. Once we have the results of the parsing, which are a dictionary

and a parse, building the BWT out of them is more involved, but when our approach works well, the dictionary and the parse are together much smaller than the initial dataset and that makes the BWT computation less resource-intensive.

Our Contributions. In this paper, we formally define and present prefix-free parsing. The main idea of this method is to divide the input text into overlapping variable-length phrases with delimiting prefixes and suffixes. To accomplish this division, we slide a window of length w over the text and, whenever the Karp-Rabin hash of the window is 0 modulo p, we terminate the current phrase at the end of the window and start the next one at the beginning of the window. This concept is partly inspired by `rsync`'s [1] use of a rolling hash for content-slicing. Here, w and p are parameters that affect the size of the dictionary of distinct phrases and the number of phrases in the parse. This takes linear-time and one pass over the text, or it can be sped up by running several windows in different positions over the text in parallel and then merging the results.

Just as `rsync` can usually recognize when most of a file remains the same, we expect that for most genomic databases and good choices of w and p, the total length of the phrases in the dictionary and the number of phrases in the parse will be small in comparison to the uncompressed size of the database. We demonstrate experimentally that with prefix-free parsing we can compute BWT using less memory and equivalent time. In particular, using our method we reduce peak memory usage up to 10x over a standard baseline algorithm which computes the BWT by first computing the suffix array using the algorithm SACA-K [16], while requiring roughly the same time on large sets of salmonella genomes obtained from GenomeTrakr.

In Section 3, we show how we can compute the BWT of the text from the dictionary and the parse alone using workspace proportional only to their total size, and time linear in the uncompressed size of the text when we can work in internal memory. In Section 4 we describe our implementation and report the results of our experiments showing that in practice the dictionary and parse often are significantly smaller than the text and so may fit in a reasonable internal memory even when the text is very large, and that this often makes the overall BWT computation both faster and smaller. We conclude in Section 5 and discuss directions for future work. Prefix-free parsing and all accompanied documents are available at `https://gitlab.com/manzai/Big-BWT`.

2 Review of the Burrows-Wheeler Transform

As part of the Human Genome Project, researchers had to piece together a huge number of relatively tiny, overlapping pieces of DNA, called reads, to assemble a reference genome about which they had little prior knowledge. Once the Project was completed, however, they could then use that reference genome as a guide to assemble other human genomes much more easily. To do this, they indexed the reference genome such that, after running a DNA sample from a new person through a sequencing machine and obtaining another collection of reads, for each of those new reads they could quickly determine which part of the reference genome it matched most closely. Since any two humans are genetically very similar, aligning the new reads against the reference genome gives a good idea of how they are really laid out in the person's genome.

In practice, the best solutions to this problem of indexed approximate matching work by reducing it to a problem of indexed exact matching, which we can formalize as follows: given a string T (which can be the concatenation of a collection of strings, terminated by

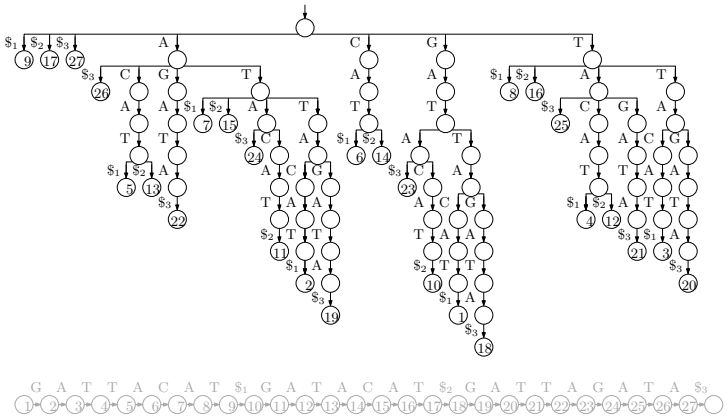


Figure 1 The suffix trie for our example with the three strings GATTACAT, GATACAT and GATTAGATA. The input is shown at the bottom, in grey because we do not need to store it.

special symbols), pre-process it such that later, given a pattern P, we can quickly list all the locations where P occurs in T. We now start with a simple but impractical solution to the latter problem, and then refine it until we arrive at a fair approximation of the basis of most modern assemblers, illustrating the workings of the Burrows-Wheeler Transform (BWT) along the way.

Suppose we want to index the three strings GATTACAT, GATACAT and GATTAGATA, so $T[1..n] = \text{GATTACAT}\$_1\text{GATACAT}\$_2\text{GATTAGATA}\$_3$, where $\$_1$, $\$_2$ and $\$_3$ are terminator symbols. Perhaps the simplest solution to the problem of indexing T is to build a trie of the suffixes of the three strings in our collection (i.e., an edge-labelled tree whose root-to-leaf paths are the suffixes of those strings) with each leaf storing the starting position of the suffix labelling the path to that leaf, as shown in Figure 1.

Suppose every node stores pointers to its children and its leftmost and rightmost leaf descendants, and every leaf stores a pointer to the next leaf to its right. Then given $P[1..m]$, we can start at the root and descend along a path (if there is one) such that the to the node at depth i is $P[i]$, until we reach a node v at depth $m+1$. We then traverse the leaves in v's subtree, reporting the the starting positions stored at them, by following the pointer from v to its leftmost leaf descendant and then following the pointer from each leaf to the next leaf to its right until we reach v's rightmost leaf descendant.

The trie of the suffixes can have a quadratic number of nodes, so it is impractical for large strings. If we remove nodes with exactly one child (concatenating the edge-labels above and below them), however, then there are only linearly many nodes, and each edge-label is a substring of the input and can be represented in constant space if we have the input stored as well. The resulting structure is essentially a suffix tree (although it lacks suffix and Weiner links), as shown in Figure 2. Notice that the label of the path leading to a node v is the longest common prefix of the suffixes starting at the positions stored at v's leftmost and rightmost leaf descendants, so we can navigate in the suffix tree, using only the pointers we already have and access to the input.

Although linear, the suffix tree still takes up an impractical amount of space, using several bytes for each character of the input. This is significantly reduced if we discard the shape of the tree, keeping only the input and the starting positions in an array, which is called the

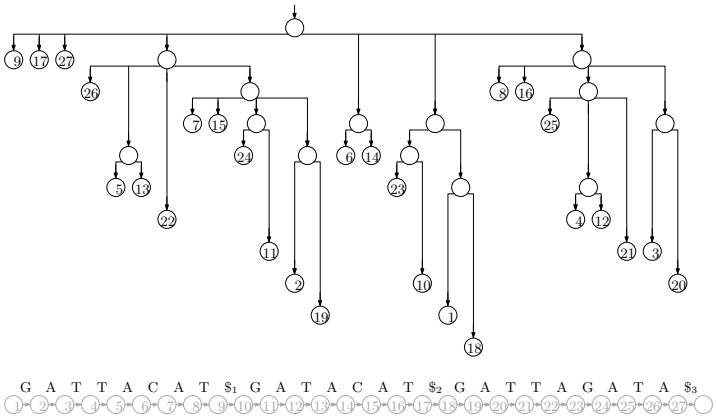


Figure 2 The suffix tree for our example. We now store the input.

suffix array (SA). The SA for our example is shown in Figure 3. Since the entries of the SA are the starting points of the suffixes in lexicographic order, with access to T we can perform two binary searchs to find the endpoints of the interval of the suffix array containing the starting points of suffixes starting with P: at each step, we consider an entry $\mathrm{SA}[i]$ and check if $T[\mathrm{SA}[i]]$ lexicographically precedes P. This takes a total of $O(m \log n)$ time done naïvely, and can be sped up with more sophisticated searching and relatively small auxiliary data structures.

Even the SA takes linear space, however, which is significantly more than what is needed to store the input when the alphabet is small (as it is in the case of DNA). Let Ψ be the function that, given the position of a value $i < n$ in the SA, returns the position of $i + 1$. Notice that, if we write down the first character of each suffix in the order they appear in the SA, the result is a sorted list of the characters in T, which can be stored using using $O(\log n)$ bits for each character in the alphabet. Once we have this list stored, given a position i in SA, we can return $T[\mathrm{SA}[i]]$ efficiently.

Given a position i in SA and a way to evaluate Ψ, we can extract $T[SA[i]..n]$ by writing $T[\mathrm{SA}[i]], T[\mathrm{SA}[\Psi(i)]], T[\mathrm{SA}[\Psi^2(i)]], \ldots$. Therefore, we can perform the same kind of binary search we use when with access to a full suffix array. Notice that if $T[\mathrm{SA}[i]] \prec T[\mathrm{SA}[i+1]]$ then $\Psi(i) < \Psi(i+1)$, meaning that $\Psi(1), \ldots, \Psi(n)$ can be divided into σ increasing consecutive subsequences, where σ is the size of the alphabet. It follows that we can store $nH_0(T) + o(n \log \sigma)$ bits, where $H_0(T)$ is the 0th-order empirical entropy of T, such that we can quickly evaluate Ψ. This bound can be improved with a more careful analysis.

Now suppose that instead of a way to evaluate Ψ, we have a way to evaluate quickly its inverse, which is called the last-to-first (LF) mapping. (This name was not chose because, if we start with the position of n in the suffix array and repeatedly apply the LF mapping we enumerate the positions in the SA in decreasing order of their contents, ending with 1; to some extent, the name is a lucky coincidence.) The LF mapping for our example is also shown with arrows in Figure 3. Since it is the inverse of Ψ, the sequence $\mathrm{LF}(1), \ldots, \mathrm{LF}(n)$ can be partitioned into σ incrementing subsequences: for each character c in the alphabet, if the starting positions of suffixes preceded by copies of c are stored in $\mathrm{SA}[j_1], \ldots, \mathrm{SA}[j_t]$ (appearing in that order in the SA), then $\mathrm{LF}(j_1)$ is 1 greater than the number of characters

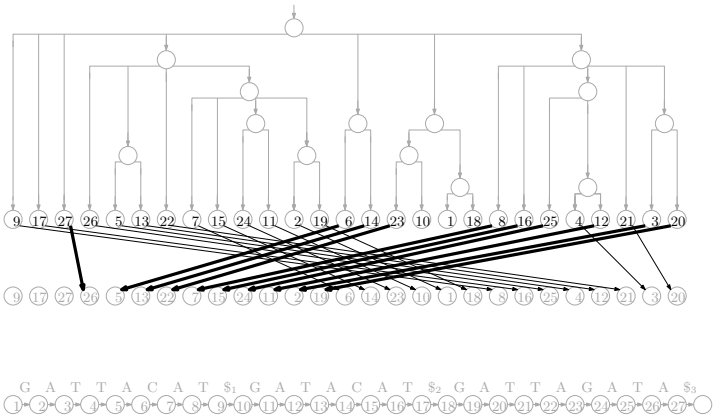


Figure 3 The suffix array for our example is the sequence of values stored in the leaves of the tree (which we need not store explicitly). The LF mapping is shown as the arrows between two copies of the suffix array; the arrows to values i such that $T[\mathrm{SA}[i]] = \mathrm{A}$ are heavy, to illustrate that they point to consecutive positions in the suffix array and do not cross. Since Ψ is the inverse of the LF mapping, it can be obtained by simply reversing the direction of the arrows.

lexicographically less than c in T and $\mathrm{LF}(j_2), \ldots, \mathrm{LF}(j_t)$ are the next $t-1$ numbers. Figure 3 illustrates this, with the arrows to values i such that $T[\mathrm{SA}[i]] = \mathrm{A}$ heavy, to illustrate that they point to consecutive positions in the suffix array and do not cross.

Consider the interval $I_{P[i..m]}$ of the SA containing the starting positions of suffixes beginning with $P[i..m]$, and the interval $I_{P[i-1]}$ containing the starting positions of suffixes beginning with $P[i-1]$. If we apply the LF mapping to the SA positions in $I_{P[i..m]}$, the SA positions we obtain that lie in $I_{P[i-1]}$ for a consecutive subinterval, containing the starting positions in T of suffixes beginning with $P[i-1..m]$. Therefore, we can search also with the LF mapping.

If we write the character preceding each suffix of T (considering it to be cyclic) in the lexicographic order of the suffixes, the result is the Burrows-Wheeler Transform (BWT) of T. A rank data structure over the BWT (which, given a character and a position, returns the number of occurrences of that character up to that position) can be used to implement searching with the LF-mapping, together with an array C indicating for each character in the alphabet how many characters in T are lexicographically strictly smaller than it. Specifically,

$$\mathrm{LF}(i) = \mathrm{BWT.rank}_{\mathrm{BWT}[i]}(i) + C[\mathrm{BWT}[i]]\,.$$

If follows that, to compute $I_{P[i-1..m]}$ from $I_{P[i..m]}$, we perform a rank query for $P[i-1]$ immediately before the beginning of $I_{P[i..m]}$ and add $C[P[i+1]]+1$ to the result, to find the beginning of $I_{P[i-1..m]}$; and we perform a rank query for $P[i-1]$ at the end of $I_{P[i..m]}$ and add $C[P[i+1]]$ to the result, to find the end of $I_{P[i-1..m]}$. Figure 4 shows the BWT for our example, and the sorted list of characters in T. Comparing it to Figure 3 makes the formula above clear: if $\mathrm{BWT}[i]$ is the jth occurrence of that character in the BWT, then the arrow from $\mathrm{LF}(i)$ leads from i to the position of the jth occurrence of that character in the sorted list. This is the main idea behind FM-indexes [8], and the main motivation for bioinformaticians to be interested in building BWTs.

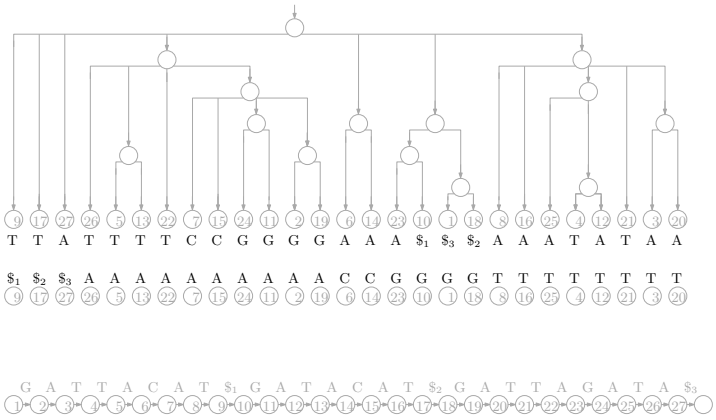


Figure 4 The BWT and the sorted list of characters for our example. Drawing arrows between corresponding occurrences of characters in the two strings gives us the diagram for the LF-mapping.

3 Theory

We let $E \subseteq \Sigma^w$ be any set of strings each of length $w \geq 1$ over the alphabet Σ and let $E' = E \cup \{\#, \$^w\}$, where $\#$ and $\$$ are special symbols lexicographically less than any in Σ. We consider a text $T[0..n-1]$ over Σ and let D be the maximum set such that for $d \in D$,

- d is a substring of $\#\,T\,\$^w$,
- exactly one proper prefix of d is in E',
- exactly one proper suffix of d is in E',
- no other substring of d is in E'.

We let S be the set of suffixes of length greater than w of elements of D.

Given T and a way to recognize strings in E, we can build D iteratively by simulating scanning $\#\,T\,\$^w$ to find occurrences of elements of E', adding to D each substring of $\#\,T\,\$^w$ that starts at the beginning of one such occurrence and ends at the end of the next one. While we are building D we also build a list P of the occurrences of the elements of D in T, which we call the parse (although each consecutive pair of elements overlap by w characters, so P is not a partition of the characters of $\#\,T\,\$^w$). We then build S from D and sort it.

For example, suppose we have $\Sigma = \{!, \mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, $w = 2$, $E = \{\mathtt{AC}, \mathtt{AG}, \mathtt{T!}\}$ and

$$T = \mathtt{GATTACAT!GATACAT!GATTAGATA}\,.$$

Then it follows that we get

$$\begin{aligned} D &= \{\#\mathtt{GATTAC}, \mathtt{ACAT!}, \mathtt{AGATA\$\$}, \mathtt{T!GATAC}, \mathtt{T!GATTAG}\}\,, \\ S &= \{\#\mathtt{GATTAC}, \mathtt{GATTAC}, \ldots, \mathtt{TAC}, \\ &\quad\ \mathtt{ACAT!}, \mathtt{CAT!}, \mathtt{AT!}, \\ &\quad\ \mathtt{AGATA\$\$}, \mathtt{GATA\$\$}, \ldots, \mathtt{A\$\$}, \\ &\quad\ \mathtt{T!GATAC}, \mathtt{!GATAC}, \ldots, \mathtt{TAC}, \\ &\quad\ \mathtt{T!GATTAG}, \mathtt{!GATTAG}, \ldots, \mathtt{TAG}\} \end{aligned}$$

and, identifying elements of D by their lexicographic ranks, $P = 0, 1, 3, 1, 4, 2$.

▶ **Lemma 1.** *S is a prefix-free set.*

Proof. If $s \in S$ were a proper prefix of $s' \in S$ then, since $|s| > w$, the last w characters of s – which are an element of E' – would be a substring of s' but neither a proper prefix nor a proper suffix of s'. Therefore, any element of D with s' as a suffix would contain at least three substrings in E', contrary to the definition of D. ◀

▶ **Lemma 2.** *Suppose $s, s' \in S$ and $s \prec s'$. Then $sx \prec s'x'$ for any strings $x, x' \in (\Sigma \cup \{\#, \$\})^*$.*

Proof. By Lemma 1, s and s' are not proper prefixes of each other. Since they are not equal either (because $s \prec s'$), it follows that sx and $s'x'$ differ on one of their first $\min(|s|, |s'|)$ characters. Therefore, $s \prec s'$ implies $sx \prec s'x'$. ◀

▶ **Lemma 3.** *For any suffix x of $\#\,T\,\$^w$ with $|x| > w$, exactly one prefix s of x is in S.*

Proof. Consider the substring d stretching from the beginning of the last occurrence of an element of E' that starts before or at the starting position of x, to the end of the first occurrence of an element of E' that starts strictly after the starting position of x. Regardless of whether d starts with $\#$ or another element of E', it is prefixed by exactly one element of E'; similarly, it is suffixed by exactly one element of E'. It follows that d is an element of D. Let s be the prefix of x that ends at the end of that occurrence of d in $\#\,T\,\$^w$, so $|s| > w$ and is a suffix of an element of D and thus $s \in S$. By Lemma 1, no other prefix of x is in S. ◀

Let f be the function that maps each suffix x of $\#\,T\,\$^w$ with $|x| > w$ to the unique prefix s of x with $s \in S$.

▶ **Lemma 4.** *Let x and x' be suffixes of $\#\,T\,\$^w$ with $|x|, |x'| > w$. Then $f(x) \prec f(x')$ implies $x \prec x'$.*

Proof. By the definition of f, $f(x)$ and $f(x')$ are prefixes of x and x' with $|f(x)|, |f(x')| > w$. Therefore, $f(x) \prec f(x')$ implies $x \prec x'$ by Lemma 2. ◀

Define $T'[0..n] = T\,\$$. Let g be the function that maps each suffix y of T' to the unique suffix x of $\#\,T\,\$^w$ that starts with y, except that it maps $T'[n] = \$$ to $\#\,T\,\$^w$. Notice that $g(y)$ always has length greater than w, so it can be given as an argument to f.

▶ **Lemma 5.** *The permutation that lexicographically sorts $T[0..n{-}1]\,\$^w, \ldots, T[n{-}1]\,\$^w, \#\,T\,\$^w$ also lexicographically sorts $T'[0..n], \ldots, T'[n-1..n], T'[n]$.*

Proof. Appending copies of $\$$ to the suffixes of T' does not change their relative order, and just as $\#\,T\,\$^w$ is the lexicographically smallest of $T[0..n-1]\,\$^w, \ldots, T[n-1]\,\$^w, \#\,T\,\$^w$, so $T'[n] = \$$ is the lexicographically smallest of $T'[0..n], \ldots, T'[n-1..n], T'[n]$. ◀

Let β be the function that, for $i < n$, maps $T'[i]$ to the lexicographic rank of $f(g(T'[i+1..n]))$ in S, and maps $T[n]$ to the lexicographic rank of $f(g(T')) = f(T\,\$^w)$.

▶ **Lemma 6.** *Suppose β maps k copies of a to $s \in S$ and maps no other characters to s, and maps a total of t characters to elements of S lexicographically less than s. Then the $(t+1)$st through $(t+k)$th characters of the BWT of T' are copies of a.*

Proof. By Lemmas 4 and 5, if $f(g(y)) \prec f(g(y'))$ then $y \prec y'$. Therefore, β partially sorts the characters in T' into their order in the BWT of T'; equivalently, the characters' partial order according to β can be extended to their total order in the BWT. Since every total extension of β puts those k copies of a in the $(t+1)$st through $(t+k)$th positions, they appear there in the BWT. ◀

From D and P, we can compute how often each element $s \in S$ is preceded by each distinct character a in $\#T\$^w$ or, equivalently, how many copies of a are mapped by β to the lexicographic rank of s. If an element $s \in S$ is a suffix of only one element $d \in D$ and a proper suffix of that – which we can determine first from D alone – then β maps only copies of the the preceding character of d to the rank of s, and we can compute their positions in the BWT of T'. If $s = d$ or a suffix of several elements of D, however, then β can map several distinct characters to the rank of s. To deal with these cases, we can also compute which elements of D contain which characters mapped to the rank of s. We will explain in a moment how we use this information.

For our example, $T =$ `GATTACAT!GATACAT!GATTAGATA`, we compute the information shown in Table 1. To ease the comparison to the standard computation of the BWT of $T'\$$, shown in Table 2, we write the characters mapped to each element $s \in S$ before s itself.

By Lemma 6, from the characters mapped to each rank by β and the partial sums of frequencies with which β maps characters to the ranks, we can compute the subsequence of the BWT of T' that contains all the characters β maps to elements of S, which are not complete elements of D and to which only one distinct character is mapped. We can also leave placeholders where appropriate for the characters β maps to elements of S, which are complete elements of D or to which more than one distinct character is mapped. For our example, this subsequence is `ATTTTTTCCGGGGAAA!$!AAA--TAA`. Notice we do not need all the infomation in P to compute this subsequence, only D and the frequencies of its elements in P.

Suppose $s \in S$ is an entire element of D or a suffix of several elements of D, and occurrences of s are preceded by several distinct characters in $\#T\$^w$, so β assigns s's lexicographic rank in S to several distinct characters. To deal with such cases, we can sort the suffixes of the parse P and apply the following lemma.

▶ **Lemma 7.** *Consider two suffixes t and t' of $\#T\$^w$ starting with occurrences of $s \in S$, and let q and q' be the suffixes of P encoding the last w characters of those occurrences of s and the remainders of t and t'. If $t \prec t'$ then $q \prec q'$.*

Proof. Since s occurs at least twice in $\#T\$^w$, it cannot end with $\$^w$ and thus cannot be a suffix of $\#T\$^w$. Therefore, there is a first character on which t and t' differ. Since the elements of D are represented in the parse by their lexicographic ranks, that character forces $q \prec q'$. ◀

We consider the occurrences in P of the elements of D suffixed by s, and sort the characters preceding those occurrences of s into the lexicographic order of the remaining suffixes of P which, by Lemma 7, is their order in the BWT of T'. In our example, `TAC` $\in S$ is preceded in $\#T\$\$$ by a `T` when it occurs as a suffix of `#GATTAC` $\in D$, which has rank 0 in D, and by an `A` when it occurs as a suffix of `T!GATAC` $\in D$, which has rank 3 in D. Since the suffix following 0 in $P = 0, 1, 3, 1, 4, 2$ is lexicographically smaller than the suffix following 3, that `T` precedes that `A` in the BWT.

Since we need only D and the frequencies of its elements in P to apply Lemma 6 to build and store the subsequence of the BWT of T' that contains all the characters β maps to elements of S, to which only one distinct character is mapped, this takes space proportional to the total length of the elements of D. We can then apply Lemma 7 to build the subsequence of missing characters in the order they appear in the BWT. Although this subsequence of missing characters could take more space than D and P combined, as we generate them we can interleave them with the first subsequence and output them, thus still using workspace proportional to the total length of P and the elements of D and only one pass over the space used to store the BWT.

Table 1 The information we compute for our example, $T =$ `GATTACAT!GATACAT!GATTAGATA`. Each line shows the lexicographic rank r of an element $s \in S$; the characters mapped to r by β; s itself; the elements of D from which the mapped characters originate; the total frequency with which characters are mapped to r; and the preceding partial sum of the frequencies.

rank	mapped characters	suffix	sources	frequency	preceding partial sum
0	A	#GATTAC	1	1	0
1	T	!GATAC	2	1	1
2	T	!GATTAG	3	1	2
3	T	A$$	5	1	3
4	T	ACAT!	4	2	4
5	T	AGATA$$	5	1	6
6	C	AT!	4	2	7
7	G	ATA$$	5	1	9
8	G	ATAC	2	1	10
9	G	ATTAC	1	1	11
10	G	ATTAG	3	1	12
11	A	CAT#	4	2	13
12	A	GATA$$	5	1	15
13	!	GATAC	2	1	16
14	$	GATTAC	1	1	17
15	!	GATTAG	3	1	18
16	A	T!GATAC	2	1	19
17	A	T!GATTAG	3	1	20
18	A	TA$$	5	1	21
19	T, A	TAC	1; 2	2	22
20	T	TAG	3	1	24
21	A	TTAC	1	1	25
22	A	TTAG	3	1	26

If we want, we can build the first subsequence from D and the frequencies of its elements in P; store it in external memory; and make a pass over it while we generate the second one from D and P, inserting the missing characters in the appropriate places. This way we use two passes over the space used to store the BWT, but we may use significantly less workspace.

Summarizing, assuming we can recognize the strings in E quickly, we can quickly compute D and P with one scan over T and then from them, with Lemmas 6 and 7, we can compute the BWT of $T' = T\,\$$ by sorting the suffixes of the elements of D and the suffixes of P. Since there are linear-time and linear-space algorithms for sorting suffixes when working in internal memory, this implies our main theoretical result:

▶ **Theorem 8.** *We can compute the BWT of $T\,\$$ from D and P using workspace proportional to sum of the total length of P and the elements of D, and $O(n)$ time when we can work in internal memory.*

Table 2 The BWT for $T' =$ `GATTACAT!GATACAT!GATTAGATA$`. Each line shows a position in the BWT; the character in that position; and the suffix immediately following that character in T'.

i	BWT[i]	suffix
0	A	$
1	T	!GATACAT!GATTAGATA$
2	T	!GATTAGATA$
3	T	A$
4	T	ACAT!GATACAT!GATTAGATA$
5	T	ACAT!GATTAGATA$
6	T	AGATA$
7	C	AT!GATACAT!GATTAGATA$
8	C	AT!GATTAGATA$
9	G	ATA$
10	G	ATACAT!GATTAGATA$
11	G	ATTACAT!GATACAT!GATTAGATA$
12	G	ATTAGATA$
13	A	CAT!GATACAT!GATTAGATA$
14	A	CAT!GATTAGATA$
15	A	GATA$
16	!	GATACAT!GATTAGATA$
17	$	GATTACAT!GATACAT!GATTAGATA$
18	!	GATTAGATA$
19	A	T!GATACAT!GATTAGATA$
20	A	T!GATTAGATA$
21	A	TA$
22	T	TACAT!GATACAT!GATTAGATA$
23	A	TACAT!GATTAGATA$
24	T	TAGATA$
25	A	TTACAT!GATACAT!GATTAGATA$
26	A	TTAGATA$

4 Practice

We have implemented our BWT construction in order to test our conjectures that, first, for most genomic databases and good choices of w and p, the total length of the phrases in the dictionary and the number of phrases in the parse will both be small in comparison to the uncompressed size of the database; second, computing the dictionary and the parse first and then computing the BWT from them leads to an overall speedup and reduction in memory usage. In this section we describe our implementation and then report our experimental results.

4.1 Implementation

As described in Sections 1 and 3, we slide a window of length w over the text, keeping track of the Karp-Rabin hash of the window; we also keep track of the hash of the entire prefix of the current phrase that we have processed so far. Whenever the hash of the window is

0 modulo p, we terminate the current phrase at the end of the window and start the next one at the beginning of the window. We prepend a NULL character to the first phrase and append w copies of NULL to the last phrase. If the text ends with w characters whose hash is 0 modulo p, then we take those w character to be the beginning of the last phrase and append to them w copies of NULL. We note that we prepend and append copies of the same NULL character; although using different characters simplifies the proofs in Section 3, it is not essential in practice.

We keep track of the set of hashes of the distinct phrases in the dictionary so far, as well as the phrases' frequencies. Whenever we terminate a phrase, we check if its hash is in that set. If not, we add the phrase to the dictionary and its hash to the set, and set its frequency to 1; if so, we compare the current phrase to the one in the dictionary with the same hash to ensure they are equal, then increment its frequency. (Using a 64-bit hash the probability of there being a collision is very low, so we have not implemented a recovery mechanism if one occurs.) In both cases, we write the hash to disk.

When the parsing is complete, we have generated the dictionary D and the parsing $P = p_1, p_2, \ldots, p_z$, where each phrase p_i is represented by its hash. Next, we sort the dictionary and make a pass over P to substitute the phrases' lexicographic ranks for their hashes. This gives us the final parse, still on disk, with each entry stored as a 4-byte integer. We write the dictionary to disk phrase by phrase in lexicographic order with a special end-of-phrase terminator at the end of each phrase. In a separate file we store the frequency of each phrase in as a 4-byte integer. Using four bytes for each integer does not give us the best compression possible, but it makes it easy to process the frequency and parse files later. Finally, we write to a separate file the array W of length $|P|$ such that $W[j]$ is the character of p_j in position $w + 1$ from the end (recall each phrase has length greater than w). These characters will be used to handle the elements of S that are also elements of D.

Next, we compute the BWT of the parsing P, with each phrase represented by its 4-byte lexicographic rank in D. The computation is done using the SACA-K suffix array construction algorithm [16] which, among the linear time algorithms, is the one using the smallest workspace. Instead of storing $BWT(P) = b_1, b_2, \ldots, b_z$, we save the same information in a format more suitable for the next phase. We consider the dictionary words in lexicographic order, and, for each word d_i, we write the list of BWT positions where d_i appears. We call this the inverted list for word d_i. Since the size of the inverted list of each word is equal to its frequency, which is available separately, we write to file the plain concatenation of the inverted lists using again four bytes per entry, for a total of $4|P|$ bytes. In this phase we also permute the elements of W so that now $W[j]$ is the character coming from the phrase that precedes b_j in the parsing, i.e. $P[SA[j] - 2]$.

In the final phase of the algorithm we compute the BWT of the input T. We deviate slightly from the description in Section 3 in that instead of lexicographically sorting the suffixes in D larger than w we sort all of them and later ignore those which are of length $\leq w$. The sorting is done applying the gSACAK algorithm [14] which computes the SA and LCP array for the set of dictionary phrases. We then proceed as in Section 3. If during the scanning of the sorted set S we meet s which is a proper suffix of several elements of D we use a heap to merge their respective inverted lists writing a character to the final BWT file every time we pop a position from the heap. If we meet s which coincides with a dictionary word d we write the characters retrieved from W from the positions obtained from d's inverted list.

Table 3 The dictionary and parse sizes for several files from the Pizza & Chili repetitive corpus, with three settings of the parameters w and p. All sizes are reported in megabytes; percentages are the sums of the sizes of the dictionaries and parses, divided by the sizes of the uncompressed files.

		$w = 6, p = 20$			$w = 8, p = 50$			$w = 10, p = 100$		
file	size	dict.	parse	%	dict.	parse	%	dict.	parse	%
`cere`	440	61	77	31	43	159	46	**89**	**17**	**24**
`cere_no_Ns`	409	33	77	27	**43**	**33**	**18**	60	17	19
`dna.001.1`	100	8	20	27	**13**	**9**	**21**	21	4	25
`einstein.en.txt`	446	2	87	20	3	39	9	**4**	**17**	**5**
`influenza`	148	16	28	30	**32**	**12**	**29**	49	6	37
`kernel`	247	14	52	26	14	20	13	**15**	**10**	**10**
`world_leaders`	45	5	5	21	**8**	**2**	**21**	11	1	26
`world_leaders_no_dots`	23	4	5	34	**6**	**2**	**31**	7	1	33

4.2 Experiments

In this section, the parsing and BWT computation are experimentally evaluated. All experiments were run on a server with Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz and 756 gigabytes of RAM.

Table 3 shows the sizes of the dictionaries and parses for several files from the Pizza & Chili repetitive corpus [2], with three settings of the parameters w and p. We note that `cere` contains long runs of `N`s and `world_leaders` contains long runs of periods, which can either cause many phrases, when the hash of w copies of those characters is 0 modulo p, or a single long phrase otherwise; we also display the sizes of the dictionaries and parses for those files with all `N`s and periods removed. The dictionaries and parses occupy between 5 and 31 percent of the space of the uncompressed files.

Table 4 shows the sizes of the dictionaries and parses for prefixes of a database of Salmonella genomes [20]. The dictionaries and parses occupy between 14 and 44 percent of the space of the uncompressed files, with the compression improving as the number of genomes increases. In particular, the dataset of ten thousand genomes takes nearly 50 GB uncompressed, but with $w = 10$ and $p = 100$ the dictionary and parse take only about 7 GB together, so they would still fit in the RAM of a commodity machine. This seems promising, and we hope the compression is even better for larger genomic databases.

Table 5 shows the runtime and peak memory usage for computing the BWT from the parsing for the database of Salmonella genomes. As a baseline for comparison, `simplebwt` computes the BWT by first computing the Suffix Array using algorithm SACA-K [16]; SACA-K is a linear time algorithm that uses $O(1)$ workspace and is fast in practice. As shown in Table 5, the peak memory usage of `simplebwt` is reduced by a factor of 4 to 10 by computing the BWT from the parsing; furthermore, the total runtime is competitive with `simplebwt`. In some instances, for example the databases of $5000, 10000$ genomes, computing the BWT from the parsing achieved significant runtime reduction over `simplebwt`; with $w = 10, p = 100$ on these instances, the runtime reduction is more than factors of $2, 4$, respectively. For our BWT computations, the peak memory usage with $w = 6$, $p = 20$ stays within a factor of roughly 2 of the original file size and is smaller than the orginal file size on the larger databases of 1000 genomes. For the database of 5000 genomes, the most expensive steps were parsing and computing the missing characters – about 23% of the total BWT – to fill in the subsequence.

Qualitatively similar results on files from the Pizza & Chili corpus are shown in Table 6.

Table 4 The dictionary and parse sizes for prefixes of a database of Salmonella genomes, with three settings of the parameters w and p. Again, all sizes are reported in megabytes; percentages are the sums of the sizes of the dictionaries and parses, divided by the sizes of the uncompressed files.

number of		$w = 6, p = 20$			$w = 8, p = 50$			$w = 10, p = 100$		
genomes	size	dict.	parse	%	dict.	parse	%	dict.	parse	%
50	249	68	43	44	**77**	**20**	**39**	91	10	40
100	485	83	85	35	**99**	**39**	**28**	122	19	29
500	2436	273	424	29	314	194	21	**377**	**96**	**19**
1000	4861	475	847	27	541	388	19	**643**	**192**	**17**
5000	24936	2663	4334	28	2915	1987	20	**3196**	**985**	**17**
10000	49420	4190	8611	26	4652	3939	17	**5176**	**1955**	**14**

Table 5 Time (seconds) and peak memory consumption (megabytes) of BWT calculations for prefixes of a database of Salmonella genomes, for three settings of the parameters w and p and for the comparison method `simplebwt`.

number of	$w = 6, p = 20$		$w = 8, p = 50$		$w = 10, p = 100$		`simplebwt`	
genomes	time	peak	time	peak	time	peak	time	peak
50	71	**545**	63	642	65	782	**53**	2247
100	118	**709**	**100**	837	102	1059	103	4368
500	570	**2519**	443	2742	**402**	3304	565	21923
1000	1155	**4517**	876	4789	**776**	5659	1377	43751
5000	7412	**42067**	5436	46040	**4808**	51848	11600	224423
10000	19152	**68434**	12298	74500	**10218**	84467	43657	444780

Table 6 Time (seconds) and peak memory consumption (megabytes) of BWT calculations on various files from the Pizza & Chili repetitive corpus, for three settings of the parameters w and p and for the comparison method `simplebwt`.

	$w = 6, p = 20$		$w = 8, p = 50$		$w = 10, p = 100$		`simplebwt`	
file	time	peak	time	peak	time	peak	time	peak
`cere`	90	603	79	**559**	**74**	801	90	3962
`einstein.en.txt`	53	196	40	88	**35**	**53**	97	4016
`influenza`	**27**	**166**	27	284	33	435	30	1331
`kernel`	43	170	29	**143**	**25**	144	50	2216
`world_leaders`	7	**50**	7	74	**7**	98	7	405

5 Conclusion and Future Work

We have described how prefix-free parsing can be used as preprocessing step to enable compression-aware computation of BWTs of large genomic databases. Our results demonstrate that the dictionaries and parses are often significantly smaller than the original input, and so may fit in a reasonable internal memory even when T is very large. Finally, we show how the BWT can be constructed from a dictionary and parse alone. We plan to investigate using compressed suffix arrays during the construction of the BWT, instead of suffix arrays, which should reduce our memory usage at the cost of increasing the running time by approximately a factor logarithmic in the size of the input; we will report the results in the full version of this paper.

In future extended versions of this work, we plan to explore its applications to sequence datasets that are terabytes in size; such as GenomeTrakr [19] and MetaSub [15]. We note that when downloading large datasets, prefix-free parsing can avoid storing the whole uncompressed dataset in memory or on disk. Suppose we run the parser on the dataset as it is downloaded, either as a stream or in chunks. We have to keep the dictionary in memory for parsing but we can write the parse to disk as we go, and in any case we can use less total space than the dataset itself. Ideally, the parsing could even be done server-side to reduce transmission time and/or bandwidth – which we leave for future implementation and experimentation.

A natural extension of our method is to consider efficient parallelization of the parsing. With k processors, we could divide the input string into k equal blocks, with each consecutive pair of blocks overlapping by w characters; we scan each block with a processor, to find the locations of the substrings of length w with Karp-Rabin hashes congruent to 0 modulo p; and then we scan the input with the k processors in parallel to compute the dictionary and parse, starting at roughly evenly-spaced locations of such substrings. Alternatively, our approach can be viewed as a modified Schindler Transform [3] and since previous authors [6] have shown how the Schindler Transform benefits from GPU parallelization, we believe that GPU-based parallelization could be both easy and effective.

Perhaps the main use of BWTs is in FM-indexes [8], which are at the heart of the most popular DNA aligners, including Bowtie [11, 10], BWA [12] and SOAP 2 [13]. With only rank support over a BWT, we can count how many occurrences of a given pattern there are in the text, but we cannot tell where they are without using a suffix-array sample. Until recently, suffix-array samples for massive, highly repetitive datasets were usually either much larger than the datasets BWTs, or very slow. Gagie, Navarro and Prezza [9] have now shown we need only store suffix array values at the ends of runs in the BWT, however, and we conjecture that we can build this sample while computing the BWT from the dictionary and the parse. Indeed, we were initially motivated to study new approaches to BWT construction because without them, Gagie et al.'s result may never realize its full potential.

References

1 rsync. URL: `https://rsync.samba.org`.

2 Repetitive corpus. URL: `http://pizzachili.dcc.uchile.cl/repcorpus.html`.

3 The sort transformation. URL: `http://www.compressconsult.com/st`.

4 Michael Burrows and David J. Wheeler. A block-sorting lossless compression algorithm. Technical report, Digital Equipment Corporation, 1994.

5 H.A. Carleton and P. Gerner-Smidt. Whole-genome sequencing is taking over foodborne disease surveillance. *Microbe*, 11:311–317, 2016.

6 Chia-Hua Chang, Min-Te Chou, Yi-Chung Wu, Ting-Wei Hong, Yun-Lung Li, Chia-Hsiang Yang, and Jui-Hung Hung. sBWT: memory efficient implementation of the hardware-acceleration-friendly Schindler transform for the fast biological sequence mapping. *Bioinformatics*, 32(22):3498–3500, 2016.

7 Paolo Ferragina, Travis Gagie, and Giovanni Manzini. Lightweight data indexing and compression in external memory. *Algorithmica*, 63(3):707–730, 2012.

8 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM (JACM)*, 52(4):552–581, 2005.

9 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in bwt-runs bounded space. In *Proceedings of the 29th Symposium on Discrete Algorithms (SODA)*, pages 1459–1477, 2018.

10 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–360, 2012. doi:10.1038/nmeth.1923.

11 Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

12 Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

13 Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

14 Felipe Alves Louza, Simon Gog, and Guilherme P. Telles. Inducing enhanced suffix arrays for string collections. *Theor. Comput. Sci.*, 678:22–39, 2017.

15 MetaSUB International Consortium. The Metagenomics and Metadesign of the Subways and Urban Biomes (MetaSUB) International Consortium inaugural meeting report. *Microbiome*, 4(1):24, 2016.

16 Ge Nong. Practical linear-time $O(1)$-workspace suffix sorting for constant alphabets. *ACM Trans. Inf. Syst.*, 31(3):15, 2013.

17 Alberto Policriti and Nicola Prezza. From LZ77 to the run-length encoded burrows-wheeler transform, and back. In *Proceedings of the 28th Symposium on Combinatorial Pattern Matching (CPM)*, pages 17:1–17:10, 2017.

18 Jouni Sirén. Burrows-Wheeler transform for terabases. In *Proccedings of the 2016 Data Compression Conference (DCC)*, pages 211–220, 2016.

19 E.L. Stevens, R. Timme, E.W. Brown, M.W. Allard, E. Strain, K. Bunning, and S. Musser. The public health impact of a publically available, environmental database of microbial genomes. *Frontiers in Microbiology*, 8:808, 2017.

20 Eric L. Stevens, Ruth Timme, Eric W. Brown, Marc W. Allard, Errol Strain, Kelly Bunning, and Steven Musser. The public health impact of a publically available, environmental database of microbial genomes. *Frontiers in Microbiology*, 8:808, 2017.

21 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015.

22 C. Turnbull et al. The 100,000 genomes project: bringing whole genome sequencing to the nhs. *British Medical Journal*, 361:k1687, 2018.

Detecting Mutations by eBWT

Nicola Prezza
Dipartimento di Informatica, University of Pisa, Italy
nicola.prezza@di.unipi.it
https://orcid.org/0000-0003-3553-4953

Nadia Pisanti
Dipartimento di Informatica, University of Pisa, Italy, and
ERABLE Team, INRIA, Lyon, France
pisanti@di.unipi.it
https://orcid.org/0000-0003-3915-7665

Marinella Sciortino
Dipartimento di Matematica e Informatica, University of Palermo, Italy
marinella.sciortino@unipa.it
https://orcid.org/0000-0001-6928-0168

Giovanna Rosone[1]
Dipartimento di Informatica, University of Pisa, Italy
giovanna.rosone@unipi.it
https://orcid.org/0000-0001-5075-1214

Abstract

In this paper we develop a theory describing how the extended Burrows-Wheeler Transform (eBWT) of a collection of DNA fragments tends to cluster together the copies of nucleotides sequenced from a genome G. Our theory accurately predicts how many copies of any nucleotide are expected inside each such cluster, and how an elegant and precise LCP array based procedure can locate these clusters in the eBWT.

Our findings are very general and can be applied to a wide range of different problems. In this paper, we consider the case of alignment-free and reference-free SNPs discovery in multiple collections of reads. We note that, in accordance with our theoretical results, SNPs are clustered in the eBWT of the reads collection, and we develop a tool finding SNPs with a simple scan of the eBWT and LCP arrays. Preliminary results show that our method requires much less coverage than state-of-the-art tools while drastically improving precision and sensitivity.

2012 ACM Subject Classification Applied computing → Bioinformatics, Mathematics of computing → Combinatorial algorithms

Keywords and phrases BWT, LCP Array, SNPs, Reference-free, Assembly-free

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.3

Supplement Material `https://github.com/nicolaprezza/eBWTclust`

Funding G. Rosone, N. Pisanti, M. Sciortino are partially and N. Prezza is supported by the project MIUR-SIR CMACBioSeq ("Combinatorial methods for analysis and compression of biological sequences") grant n. RBSI146R5L.

[1] Corresponding author

Acknowledgements We want to thank the anonymous reviewers and Mikaël Salson for their interesting and useful comments.

1 Introduction

The cheap and fast Next Generation Sequencing (NGS) technologies are producing huge amounts of data that put a growing pressure on data structures designed to store raw sequencing information, as well as on efficient analysis algorithms: collections of billion of DNA fragments (reads) need to be efficiently indexed for downstream analysis. After a sequencing experiment, the most traditional analysis pipeline begins with an error-prone and lossy mapping of the collection of reads onto a reference genome. Among the most widespread tools to align reads on a reference genome we can mention BWA [20], Bowtie2 [16], SOAP2 [21]. These methods share the use of the FM-index [10], an indexing machinery based on the Burrows-Wheeler Transform (BWT) [4]. Other approaches [13, 14] combine an index of the reference genome with the BWT of the reads collection in order to boost efficiency and accuracy. In some applications, however, aligning reads on a reference genome presents limitations mainly due to the difficulty of mapping highly repetitive regions, especially in the event of a low-quality reference genome (not to mention the cases in which the reference genome is not even available).

For this reason, indices of reads collections have also been suggested as a lossless dictionary of sequencing data, where sensitive analysis methods can be directly applied without mapping the reads to a reference genome (thus without needing one), nor assembling [29, 31, 17, 12]. In [7] the BWT, or more specifically its extension to string collections (named eBWT [27, 2]), is used to index reads from the 1000 Genomes Project [37] in order to support k-mer search queries. An eBWT-based compressed index of sets of reads has also been suggested as a basis for both RNA-Seq [6] and metagenomic [1] analyses. There exist also suffix array based data structures devised for indexing reads collections: the Gk array [30, 39] and the PgSA [15]. The latter does not have a fixed k-mer size. The tool SHREC [35] also uses a suffix-sorting-based index to detect and correct errors in sets of reads. The main observation behind the tool is that sequencing errors disrupt unary paths at deep levels of the reads' suffix trie. The authors provide a statistical analysis allowing to detect such branching points. Finally, there are several tools ([29, 31, 19, 3, 18, 17, 12]) that share the idea of using the de Bruijn graph (dBG) of the reads' k-mers. The advantages of dBG-based indices include allowing therein the characterization of several biologically-interesting features of the data as suitably shaped and sized *bubbles*[2] (e.g. SNPs, INDELs, Alternative Splicing events on RNA-Seq data, sequencing errors can all be modeled as bubbles in the dBG of sequencing data [29, 31, 19, 3, 18]). The drawback of these dBG representation, as well as those of suffix array based indices such as the ones introduced in [30, 39], is the lossy aspect of getting down to k-mers rather than representing the actual whole collection of reads. Also [13, 14] have this drawback as they index k-mers. An eBWT-based indexing method for reads collections, instead, has the advantages to be easy to compress and, at the same time, lossless: (e)BWT indexes support querying k-mers without the need to build different indexes for different values of k.

Here we introduce the *Positional Clustering* framework: an eBWT-based index of reads collections where we give statistical characterizations of (i) read suffixes prefixing the same genome's suffix as *clusters* in the eBWT, and (ii) the onset of these clusters by means of the

[2] A *bubble* in a graph is a pair of disjoint paths sharing the same source node and target node.

LCP. This clustering allows to locate and investigate, in a lossless index of reads collections, genome positions possibly equivalent to bubbles in the dBG [19, 29] *independently* from the k-mer length (a major drawback of dBG-based strategies). We thus gain the advantages of dBG-based indices while maintaining those of (e)BWT-based ones. Besides, the eBWT index also contains abundance data (useful to distinguish errors from variants, as well as distinct variant types) and does not need the demanding read-coherency check at post processing as no micro-assembly has been performed. With these promising advantages with respect to dBG-based strategies, the positional clustering framework allows likewise reference-free and assembly-free detection of SNPs [29, 28, 12], small INDELs [22, 12], sequencing errors [32, 33, 23], alternative splicing events [31], rearrangements breakpoints [19] on raw reads collections. To our knowledge, SHREC [35] and our positional clustering framework are the only attempts to characterize the statistical behavior of suffix trees of reads sets in presence of errors. We note that, while the two solutions are completely different from the algorithmic and statistical points of view, they are also, in some sense, complementary. SHREC characterizes errors as branching points at deep levels of the suffix trie; on the contrary, our positional framework characterizes clusters of read suffixes prefixing the same genome's suffix and identifies mutations (e.g. sequencing errors or SNPs) in the characters *preceding* those suffixes (i.e. eBWT characters). We note that our cluster characterization could be used to detect the suffix trie level from where sequencing errors are detected in SHREC. Similarly, SHREC's characterization of errors as branching points could be used in our framework to detect further mutations in addition to those in the eBWT clusters.

As a proof-of-concept, we test our theoretical framework with a prototype tool named eBWTclust designed to detect positional clusters and post-process them for assembly-free and reference-free SNPs detection directly on the eBWT of reads collection. Among several reference-free SNPs finding tools in the literature [29, 38, 28, 12], the state-of-the-art is represented by the well documented and maintained KisSNP and DiscoSnp suite [29, 38, 11], where DiscoSnp++ [28] is the latest and best performing tool. In order to validate the accuracy of positional clustering for finding SNPs, we compared DiscoSnp++ sensitivity and precision to those of our prototype eBWTclust. Preliminary results on human chromosomes show that, for example, using coverage 22x our tool is able to find 91% of all SNPs (vs 70% of DiscoSnp++) with an accuracy of 98% (vs 94% of DiscoSnp++).

2 Preliminaries

Let $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$ be a finite ordered alphabet with $c_1 < c_2 < \ldots < c_\sigma$, where $<$ denotes the standard lexicographic order. For $s \in \Sigma^*$, we denote its letters by $s[1], s[2], \ldots, s[n]$, where n is the *length* of s, denoted by $|s|$. We append to $s \in \Sigma^*$ an end-marker symbol \$ that satisfies $\$ < c_1$. Note that, for $1 \leq i \leq n$, $s[i] \in \Sigma$ and $s[n+1] = \$ \notin \Sigma$. A *substring* of s is denoted as $s[i, j] = s[i] \cdots s[j]$, with $s[1, j]$ being called a *prefix* and $s[i, n+1]$ a *suffix* of s.

We denote by $\mathcal{S} = \{R_1, R_2, \ldots, R_m\}$ a collection of m strings (reads), and by $\$_i$ the end-marker appended to R_i (for $1 \leq i \leq m$), with $\$_i < \$_j$ if $i < j$. Let us denote by P the sum of the lengths of all strings in $\mathcal{S}$. The *generalized suffix array* GSA of the collection $\mathcal{S}$ (see [36, 5, 25]) is an array containing P pairs of integers (r, j), corresponding to the lexicographically sorted suffixes $R_r[j, |R_r| + 1]$, where $1 \leq j \leq |R_r| + 1$ and $1 \leq r \leq m$. In particular, $\mathsf{gsa}(\mathcal{S})[i] = (r, j)$ (for $1 \leq i \leq P$) if the suffix $R_r[j, |R_r| + 1]$ is the i-th smallest suffix of the strings in $\mathcal{S}$. Such a notion is a natural extension of the suffix array of a string (see [26]). The Burrows-Wheeler Transform (BWT) [4], a well known text transformation largely used for data compression and self-indexing compressed data structure, has also been extended to a collection $\mathcal{S}$ of strings (see [27]). Such an extension, known as *extended*

Burrows-Wheeler Transform (eBWT) or multi-string BWT, is a reversible transformation that produces a string that is a permutation of the letters of all strings in $\mathcal{S}$: $\mathsf{ebwt}(\mathcal{S})$ is obtained by concatenating the symbols cyclically preceding each suffix in the list of lexicographically sorted suffixes of all strings in $\mathcal{S}$. The eBWT applied to $\mathcal{S}$ can also be defined in terms of the generalized suffix array of $\mathcal{S}$ ([2]): if $\mathsf{gsa}(\mathcal{S})[i] = (j, t)$ then $\mathsf{ebwt}(\mathcal{S})[i] = R_j[t-1]$; when $t = 1$, then $\mathsf{ebwt}(\mathcal{S})[i] = \$_j$. For gsa, ebwt, and lcp, the LF mapping (resp. FL) is a function that associates to each (e)BWT symbol the position preceding (resp. following) it on the text.

The *longest common prefix* (LCP) array of a collection $\mathcal{S}$ of strings (see [5, 25, 8]), denoted by $\mathsf{lcp}(\mathcal{S})$, is an array storing the length of the longest common prefixes between two consecutive suffixes of $\mathcal{S}$ in lexicographic order. For each $i = 2, \ldots, P$, if $\mathsf{gsa}(\mathcal{S})[i-1] = (p_1, p_2)$ and $\mathsf{gsa}(\mathcal{S})[i] = (q_1, q_2)$, $\mathsf{lcp}(\mathcal{S})[i]$ is the length of the longest common prefix of suffixes starting at positions p_2 and q_2 of the strings R_{p_1} and R_{q_1}, respectively. We set $\mathsf{lcp}(\mathcal{S})[1] = 0$.

For gsa, ebwt, and lcp, the set $\mathcal{S}$ will be omitted when clear from the context.

3 eBWT positional clustering

Let R be a read sequenced from a genome $G[1, n]$. We say that $R[j]$ is a *read-copy* of $G[i]$ iff $R[j]$ is copied from $G[i]$ during the sequencing process (and then possibly changed due to sequencing errors). Let us consider the eBWT of a set of reads $\{R_1, \ldots, R_m\}$ of length[3] r, sequenced from a genome G. Assuming that c is the *coverage* of $G[i]$, let us denote with $R_{i_1}[j_1], \ldots, R_{i_c}[j_c]$ the c read-copies of $G[i]$. Should not there be any sequencing error, if we consider k such that the genome fragment $G[i+1, i+k]$ occurs only once in G (that is, nowhere else than right after $G[i]$) and if r is large enough so that with high probability each $R_{i_t}[j_t]$ is followed by at least k nucleotides, then we observe that the c read copies of $G[i]$ would appear contiguously in the eBWT of the reads. We call this phenomenon eBWT *positional clustering*. Due to sequencing errors, and to the presence of repetitions with mutations in real genomes, a *clean* eBWT positional clustering is not realistic. However, in this section we show that, even in the event of sequencing errors, in the eBWT of a collection of reads sequenced from a genome G, the read-copies of $G[i]$ still *tend to be clustered* together according to a suitable Poisson distribution.

We make the following assumptions: (i) the sequencing process is uniform, i.e. the positions from where each read is sequenced are uniform and independent random variables, (ii) the probability ϵ that a base is subject to a sequencing error is a constant[4], (iii) a sequencing error changes a base to a different one uniformly (i.e. with probability 1/3 for each of the three possible variants), and (iv) the number m of reads is large (hence, in our theoretical analysis we can assume $m \to \infty$).

▶ **Definition 1** (eBWT cluster)**.** The eBWT *cluster of* i, with $1 \leq i \leq n$ being a position on G, is the substring $\mathsf{ebwt}[a, b]$ such that $\mathsf{gsa}[a, b]$ is the range of read suffixes prefixed by $G[i+1, i+k]$, where $k < r$ is the smallest value for which $G[i+1, i+k]$ appears only once in G. If no such value of k exists, we take $k = r - 1$ and say that the cluster is *ambiguous*.

If no value $k < r$ guarantees that $G[i+1, i+k]$ appears only once in G, then the eBWT cluster of i does not contain only read-copies of $G[i]$ but also those of other $t-1$ characters $G[i_2], \ldots, G[i_t]$. We call t the *multiplicity* of the eBWT cluster. Note that $t = 1$ for non-ambiguous clusters.

[3] For simplicity of exposition, here we assume that all the reads have the same length r. With little more effort, it can be shown that our results hold also when r is the *average* read length.

[4] We assume this to simplify the theoretical framework. In Section 4 we will see that our framework works even on data with sequencing simulated using realistic position-dependent errors distribution.

▶ **Theorem 2** (eBWT positional clustering). *Let $R_{i_1}[j_1], \ldots, R_{i_c}[j_c]$ be the c read-copies of $G[i]$. An expected number $X \leq c$ of these read copies will appear in the* eBWT *cluster* $\mathsf{ebwt}[a, b]$ *of i, where $X \sim Poi(\lambda)$ is a Poisson random variable with mean*

$$\lambda = m \cdot \frac{r-k}{n} (1-\epsilon)^k$$

and where k is defined as in Definition 1.

Proof. The probability that a read covers $G[i]$ is r/n. However, we are interested only in those reads such that, if $R[j]$ is a read-copy of $G[i]$, then the suffix $R[j+1, r+1]$ contains at least k nucleotides, i.e. $j \leq r-k$. In this way, the suffix $R[j+1, r+1]$ will appear in the GSA range $\mathsf{gsa}[a, b]$ of suffixes prefixed by $G[i+1, i+k]$ or, equivalently, $R[j]$ will appear in $\mathsf{ebwt}[a, b]$. The probability that a random read from the set is uniformly sampled from such a position is $(r-k)/n$. If the read contains a sequencing error inside $R[j+1, j+k]$, however, the suffix $R[j+1, r+1]$ will not appear in the GSA range $\mathsf{gsa}[a, b]$. The probability that this event does not happen is $(1-\epsilon)^k$. Since we assume that these events are independent, the probability of their intersection is therefore

$$Pr(R[j] \in \mathsf{ebwt}[a, b]) = \frac{r-k}{n} (1-\epsilon)^k$$

This is a Bernoullian event, and the number X of read-copies of $G[i]$ falling in $\mathsf{ebwt}[a, b]$ is the sum of m independent events of this kind. Then, X follows a Poisson distribution with mean $\lambda = m \cdot \frac{r-k}{n} (1-\epsilon)^k$. ◀

Theorem 2 states that, if there exists a value $k < r$ such that $G[i+1, i+k]$ appears only once in G (i.e. if the cluster of i is not ambiguous), then X of the $b-a+1$ letters in $\mathsf{ebwt}[a, b]$ are read-copies of $G[i]$. The remaining $(b-a+1)-X$ letters are noise introduced by suffixes that mistakenly end up inside $\mathsf{gsa}[a, b]$ due to sequencing errors. It is not hard to show that this noise is extremely small under the assumption that G is a uniform text; we are aware that this assumption is not realistic, but we will experimentally show in Section 4 that also on real genomes and simulated reads our approach produces extremely accurate results (as predicted by our simplified theoretical framework).

Note that the expected coverage of position $G[i]$ is also a Poisson random variable, with mean $\lambda' = \frac{mr}{n}$ equal to the average coverage. On expectation, the size of non-ambiguous eBWT clusters is thus $\lambda/\lambda' = \frac{(r-k)(1-\epsilon)^k}{r} < 1$ times the average coverage. E.g., with $k = 14$, $\epsilon = 0.0033$ (see [34, Table 1, HiSeq, R2]), and $r = 100$ the expected cluster size is $100 \cdot \lambda/\lambda' \approx 80\%$ the average coverage.

Finally, it is not hard to prove, following the proof of Theorem 2, that in the general case with multiplicity $t \geq 1$ the expected cluster size follows a Poisson distribution with mean $t \cdot \lambda$ (because the read-copies of t positions are clustered together). This observation will allow us to detect and discard ambiguous clusters using a significance test.

So far, we have demonstrated the eBWT positional clustering property but we don't have a way for identifying the eBWT clusters. A naive strategy could be to fix a value of k and define clusters to be ranges of k-mers in the GSA. This solution, however, fails to separate read suffixes differing after k positions (this is, indeed, a drawback of all k-mer-based strategies). The aim of Theorem 3 is precisely to fill this gap, allowing us to move from theory to practice. Intuitively, we show that clusters lie between local minima in the LCP array. This strategy automatically detects the value k satisfying Definition 1 in a data-driven approach.

Our result holds only if two conditions on the eBWT cluster under investigation are satisfied (see Proposition 4 for an analysis of the success probability). More specifically, we require that the eBWT cluster $\mathsf{ebwt}[a,b]$ of position i satisfies:

(1) The cluster does not have noise, i.e. $X = b - a + 1$, and

(2) Let $(p_1, j_1), (p_2, j_2) \in \mathsf{gsa}[a,b]$. For any x such that $k \leq x < r$, if both $R_{p_1}[j_1, r]$ and $R_{p_2}[j_2, r]$ contain their leftmost sequencing errors in $R_{p_1}[j_1 + x]$ and $R_{p_2}[j_2 + x]$, then $R_{p_1}[j_1 + x] \neq R_{p_2}[j_2 + x]$.

Proposition 4 will show that with probability high enough Condition (2) is satisfied in practice. E.g., with $r = 100$, $\epsilon = 0.0033$ (see [34, Table 1, HiSeq, R2]), mean coverage $\lambda' = 20$, and for any $k \geq 14$, Proposition 4 shows that the condition holds (in expectation) on at least 85% of the non-ambiguous clusters. Decreasing ϵ to 0.002 [34, Table 1, HiSeq, R1] this percentage increases to 93%. While the rough lower bound of Proposition 4 is quite sensitive with respect to the substitution rate ϵ, the sensitivity of $\approx 90\%$ we obtain in our experiments (see Section 4.4) suggests that on our test-cases at least 90% of the clusters satisfy Condition (2).

▶ **Theorem 3.** *Let* $\mathsf{ebwt}[a,b]$ *be the* eBWT *cluster of a position* i *meeting Conditions (1) and (2). Then, there exists a value* $a < p \leq b$ *such that* $\mathsf{lcp}[a+1, p]$ *is a non-decreasing sequence and* $\mathsf{lcp}[p+1, b]$ *is a non-increasing sequence.*

Proof. Let us denote by p_M the largest index in $(a,b]$ such that $\mathsf{lcp}[p_M] = M$, where M is the maximum value of LCP in $(a,b]$ (if M occurs multiple times, take the rightmost occurrence). We claim the theorem holds for $p = p_M$. Let us denote by q and j, $1 \leq q \leq m$, $1 \leq j \leq r$, the positive integers such that $\mathsf{gsa}[p_M] = (q, j)$. This means, by using Condition (2), that the read R_q contains the longest prefix $R_q[j, j + M - 1]$ without sequencing errors, $j + M \leq r + 1$. Consider any other suffix $R_u[j_u, r+1]$ in the range and let $R_u[j_u + x]$ be the leftmost mismatch letter in $R_u[j_u, r + 1]$, with $x \geq k$. Note that if $R_u[j_u, r + 1]$ does not contain sequencing errors, then $j_u + x - 1 = r$, otherwise $R_u[j_u + x]$ has been mutated with an error. We suppose that the mutation at position $j_u + x$ generated a letter lexicographically smaller than that of the genome (the other case is symmetric). By Condition (2), $x \leq M + 1$ and no other suffix $R_v[j_v, r + 1]$ in the range satisfies $R_v[j_v + x] = R_u[j_u + x]$. Then, $R_u[j_u, r + 1]$ falls right after a suffix $R_{u'}[j_{u'}, r + 1]$ such that either $R_{u'}[j_{u'}, r]$ properly prefixes $R_u[j_u, r + 1]$ or the leftmost mismatch occurs at position $x' \leq x$. Similarly, $R_u[j_u, r + 1]$ falls right before a suffix $R_{u''}[j_{u''}, r + 1]$ such that $R_u[j_u + x] < R_{u''}[j_{u''} + x]$ and whose leftmost mismatch position x'' satisfies $x \leq x'' \leq M + 1$. This shows that, before suffix $R_q[j, r + 1]$, other suffixes are ordered by increasing position of their leftmost mismatch letter (since $x' \leq x \leq x''$) and, then, by the lexicographic order among mismatch letters, which in particular implies that before suffix $R_q[j, r + 1]$ the lcp values are non-decreasing. Symmetrically, with a similar reasoning one can easily prove that after suffix $R_q[j, r + 1]$ the lcp values are non-increasing. ◀

▶ **Proposition 4.** *Given a* eBWT *cluster* $\mathsf{ebwt}[a,b]$ *with multiplicity* $t \geq 1$*, Condition (2) holds with probability at least*

$$CDF(t\lambda, \lceil t\lambda \rceil + \delta) \cdot \left(\sum_{e=0}^{3} \binom{\lceil t\lambda \rceil + \delta}{e} \cdot (1-\epsilon)^{\lceil t\lambda \rceil + \delta - e} \cdot \epsilon^e \cdot c_e \right)^{r-k}$$

for any integer $\delta \geq 0$ *and where:* k *is the value defined in Definition 1,* λ *is the mean of the Poisson distribution of Theorem 2,* $CDF(\mu, z)$ *is the cumulative distribution function of a Poisson random variable with mean* μ *evaluated in* z*, and* $c_0 = c_1 = 1$*,* $c_2 = 2/3$*,* $c_3 = 2/9$*.*

Proof. First note that, by Theorem 2, the number of suffixes in the cluster sharing their first k bases is at most $\lceil t\lambda \rceil + \delta$ with probability $CDF(t\lambda, \lceil t\lambda \rceil + \delta)$. We first analyze the probability that the condition holds for a fixed offset x (i.e. looking at the $(x+1)$-th base of all suffixes in the range sharing their first x characters), and then compute the joint probability for all $k \leq x < r$.

Note that the condition cannot fail if the number e of bases $R_{p_h}[j_h + x]$ that have been subject to errors is either $e = 0$ or $e = 1$. The condition does not fail also if we have $e = 2$ or $e = 3$ *distinct* errors, while it always fails for $e \geq 4$ (since at least two errors will produce the same base). On a generic number $Y \leq \lceil t\lambda \rceil + \delta$ of suffixes (those sharing their first x bases), one can verify that the probability that the condition does not fail for a fixed number $e < 4$ of errors is $\binom{Y}{e} \cdot (1-\epsilon)^{Y-e} \cdot \epsilon^e \cdot c_e$, where $c_0 = c_1 = 1$, $c_2 = 2/3$, $c_3 = 2/9$. Since these events are disjoint, we can sum these probabilities to get a lower bound to the probability that condition (2) holds on a specific offset x on Y suffixes. Since this probability decreases as Y increases, we get a lower bound by taking the maximum $Y = \lceil t\lambda \rceil + \delta$, and obtain probability $\sum_{e=0}^{3} \binom{\lceil t\lambda \rceil + \delta}{e} \cdot (1-\epsilon)^{\lceil t\lambda \rceil + \delta - e} \cdot \epsilon^e \cdot c_e$.

To get a lower bound to the probability that the condition holds *simultaneously* on all $k \leq x < r$, we take the product of these probabilities (note that errors at different offsets are independent events). This reasoning holds only if there are at most $\lceil t\lambda \rceil + \delta$ suffixes sharing their first k bases – which happens with probability $CDF(t\lambda, \lceil t\lambda \rceil + \delta)$ – so we must include this multiplicative correction factor to get our final lower bound. ◀

Note: to get a lower bound that is independent from δ in Proposition 4, use the δ that maximizes the expression (since the lower bound holds for any δ).

According to Theorem 3, clusters are delimited by local minima in the LCP array of the read set. This gives us a strategy for finding clusters that is *independent* from k. Importantly, the proof of Theorem 3 also gives us the suffix in the range (the p-th suffix) whose longest prefix without sequencing errors is maximized. This will be useful in the next section to efficiently compute a consensus of the reads in the cluster.

Observe that by applying Theorem 3 we also find ambiguous clusters. However, the expected length of these clusters is a multiple of λ, so they can be reliably discarded with a significance test based on the Poisson distribution of Theorem 2.

To conclude, we note that our analysis automatically adapts to the case where also indels are present in the reads (i.e. not just substitutions, but possibly also insertions and deletions). To see why this holds true, note that our analysis only looks at the *first* base that changes w.r.t. the reference in any read suffix. Since we do not look at the following bases, indels behave exactly like SNPs in Theorems 2 and 3. For the same reason, indels between the two individuals will produce clusters containing two distinct letters (i.e. we capture the last letter of the indel, which by definition differs from the corresponding letter in the reference). On the other hand, this shows that we will mistakenly classify indels as SNPs. The straightforward solution (i.e. performing a multiple alignment on the left-contexts in each cluster) will be discussed in a forthcoming extension of this paper.

4 Experimental Validation: Reference-Free SNPs Discovery

When the reads dataset contains variations (e.g. two allele of the same individual, or two or more distinct individuals, or different isoforms of the same gene in RNA-Seq data, or different reads covering the same genome fragment in a sequencing process, etc.), the eBWT positional clustering described in the previous section can be used to detect, directly from the raw reads (hence, without assembly and without the need of a reference genome), positions

$G[i]$ exhibiting possibly different values, but followed by the same context: they will be in a cluster delimited by LCP minima and containing possibly different letters (corresponding to the read copies of the variants of $G[i]$ in the read set). This general idea can be used in several applications: error correction, assembly (the strategy finds overlaps between reads, so it can be used for assembly purposes), haplotype discovery (if fed with reads from just one diploid sample, this strategy can find heterozygous sites), and so on.

In this section, with the purpose of experimentally validating the theoretical framework of Section 3, we describe a new alignment-free and reference-free method that, with a simple scan of the eBWT and LCP arrays, detects SNPs in read collections.

Since (averagely) half of the reads comes from the forward (F) strand, and half from the reverse-complement (RC) strand, we denote with the term *right* (resp. *left*) *breakpoint* those variants found in a cluster formed by reads coming from the F (resp. RC) strand, and therefore sharing the right (resp. left) context adjacent to the variant. A *non-isolated SNP* [38] is a variant at position i such that the closest variant is within k bases from i, for some fixed k (we use $k = 31$ in our validation procedure, see below). The SNP is *isolated* otherwise. Note that, while isolated SNPs are found twice with our method (one as a right breakpoint and one as a left breakpoint), this is not true for non-isolated SNPs: variants at the sides of a group of non-isolated SNPs are found as either left or right breakpoint, while SNPs *inside* the group will be found with positional clustering plus a partial local assembly of the reads in the cluster. In the next two subsections we give all the details of our strategy.

4.1 Pre-processing (eBWT computation)

Since we do not aim at finding matches between corresponding pairs of clusters on the forward and reverse strands, we augment the input adding the reverse-complement of the reads: for a reads set $\mathcal{S}$, we add $\mathcal{S}^{RC}$ as well. Hence, given two reads sets $\mathcal{S}$ and $\mathcal{T}$, in the pre-processing phase we compute $\mathsf{ebwt}(\mathcal{R})$, $\mathsf{lcp}(\mathcal{R})$, and $\mathsf{gsa}(\mathcal{R})$, for $\mathcal{R} = \{\mathcal{S} \cup \mathcal{S}^{RC} \cup \mathcal{T} \cup \mathcal{T}^{RC}\}$. This task can be achieved using, for example, BCR[5] [5], Egsa[6] [25] or gsacak[7] [24]. We also compute $\mathsf{gsa}(\mathcal{R})$ because we will need it (see Subsection 4.2) to extract left and right contexts of the SNP. Though this could be achieved by performing (in external memory) multiple steps of LF- and FL-mappings on the eBWT, this would significantly slow-down our tool. Note that our approach can also be generalized to more than two reads collections.

4.2 SNP calling

Our SNPs calling approach takes as input $\mathsf{ebwt}(\mathcal{R})$, $\mathsf{lcp}(\mathcal{R})$, and $\mathsf{gsa}(\mathcal{R})$ and outputs SNPs in KisSNP2 format [11]: a fasta file containing a pair of sequences per SNP (one per sample, containing the SNP and its context). The SNP calling is divided in two main steps.

Build clusters. First, we scan $\mathsf{ebwt}(\mathcal{R})$ and $\mathsf{lcp}(\mathcal{R})$, find clusters using Theorem 3, and store them to file as a sequence of ranges on the eBWT. In addition, while computing clusters we also apply a threshold of minimum LCP (by default, 16): we cut clusters' tails containing LCP values smaller than the threshold. This additional filtering drastically reduces the number of clusters saved to file (and hence memory usage and running time), since the original strategy would otherwise output many short clusters containing small LCP values corresponding to noise.

[5] `https://github.com/giovannarosone/BCR_LCP_GSA`
[6] `https://github.com/felipelouza/egsa`
[7] `https://github.com/felipelouza/sacak-lcp`

Call SNPs. The second step takes as input the clusters file, $\mathsf{ebwt}(\mathcal{R})$, $\mathsf{lcp}(\mathcal{R})$, $\mathsf{gsa}(\mathcal{R})$, and $\mathcal{R}$, and processes clusters from first to last as follows:

1. We test the cluster's length using the Poisson distribution predicted by Theorem 2; if the cluster's length falls in one of the two tails at the sides of the distribution (by default, the two tails summing up to 5% of the distribution), then the cluster is discarded; Moreover, due to k-mers that are not present in the genome but appear in the reads because of sequencing errors (which introduce noise around cluster length equal to 1), we also fix a minimum value of length for the clusters (by default, 4 letters per sample).
2. In the remaining clusters, we find the most frequent nucleotides b_1 and b_2 of samples 1 and 2, respectively, and check whether $b_1 \neq b_2$; if so, then we have a candidate SNP: for each sample, we use the GSA to retrieve the coordinate of the read containing the longest right-context without errors (see explanation after Proposition 4); moreover, we retrieve, and temporarily store in a buffer, the coordinates of the remaining reads in the cluster.
3. After processing all events, we scan the fasta file storing $\mathcal{R}$ to retrieve the reads of interest (those whose coordinates are in the buffer); for each one of them, we compute a partial assembly of the read prefixes preceding the SNP, for each of the two samples. This allows us to compute a left-context for each SNP (by default, of length 20), and it also represents a further validation step: if the assembly cannot be built because a consensus cannot be found, then the cluster is discarded. Note that these left-contexts preceding SNPs (which are actually right-contexts if the cluster is formed by reads from the RC strand) allow us to capture non-isolated SNPs.

Complexity. In the clustering step, we process the eBWT and LCP and on-the-fly output clusters to disk. The SNP-calling step performs one scan of the eBWT, GSA, and clusters file to detect interesting clusters, plus one additional scan of the read set to retrieve contexts surrounding SNPs. Both these phases take linear time in the size of the input and do not use disk space in addition to the input and output. Due to the fact that we store in a buffer the coordinates of reads inside interesting clusters, this step uses an amount of RAM proportional to the number of SNPs times the average cluster size λ times the read length r (e.g. a few hundred MB in our case study of Section 4.4). Notice that our method is very easy to parallelize, as the analysis of each cluster is independent from the others.

4.3 Validation

Here we describe the validation tool we designed to measure the sensitivity and precision of any tool outputting SNPs in KisSNP2 format. Note that we output SNPs as pairs of reads containing the actual SNPs plus their contexts (one sequence per sample). This can be formalized as follows: the output is a series of pairs of triples (we call them *calls*) (L', s', R'), (L'', s'', R'') where L', R', L'', R'' are the left/right contexts of the SNP in the two samples, and letters s', s'' are the actual variant. Given a .vcf file (Variant Call Format) containing the ground truth, the most precise way to validate this kind of output is to check that the triples actually match contexts surrounding true SNPs on the reference genome (used here just for accuracy validation purposes). That is, for each pair in the output calls:

1. If there is a SNP $s' \to s''$ in the .vcf that is surrounded in the first sample by contexts L', R' (or their RC), then (L', s', R'), (L'', s'', R'') is a true positive (TP).
2. Any pair (L', s', R'), (L'', s'', R'') that is not matched with any SNP in the ground truth (as described above) is a false positive (FP).
3. Any SNP in the ground truth that is not matched with any call is a false negative (FN).

We implemented the above validation strategy with a (quite standard) reduction of the problem to the 2D range reporting problem: we insert in a two-dimensional grid two points per SNP (from the .vcf) using as coordinates the ranks of its right and (reversed) left contexts among the sorted right and (reversed) left contexts of all SNPs (contexts from the first sample) on the F and RC strands. Given a pair (L', s', R'), (L'', s'', R''), we find the two-dimensional range corresponding to all SNPs in the ground truth whose right and (reversed) left contexts are prefixed by R' and (the reversed) L', respectively. If there is at least one point in the range matching the variation $s' \to s''$, then the call is a TP[8] (case 1 above; note: to be a TP, a SNP can be found either on the F or on the RC strand, or both); else, it is a FP (case 2 above). Finally, pairs of points (same SNP on the F/RC strands) that have not been found by any call are marked as FN (case 3 above). We repeat the procedure for any other SNP found between the two strings $L's'R'$ and $L''s''R''$ to find non-isolated SNPs.

4.4 Preliminary Experiments

In order to evaluate our method, we compare eBWTclust with DiscoSnp++, that is a revisiting of the DiscoSnp algorithm: while DiscoSnp detects (both heterozygous and homozygous) *isolated* SNPs from any number of read datasets without a reference genome, DiscoSnp++ detects and ranks all kinds of SNPs as well as small indels. As shown in [28], DiscoSnp++ performs better than state-of-the-art methods in terms of both computational resources and quality of the results.

DiscoSnp++ is composed of several independent tools. As a preprocessing step, the dBG of the input datasets is built, by also removing erroneous k-mers. Then, DiscoSnp++ detects bubbles generated by the presence of SNPs (isolated or not) and indels, and it outputs a fasta file containing the variant sequences (KisSnp2 module). A final step (kissreads2) maps back the reads from all input read sets on the variant sequences, mainly in order to determine the read coverage per allele and per read set of each variant. This module also computes a rank per variant, indicating whether it exhibits discriminant allele frequencies in the datasets. The last module generates a .vcf of the predicted variants. If no reference genome is provided this step is a change of format from fasta to .vcf (VCFcreator module).

We propose two experiments simulating two human chromosomes haploid read sets obtained mutating (with real .vcf files) real reference chromosomes[9]. The final goal of the experiments is to reconstruct the variations contained in the original (ground truth) .vcf files. We generated the mutated chromosomes using the 1000 genome project (phase 3) .vcf files[10] related to chromosomes 16 and 22, suitably filtered to keep only SNPs of individuals HG00100 (ch.16) and HG00096 (ch.22). From these files, we simulated Illumina sequencing with SimSeq [9], both for reference and mutated chromosomes: individual HG00096 (ch.22) at a 29x getting $15,000,000$ of 100-bp reads, and individual HG00100 (ch.16) a 22x getting $20,000,000$ of 100-bp reads. To simulate the reads, we used the HiSeq error profile[11] publicly available in the SimSeq's repository. Note that our experiments are easily reproducible given the links of the datasets, simulator, and error profile we have provided.

---

[8] Since other tools such as DiscoSnp++ do not preserve the order of samples in the output, we actually check also the variant $s'' \to s'$ and also search the range corresponding to L'' and R''

[9] `ftp.1000genomes.ebi.ac.uk//vol1/ftp/technical/reference/phase2_reference_assembly_sequence/hs37d5.fa.gz`

[10] `ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/`

[11] `https://github.com/jstjohn/SimSeq/blob/master/examples/hiseq_mito_default_bwa_mapping_mq10_1.txt`

Table 1 Pre-processing comparative results of eBWTclust (i.e. building the eBWT using either Egsa or BCR) and DiscoSnp++ (i.e. building the de Bruijn graph). Wall clock is the elapsed time from start to completion of the instance, while RAM is the peak Resident Set Size (RSS). Both values were taken with `/usr/bin/time` command.

Dataset	Coverage per Sample	#reads	Preprocessing	Wall Clock (h:mm:ss)	RAM (MB)
HG00096 (ch. 22)	29x	15,000,000	gsacak	0:53:34	100607
			Egsa	1:41:37	30720
			BCR	4:18:00	1970
			DiscoSnp++	00:01:09	5170
HG00100 (ch. 16)	22x	20,000,000	gsacak	1:13:04	112641
			Egsa	3:39:04	30720
			BCR	6:10:28	3262
			DiscoSnp++	00:02:01	6111

Our framework has been implemented in C++ and is available at `https://github.com/nicolaprezza/eBWTclust`. All tests were done on a DELL PowerEdge R630 machine, used in non exclusive mode. Our platform is a 24-core machine with Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40 GHz, with 128 GB of shared memory. The system is Ubuntu 14.04.2 LTS. Note that, unlike DiscoSnp++, our tool is currently able to use one core only.

We experimentally observed that the pre-processing step (see Table 1) is more computationally expensive than the actual SNP calling step. The problem of computing the eBWT is being intensively studied, and improving its efficiency is out of the aim of this paper. However, a recent work [7] suggests that direct storing of read data with a compressed eBWT leads to considerable space savings, and could therefore become the standard in the future. Our strategy can be easily adapted to directly take as input these compressed formats. Building the dBG requires a few minutes and, in order to keep the RAM usage very low, no other information other than k-mer presence is stored in the dBG used by DiscoSnp++. On the other hand, the construction of the eBWT, LCP and GSA arrays can take a few hours. So, overall DiscoSnp++ is faster than eBWTclust when considering the whole processing.

We run DiscoSnp++ with default parameters (includes k-mers size 31) except for $P = 3$ (it searches up to P SNPs per bubble) and b ($b = 0$ forbids variants for which any of the two paths is branching; $b = 2$ imposes no limitation on branching; $b = 1$ is inbetween).

eBWTclust takes as input few main parameters, among which the most important are the lengths of right and left contexts surrounding SNPs in the output (`-L` and `-R`), the minimum cluster size (`-m`), and (`-v`) the maximum number of non-isolated SNPs to seek in the left contexts (like parameter P of DiscoSnp++). In order to make a fair comparison between DiscoSnp++ and eBWTclust, with eBWTclust we decided to output (exactly as for DiscoSnp++) 30 nucleotides following the SNP (`-R 30`), 31 nucleotides preceding and including the SNP (`-L 31`) (i.e. the output reads are of length 61, with the SNP in the middle position), and `-v 3` (as we used $P = 3$ with DiscoSnp++). For the minimum cluster size, we tested both combinations `-m 6` and `-m 4`. We also show experiments with `-L 20` in order to study how extracting shorter left-contexts impacts running times and precision.

In Table 2, we show the number of TP, FP and FN as well as sensitivity (SEN), precision (PREC), and the number of non-isolated SNPs found by the tools. The outcome is that eBWTclust is always more precise and sensitive than DiscoSnp++. Moreover, while in our case precision is stable and always quite high (always between 94% – 99%), for

Table 2 Post-processing comparative results of eBWTclust (i.e. building clusters from the eBWT and performing SNP calling) and DiscoSnp++ (i.e. running KisSnp2 and kissreads2 using the pre-computed de Bruijn graph). Wall clock (mm:ss) is the elapsed time from start to completion of the instance, while RAM is the peak Resident Set Size (RSS). Both values were taken with `/usr/bin/time` command.

Individual HG00096 vs reference (chromosome 22, 50818468bp), coverage 29× per sample

Tool	Param.	Wall Clock	RAM (MB)	TP	FP	FN	SEN (%)	PREC (%)	Non-isol. SNP
DiscoSnp++	b=0	5:07	101	32773	3719	13274	71.17	89.81	4707/8658
	b=1	16:39	124	37155	10599	8892	80.69	77.80	5770/8658
	b=2	20:42	551	40177	58227	5870	87.25	40.83	6325/8658
eBWTclust	m=4 L=20	19:43	415	42973	2639	3074	93.32	94.21	7268/8658
	m=6 L=20	24:58	411	41972	630	4075	91.15	98.52	6940/8658
	m=4 L=31	35:56	314	42309	1487	3738	91.88	96.60	7233/8658
	m=6 L=31	22:19	300	40741	357	5306	88.47	99.13	6884/8658

Individual HG00100 vs reference (chromosome 16, 90338345bp), coverage 22× per sample

Tool	Param.	Wall Clock	RAM (MB)	TP	FP	FN	SEN (%)	PREC (%)	Non-isol. SNP
DiscoSnp++	b=0	6:20	200	48119	10226	18001	72.78	82.47	6625/11055
	b=1	31:57	208	53456	24696	12664	80.85	68.40	7637/11055
	b=2	51:45	1256	57767	124429	8353	87.37	31.71	8307/11055
eBWTclust	m=4 L=20	41:12	423	61264	943	4856	92.66	98.48	9314/11055
	m=6 L=20	43:51	419	58085	391	8035	87.85	99.33	8637/11055
	m=4 L=31	33:24	418	59668	898	6452	90.24	98.51	9287/11055
	m=6 L=31	44:53	337	53749	190	12371	81.29	99.64	8169/11055

DiscoSnp++ precision is much lower in general, and even drops with $b = 2$, especially with lower coverage, when inversely sensitivity grows. Sensitivity of DiscoSnp++ gets close to that of eBWTclust only in case $b = 2$, when its precision drops and memory and time get worse than ours.

Note that precision and sensitivity of DiscoSnp++ are consistent with those reported in [28]. In their paper (Table 2), the authors report a sensitivity of 79.31% and a precision of 72.11% for DiscoSnp++ evaluated on a Human chromosome with simulated reads (i.e. using an experimental setting similar to ours). In our experiments, using parameter $b = 1$, DiscoSnp++'s sensitivity and precision are, on average between the two datasets, 80.67% and 73.1%, respectively. Therefore, such results almost perfectly match those obtained by the authors of [28]. The same Table 2 of [28] shows that DiscoSnp++ can considerably increase precision at the expense of sensitivity by filtering low-ranking calls. By requiring $rank > 0.2$, the authors show that their tool achieves a sensitivity of 65.17% and a precision of 98.73%. While we have not performed this kind of filtering in our experiments, we note

that also in this case eBWTclust's sensitivity would be higher than that of DiscoSnp++. Precision of the two tools, on the other hand, would be comparable.

Finally, we note that also DiscoSnp++ has been evaluated by the authors of [28] using the SimSeq simulator (in addition to other simulators which, however, yield similar results). We remark that SimSeq simulates position-dependent sequencing errors, while our theoretical assumptions are more strict and require position-independent errors. Similarly, we assume a uniform random genome, while in our experiments we used real Human chromosomes. Since in both cases our theoretical assumptions are more stringent than those holding on the datasets, the high accuracy we obtain is a strong evidence that our theoretical analysis is robust to changes towards less-restrictive assumptions. We plan to test our (extended) prototype on real reads in a forthcoming extension of this paper.

5 Conclusions

We introduced a positional clustering framework for the characterization of breakpoints in the eBWT, paving the way to several possible applications in assembly-free and reference-free analysis of NGS data. The experiments proved the feasibility and potential of our approach. Further work will focus on improving the prediction in highly repeated genomic regions and using our framework to predict SNPs, predict INDELs, haplotyping, correcting sequencing errors, detecting Alternative Splicing events in RNA-Seq data, and sequence assembly.

References

1 C. Ander, O.B. Schulz-Trieglaff, J. Stoye, and A.J. Cox. metaBEETL: high-throughput analysis of heterogeneous microbial populations from shotgun DNA sequences. *BMC Bioinf.*, 14(5):S2, 2013.

2 M.J. Bauer, A.J. Cox, and G. Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theoret. Comput. Sci.*, 483(0):134–148, 2013.

3 E. Birmelé, P. Crescenzi, R.A. Ferreira, R. Grossi, V. Lacroix, A. Marino, N. Pisanti, G.A.T. Sacomoto, and M.-F. Sagot. Efficient Bubble Enumeration in Directed Graphs. In *SPIRE*, LNCS 7608, pages 118–129, 2012.

4 M. Burrows and D.J. Wheeler. A Block Sorting data Compression Algorithm. Technical report, DIGITAL System Research Center, 1994.

5 A.J. Cox, F. Garofalo, G. Rosone, and M. Sciortino. Lightweight LCP construction for very large collections of strings. *J. Discrete Algorithms*, 37:17–33, 2016.

6 A.J. Cox, T. Jakobi, G. Rosone, and O.B. Schulz-Trieglaff. Comparing DNA sequence collections by direct comparison of compressed text indexes. In *WABI*, LNBI 7534, pages 214–224, 2012.

7 D.D. Dolle, Z. Liu, M. Cotten, J.T. Simpson, Z. Iqbal, R. Durbin, S.A. McCarthy, and T.M. Keane. Using reference-free compressed data structures to analyze sequencing reads from thousands of human genomes. *Gen. Res.*, 27(2):300–309, 2017.

8 L. Egidi and G. Manzini. Lightweight BWT and LCP merging via the Gap algorithm. In *SPIRE*, LNCS 10508, pages 176–190, 2017.

9 D. Earl et al. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Gen. Res.*, 21(12):2224–2241, 2011.

10 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *FOCS*, pages 390–398, 2000.

11 S.N. Gardner and B.G. Hall. When Whole-Genome Alignments Just Won't Work: kSNP v2 Software for Alignment-Free SNP Discovery and Phylogenetics of Hundreds of Microbial Genomes. *PLoS ONE*, 8(12):e81760, 2013.

12 Z. Iqbal, I. Turner, G. McVean, P. Flicek, and M. Caccamo. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

13 K. Kimura and A. Koike. Analysis of genomic rearrangements by using the Burrows-Wheeler transform of short-read data. *BMC Bioinf.*, 16(suppl.18):S5, 2015.

14 K. Kimura and A. Koike. Ultrafast SNP analysis using the Burrows-Wheeler transform of short-read data. *Bioinformatics*, 31(10):1577–1583, 2015.

15 T.M. Kowalski, S. Grabowski, and S. Deorowicz. Indexing arbitrary-length k-mers in sequencing reads. *PLoS ONE*, 10(7), 2015.

16 B. Langmead and S.L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, 9(4):357–359, 2012.

17 R.M. Leggett and D. MacLean. Reference-free SNP detection: dealing with the data deluge. *BMC Genomics*, 15(4):S10, 2014.

18 R.M. Leggett, R.H. Ramirez-Gonzalez, W. Verweij, C.G. Kawashima, Z. Iqbal, J.D.G. Jones, M. Caccamo, and D. MacLean. Identifying and Classifying Trait Linked Polymorphisms in Non-Reference Species by Walking Coloured de Bruijn Graphs. *PLoS ONE*, 8(3):1–11, 03 2013.

19 C. Lemaitre, L. Ciortuz, and P. Peterlongo. Mapping-free and assembly-free discovery of inversion breakpoints from raw NGS reads. In *AlCoB*, pages 119–130, 2014.

20 H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

21 R. Li, C. Yu, Y. Li, T. W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

22 S. Li, R. Li, H. Li, J. Lu, Y. Li, L. Bolund, M.H. Schierup, and J. Wang. SOAPindel: efficient identification of indels from short paired reads. *Gen. Res.*, 23(1):195–200, 2013.

23 A. Limasset, J.-F. Flot, and P. Peterlongo. Toward perfect reads: self-correction of short reads via mapping on de Bruijn graphs. *CoRR*, abs/1711.03336, 2017.

24 F.A. Louza, S. Gog, and G.P. Telles. Inducing enhanced suffix arrays for string collections. *Theor. Comput. Sci.*, 678:22–39, 2017.

25 F.A. Louza, G.P. Telles, S. Hoffmann, and C.D.A. Ciferri. Generalized enhanced suffix array construction in external memory. *Algorithms for Molecular Biology*, 12(1):26, 2017.

26 U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *SODA*, pages 319–327, 1990.

27 S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An extension of the Burrows-Wheeler Transform. *Theoret. Comput. Sci.*, 387(3):298–312, 2007.

28 P. Peterlongo, C. Riou, E. Drezen, and C. Lemaitre. DiscoSnp++: de novo detection of small variants from raw unassembled read set(s). *bioRxiv*, 2017.

29 P. Peterlongo, N. Schnel, N. Pisanti, M.-F. Sagot, and V. Lacroix. Identifying SNPs without a Reference Genome by comparing raw reads. In *SPIRE*, LNCS 6393, pages 147–158, 2010.

30 N. Philippe, M. Salson, T. Lecroq, M. Léonard, T. Commes, and E. Rivals. Querying large read collections in main memory: a versatile data structure. *BMC Bioinf.*, 12:242, 2011.

31 G.A.T. Sacomoto, J. Kielbassa, R. Chikhi, R. Uricaru, P. Antoniou, M.-F. Sagot, P. Peterlongo, and V. Lacroix. KISSPLICE: de-novo calling alternative splicing events from RNA-seq data. *BMC Bioinf.*, 13(S-6):S5, 2012.

32 L. Salmela and E. Rivals. LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514, 2014.

33 L. Salmela, R. Walve, E. Rivals, and E. Ukkonen. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics*, 33(6):799–806, 2017.

34 M. Schirmer, R. D'Amore, U.Z. Ijaz, N. Hall, and C. Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC Bioinf.*, 17(1):125, 2016.

35 J. Schröder, H. Schröder, S.J. Puglisi, R. Sinha, and B. Schmidt. SHREC: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.

36 F. Shi. Suffix Arrays for Multiple Strings: A Method for On-Line Multiple String Searches. In *ASIAN*, LNCS 1179, pages 11–22, 1996.

37 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015.

38 R. Uricaru, G. Rizk, V. Lacroix, E. Quillery, O: Plantard, R. Chikhi, C. Lemaitre, and P. Peterlongo. Reference-free detection of isolated SNPs. *Nuc.Acids Res*, 43(2):e11, 2015.

39 N. Välimäki and E. Rivals. Scalable and Versatile k-mer Indexing for High-Throughput Sequencing Data. In *ISBRA*, LNCS 7875, pages 237–248, 2013.

Haplotype-aware graph indexes

Jouni Sirén[1]
University of California, Santa Cruz, USA
Wellcome Sanger Institute, Hinxton, UK
jouni.siren@iki.fi
https://orcid.org/0000-0001-5828-4139

Erik Garrison[2]
Wellcome Sanger Institute, Hinxton, UK
eg10@sanger.ac.uk
https://orcid.org/0000-0003-3821-631X

Adam M. Novak[3]
University of California, Santa Cruz, USA
anovak@soe.ucsc.edu
https://orcid.org/0000-0001-5828-047X

Benedict J. Paten[4]
University of California, Santa Cruz, USA
bpaten@ucsc.edu
https://orcid.org/0000-0001-8863-3539

Richard Durbin[5]
Department of Genetics, University of Cambridge, UK
Wellcome Sanger Institute, Hinxton, UK
rd@sanger.ac.uk
https://orcid.org/0000-0002-9130-1006

Abstract

The variation graph toolkit (VG) represents genetic variation as a graph. Each path in the graph is a potential haplotype, though most paths are unlikely recombinations of true haplotypes. We augment the VG model with haplotype information to identify which paths are more likely to be correct. For this purpose, we develop a scalable implementation of the graph extension of the positional Burrows–Wheeler transform. We demonstrate the scalability of the new implementation by indexing the 1000 Genomes Project haplotypes. We also develop an algorithm for simplifying variation graphs for k-mer indexing without losing any k-mers in the haplotypes.

2012 ACM Subject Classification Theory of computation → Pattern matching, Theory of computation → Data compression, Applied computing → Computational genomics

Keywords and phrases FM-indexes, variation graphs, haplotypes

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.4

[1] Funded by Wellcome Trust grant WT206194, the National Institutes of Health (5U41HG007234, 1U01HL137183-01), and the W. M. Keck Foundation (DT06172015)
[2] Funded by Wellcome Trust grant WT206194.
[3] Funded by the National Institutes of Health (5U41HG007234, 1U01HL137183-01) and the W. M. Keck Foundation (DT06172015)
[4] Funded by the National Institutes of Health (5U41HG007234, 1U01HL137183-01) and the W. M. Keck Foundation (DT06172015)
[5] Funded by Wellcome Trust grant WT206194.

Supplement Material The implementation can be found at `https://github.com/vgteam/vg`, `https://github.com/jltsiren/gbwt`, and `https://github.com/jltsiren/gcsa2`.

1 Introduction

Sequence analysis pipelines often start by mapping the sequence reads to a *reference genome* of the same species. A read aligner first uses a *text index* to find candidate positions for the read. Then it aligns the read to the candidate positions, trying to find the best mapping.

The reference genome may represent the sequenced individual poorly, e.g. when it is a single sequence that diverges substantially at some location. Mapping reads to such a reference can introduce *reference bias* to the subsequent analysis. Richer reference models can help to avoid the bias, but challenges remain in choosing the right model and working with it effectively [28].

We can replace the single reference sequence with a *collection of haplotypes*. Because individual genomes are similar, compressed text indexes can store such collections in very little space [18]. However, due to this similarity, most reads map equally well to many haplotypes. If the reference model is a simple collection, we cannot tell whether a read maps to the same position in different haplotypes or not.

If the haplotypes are *aligned*, we can use the alignment to determine whether the mappings are equivalent. Text indexes can also take advantage of the alignment by storing shared substrings only once [13]. The FM-index of alignment [19, 20] goes one step further by collapsing the multiple alignment into a *directed acyclic graph* (DAG), where each node is labeled by a sequence. It indexes the graph and stores some additional information for determining which paths correspond to valid haplotypes.

We can also build a reference graph directly from a reference sequence and a set of *variants* [24]. This approach has been used in many tools such as GCSA [26], BWBBLE [12], vBWT [16], the Seven Bridges Graph Pipeline [23], and Graphtyper [5]. Algorithms for working with sequences are often easy to generalize to DAGs. On the other hand, because an acyclic graph imposes a global alignment on the haplotypes, allowing only matches, mismatches, and indels, it cannot represent structural variation such as duplications or inversions adequately.

Assembly graphs such as *de Bruijn graphs* collapse sequences by local similarity instead of global alignment. They are better suited to handling structural variation than DAGs. However, the lack of a global coordinate system limits their usefulness as references.

Graph-based reference models share certain weaknesses. Because they collapse sequences between variants, they represent both the original haplotypes and their *recombinations*, that is paths that switch between haplotypes. This may cause false positives when a read maps better to an unobserved recombination than to the correct path. Graph regions with many variants in close proximity can give rise to very large numbers of recombinant paths, and be *too complex* to index completely. Graph tools try to deal with such regions by, for example, limiting the amount of variation in the graph, artificially simplifying complex regions, and making trade-offs between query performance, index size, and maximum query length.

CHOP [17] embeds haplotypes into a graph and indexes the corresponding paths. For a given parameter k, the graph is transformed into a collection of short strings such that adjacent strings overlap by $k-1$ characters. Each haplotype can be represented as a sequence of adjacent strings. Any read aligner can be used to map reads to the strings. However, because the aligner sees only short strings, it cannot map long reads or paired-end reads.

The *variation graph toolkit* (VG) [9] works with many kinds of graphs. While the tools mentioned earlier use graphs to represent other information (e.g. sequences or variants),

the graph itself is the primary object in the VG model. A global coordinate system can be provided by designating certain paths as *reference paths*.

VG uses GCSA2 [25] as its text index. GCSA2 represents a k-mer index as a de Bruijn graph and compresses it structurally by merging redundant nodes. VG handles complex graph regions by indexing a *simplified graph*, although the final alignment is done in the original graph. The drawback of this approach is that simplification can break paths corresponding to known haplotypes, while leaving paths representing recombinations intact.

In this paper, we augment the VG model with haplotype information. We develop the GBWT, a scalable implementation of the graph extension of the positional Burrows–Wheeler transform (gPBWT) [4, 21], to store the haplotypes as paths in the graph. To demonstrate the scalability of the GBWT, we build an index for the the 1000 Genomes Project haplotypes. We also describe an algorithm that adds the haplotype paths back to the simplified graph, without reintroducing too much complexity.

The main differences to the old gPBWT implementation [21] are:

- We use *local structures* for each node instead of global structures for the graph. The index is smaller and faster and takes better advantage of memory locality.
- The GBWT is implemented as an ordinary text index instead of a special-purpose index for paths. Most FM-index algorithms can be used with it. For example, we can use the GBWT as an FMD-index [14] and support *bidirectional search*.
- We have a fast and space-efficient *incremental construction* algorithm that does not need access to the entire collection of haplotypes at the same time.
- Our implementation can be used *independently* of VG.

The haplotype information stored in GBWT can also be used to improve read mapping, and potentially, variant inference. We leave this investigation to a subsequent paper.

2 Background

2.1 Strings and graphs

A *string* $S[0, n-1] = s_0 \cdots s_{n-1}$ of length $|S| = n$ is a sequence of *characters* over an *alphabet* $\Sigma = \{0, \ldots, \sigma - 1\}$. *Text* strings $T[0, n-1]$ are terminated by an *endmarker* $T[n-1] = \$ = 0$ that does not occur anywhere else in the text. *Substrings* of string S are sequences of the form $S[i, j] = s_i \cdots s_j$. We call substrings of length k k-mers and substrings of the type $S[0, j]$ and $S[i, n-1]$ *prefixes* and *suffixes*, respectively.

Let $S[0, n-1]$ be a string. We define $S.\mathsf{rank}(i, c)$ as the number of occurrences of character c in the prefix $S[0, i-1]$. We also define $S.\mathsf{select}(i, c) = \max\{j \leq n \mid S.\mathsf{rank}(j, c) < i\}$ as the position of the occurrence of rank $i > 0$. A *bitvector* is a data structure that stores a binary sequence and supports efficient $\mathsf{rank}/\mathsf{select}$ queries over it.

A *graph* $G = (V, E)$ consists of a finite set of *nodes* $V \subset \mathbb{N}$ and a set of *edges* $E \subseteq V \times V$. We assume that the edges are *directed*: $(u, v) \in E$ is an edge *from* node u *to* node v. The *indegree* of node v is the number of *incoming* edges to v, while the *outdegree* is the number of *outgoing* edges from v. Let $P = v_0 \cdots v_{|P|-1}$ be a string over the set of nodes V. We say that P is a *path* in graph $G = (V, E)$, if $(v_i, v_{i+1}) \in E$ for all $0 \leq i < |P| - 1$.

The VG model [9] is based on *bidirected* graphs, where each node has two *orientations*. We simulate them with directed graphs. We partition the set of nodes V into *forward* nodes V_f and *reverse* nodes V_r, with $V_f \cap V_r = \emptyset$ and $|V_f| = |V_r|$. We match each forward node $v \in V_f$ with the corresponding reverse node $\overline{v} \in V_r$, with $\overline{\overline{v}} = v$ for all $v \in V_f$. For all nodes $u, v \in V$, we also require that $(u, v) \in E \iff (\overline{v}, \overline{u}) \in E$.

2.2 FM-index

The *suffix array* $\mathsf{SA}[0, n-1]$ of text $T[0, n-1]$ is an array of pointers to the suffixes of the text in *lexicographic order*. For all $i < j$, we have $T[\mathsf{SA}[i], n-1] < T[\mathsf{SA}[j], n-1]$. The *Burrows–Wheeler transform* (BWT) [2] is a permutation of the text with a similar combinatorial structure. We define it as string $\mathsf{BWT}[0, n-1]$, where $\mathsf{BWT}[i] = T[(\mathsf{SA}[i] - 1) \bmod n]$. Let $\mathsf{C}[c]$ be the number of occurrences of characters $c' < c$ in the text. The main operation in BWT is the *LF-mapping*, which we define as $\mathsf{LF}(i, c) = \mathsf{C}[c] + \mathsf{BWT.rank}(i, c)$. We use shorthand $\mathsf{LF}(i)$ for $\mathsf{LF}(i, \mathsf{BWT}[i])$ and note that $\mathsf{SA}[\mathsf{LF}(i)] = (\mathsf{SA}[i] - 1) \bmod n$

Let X be a string and let c be a character. If $T' < X$ for i suffixes T' of text T, we say that string X has *lexicographic rank* i among the suffixes of text T. The number of suffixes starting with any character $c' < c$ is $\mathsf{C}[c]$, and the number of suffixes $T' < X$ preceded by character c is $\mathsf{BWT.rank}(i, c)$. Hence the lexicographic rank of string cX is $\mathsf{LF}(i, c)$.

The FM-index [6] is a text index based on the BWT. Assume that we can compute $\mathsf{BWT.rank}(i, c)$ in t_r time. Further assume that we have stored $(i, \mathsf{SA}[i])$ for all $\mathsf{SA}[i]$ divisible by some integer $d > 0$. The FM-index supports the following queries:

- $\mathsf{find}(X)$: Return the *lexicographic range* $[sp, ep]$ of suffixes starting with *pattern* X. If $[sp_{i+1}, ep_{i+1}]$ is the lexicographic range for pattern $X[i+1, |X|-1]$, the range for pattern $X[i, |X|-1]$ is $[\mathsf{LF}(sp_{i+1}, X[i]), \mathsf{LF}(ep_{i+1}+1, X[i])-1]$. By extending the pattern backwards, we can support $\mathsf{find}(X)$ in $\mathsf{O}(|X| \cdot t_r)$ time.
- $\mathsf{locate}(sp, ep)$: Return the *occurrences* $\mathsf{SA}[sp, ep]$. For each $i \in [sp, ep]$, we iterate $\mathsf{LF}(i)$ until we find a stored pair $(\mathsf{LF}^k(i), \mathsf{SA}[\mathsf{LF}^k(i)])$. Then $\mathsf{SA}[i] = \mathsf{SA}[\mathsf{LF}^k(i)] + k$. Locating each occurrence $\mathsf{SA}[i]$ takes $\mathsf{O}(d \cdot t_r)$ time.
- $\mathsf{extract}(j, j')$: Return the substring $T[j, j']$. We start from the nearest stored $(i, \mathsf{SA}[i])$ with $\mathsf{SA}[i] > j'$ and iterate $(i, \mathsf{SA}[i]) \leftarrow (\mathsf{LF}(i), \mathsf{SA}[i] - 1)$ until $\mathsf{SA}[i] = j + 1$. As $\mathsf{BWT}[i] = T[\mathsf{SA}[i] - 1]$, we extract the substring backwards in $\mathsf{O}((d + j' - j) \cdot t_r)$ time.

We can generalize the FM-index to indexing *multiple texts* $T_0, \ldots, T_{m-1}$. Each text T_j is terminated by a distinct endmarker $\$_j$, where $\$_j < \$_{j+1}$ for all j. As the suffixes of the texts are all distinct, we can sort them unambiguously. In the final BWT, we replace each $\$_j$ with $\$$ in order to reduce alphabet size. The index works as with a single text, except that we cannot compute $\mathsf{LF}(i, \$)$. We also define the *document array* DA as an array of *text identifiers*. If $\mathsf{SA}[i]$ points to a suffix of text T_j, we define $\mathsf{DA}[i] = j$.

3 Indexing haplotypes

3.1 Positional BWT

Assume that we have m haplotype strings $S_0, \ldots, S_{m-1}$ of equal length over alphabet Σ. At each variant site i, character $S_j[i]$ tells whether haplotype j contains the reference allele ($S_j[i] = 0$) or an alternate allele ($S_j[i] > 0$). Given a pattern X and a range of sites $[i, i']$, we want to find the haplotypes S_j matching the pattern at the specified sites ($S_j[i, i'] = X$). Ordinary FM-indexes do not support such queries, as they find all occurrences of the pattern.

The *positional BWT* (PBWT) [4] is an FM-index that supports *positional queries*. We can interpret it as the FM-index of texts $T_0, \ldots, T_{m-1}$ such that $T_j[i] = (i, S_j[i])$ [7]. If we want to search for pattern X in range $[i, i']$, we search for pattern $X' = (i, X[0]) \cdots (i', X[|X|-1])$ in the FM-index. The texts are over a large alphabet, but their *first-order empirical entropy* is low. We can encode the BWT using alphabet Σ with a simple model. Assume that $\mathsf{SA}[x]$ points to a suffix starting with $(i+1, c)$. We often know the character from a previous query, and we can determine it using the C array. Then $\mathsf{BWT}[x] = (i, c')$ for a $c' \in \Sigma$, and we can

encode it as c'. (Note that we build the rank structure for the original BWT, not the encoded BWT.) Because the collection of haplotype strings is usually *repetitive*, we can compress the PBWT further by *run-length encoding* the BWT [18].

3.2 Graph extension

Haplotypes correspond to paths in the VG model. Because chromosome-length phasings are often not available, there may be multiple paths for each haplotype. The *graph extension* of the PBWT [21] generalizes the PBWT to indexing such paths. While the original extension was specific to VG graphs, we present a simplified version over directed graphs. We call this structure the *Graph BWT* (GBWT), as it encodes the BWT using the graph as a model.

Let $P_0, \ldots, P_{m-1}$ be paths in graph $G = (V, E)$. We can interpret the paths as strings over alphabet V. Assume that $0 \notin V$, as we use it as the endmarker. We build an FM-index for the *reverse strings*. We sort reverse prefixes in lexicographic order, so the LF-mapping traverses edges in the correct direction, and place the endmarker before the string.

For each node $v \in V$, we define the *local alphabet* $\Sigma_v = \{w \in V \mid (v, w) \in E\}$. We also add \$ to Σ_v if v is the last node on a path, and define $\Sigma_\$$ as the set of the initial nodes on each path. We partition the BWT into substrings BWT_v corresponding to the prefixes ending with v, and encode each substring BWT_v using the local alphabet Σ_v. If $w \in \Sigma_v$ is the kth character in the local alphabet in sorted order, we encode it as $\Sigma_v(w) = k$.

The GBWT supports the following variants of the standard FM-index queries:

- $\mathsf{find}(X)$ returns the lexicographic range of reverse prefixes starting with the reverse pattern (the range of prefixes ending with the pattern).
- $\mathsf{locate}(sp, ep)$ returns the haplotype identifiers $\mathsf{DA}[sp, ep]$. We do not return text offsets, as the node corresponding to the range $[sp, ep]$ already provides similar information.
- $\mathsf{extract}(j)$ returns the path P_j. We save memory by not supporting substring extraction.

These queries should be understood as examples of what we can support. Because the GBWT is an FM-index for multiple texts, most algorithms using an FM-index can be adapted to use the GBWT. For example, let $P = v_0 \cdots v_{|P|-1}$ be a path. The *reverse path* of P is the path $\overline{P} = \overline{v_{|P-1|}} \ldots \overline{v_0}$ traversing the reverse nodes in the reverse order. If we also index $\overline{P}$ for every path P, the GBWT becomes an FMD-index [14] that supports *bidirectional search*.

3.3 Records

We develop a GBWT representation based on the following assumptions:

1. Almost all nodes $v \in V$ have a low outdegree, making the local alphabet Σ_v small. Hence we can afford storing the rank of all $w \in \Sigma_v$ at the start of BWT_v. Decompressing that information every time we access the node does not take too much time either.
2. The number of occurrences of almost all nodes is bounded by the number of haplotypes. As the length of BWT_v is bounded for almost all $v \in V$, we can afford scanning it every time we compute rank within it.
3. The collection of paths is repetitive. Run-length encoding compresses the BWT well, reducing both index size and the time required for scanning BWT_v.
4. The set of nodes V is a dense subset of a range $[a, b]$. Hence we can afford storing some information for all $i \in [a, b]$ without using too much space.
5. The graph is almost linear and almost topologically sorted. The closer to topological order we can store the nodes, the less space we need for graph topology, and the better we can take advantage of *memory locality*.

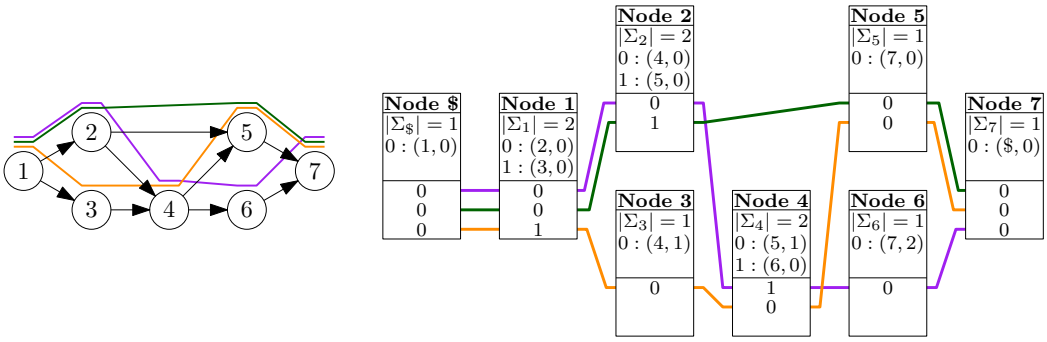


Figure 1 Left: A graph with three paths. Right: GBWT of the paths.

We store a *record* consisting of a *header* and a *body* for each node $v \in V$ and for the endmarker \$. For each character $w \in \Sigma_v$ in sorted order, the header stores a pair $(w, \mathsf{BWT.rank}(v, w))$, where $\mathsf{BWT.rank}(v, w)$ is the total number of occurrences of character w in all $\mathsf{BWT}_{v'}$ with $v' < v$. The body run-length encodes BWT_v, representing a run of ℓ copies of character w as a pair $(\Sigma_v(w), \ell)$. See Figure 1 for an example.

Because the BWT is a set of records, we use node/offset pairs as *positions*. Pair (v, i) refers to offset $\mathsf{BWT}[\mathsf{C}[v] + i] = \mathsf{BWT}_v[i]$. We define rank queries over positions as

$$\mathsf{BWT.rank}((v, i), w) = \mathsf{BWT.rank}(v, w) + \mathsf{BWT}_v.\mathsf{rank}(i, w).$$

Similarly, we define $\mathsf{LF}((v, i), w) = (w, \mathsf{BWT.rank}((v, i), w))$ and use it in place of ordinary LF-mapping in the FM-index.

The FM-index is based on iterating LF-mapping. Because LF-mapping tends to jump randomly around the BWT, this can be a significant bottleneck. GBWT achieves better memory locality, if we store the records for adjacent nodes close to each other. When we iterate LF-mapping over a path in the graph, we traverse adjacent memory regions.

As a run-length encoded FM-index, GBWT supports the *fast* locate() algorithm [18]. The *direct algorithm*, as described in Section 2.2, locates each position $i \in [sp, ep]$ separately. If we instead process the entire range at once, advancing every position by one step of LF-mapping at the same time, we achieve better memory locality. We can also compute LF-mapping for an entire run $\mathsf{BWT}_v[x, y] = w^{y+1-x}$ in the same time as for a single position $i \in [x, y]$.

3.4 GBWT encodings

Dynamic GBWT is intended for index construction, where speed is more important than size. We have an array of fixed-size records for characters \$ and $v \in [a, b]$, including character values $v \notin V$. The record for v has four pointers to arrays: header, body, incoming edges, and haplotype identifiers. For each incoming edge $(u, v) \in E$, the incoming edges array stores a pair $(u, \mathsf{BWT}_u.\mathsf{rank}(|\mathsf{BWT}_u|, v))$, recording the number of paths crossing from u to v.

Let SA_v and DA_v be the parts of SA and DA corresponding to BWT_v. The haplotype identifiers array for node v stores, in sorted order, pairs $(i, \mathsf{DA}_v[i])$ for which $\mathsf{SA}_v[i]$ points to either the last node on a path or a path position divisible by $d > 0$. These pairs are used for locate() queries, like stored SA pointers in an ordinary FM-index.

Compressed GBWT balances query performance with index size. We use it when the set of haplotypes is fixed and for storing the index on disk. Each record is a byte array. We encode integers as sequences of bytes, where the lower 7 bits contain data and the high bit

tells whether the encoding continues. The header starts with $|\Sigma_v|$. We encode the outgoing edges $(w_i, \mathsf{BWT.rank}(v, w_i))$ *differentially*, replacing w_i with $w_i - w_{i-1}$. If the local alphabet is large, each run (k, ℓ) in the body is encoded as an integer pair. Otherwise we encode k and as much of ℓ as possible in the first byte, and continue with the remaining run length in subsequent bytes. We concatenate all records and mark their starting positions in a sparse bitvector [22]. The records can be accessed with select queries on the bitvector.

Each compressed record must be decompressed sequentially. As the stored haplotype identifiers tend to cluster in certain nodes, storing them in records would make these records large and slow to decompress. Instead, we use a global structure for the haplotype identifiers. The structure consists of three bitvectors and an array of identifiers:

- Uncompressed bitvector B_s marks the records with stored identifiers. If the ith record contains identifiers, we set $B_s[i] = 1$. This allows us to skip checking the identifiers in most records when iterating $\mathsf{LF}()$.
- Sparse bitvector B_r is defined over the concatenated offset ranges of the records with stored identifiers. If $B_s[i] = 1$, the range for the record starts at $B_r.\mathsf{select}(B_s.\mathsf{rank}(i, 1) + 1, 1)$.
- Sparse bitvector B_o covers the same range as B_r. If $B_o[i + j] = 1$ and the range for the record starts at $B_o[i]$, we have an identifier for offset j at array position $B_o.\mathsf{rank}(i + j, 1)$.

4 GBWT construction

The assumptions in Section 3.3 make the GBWT easier to build than an ordinary FM-index. *Inserting* new texts into the collection updates adjacent records, just like searching traverses adjacent records. Because the local alphabet is small, because the number of occurrences of each character is limited, and because run-length encoding compresses the BWT well, records tend to be small. Hence we can afford *rebuilding* a record each time we update it.

On the other hand, the GBWT is harder to build than the PBWT. In the PBWT, all strings are of the same length and have the same variant site at the same position. Hence we can build the final record for a site in a single step. In the GBWT, indels in the haplotypes become indels on the haplotype paths, and hence we have to update the same record multiple times. We also have to buffer the strings instead of indexing them as we generate them.

4.1 Basic construction

The following algorithm [11] updates the BWT of text T to be the BWT of text cT, where c is a character. It forms the basis of many incremental BWT construction algorithms.

1. Find the offset i where $\mathsf{BWT}[i] = \$$ and replace the endmarker with character c.
2. Compute $i' = \mathsf{LF}(i, c)$ and insert a new endmarker between offsets $i' - 1$ and i'.

If we have a BWT for m texts, we can insert a new empty text by inserting an endmarker between offsets $m - 1$ and m. By iterating the above algorithm, we can then insert the actual text. If we have a *dynamic FM-index* [3], this can be quite efficient in practice.

The BCR algorithm [1] builds BWT for m texts. It starts with the BWT for m empty texts and then extends each text by one character in each step. Originally intended for indexing short reads, the BCR algorithm is also used for PBWT construction.

Our GBWT construction algorithm is similar to RopeBWT2 [15]. We have a dynamic GBWT and insert multiple texts into the index in a single *batch* using the BCR algorithm. In each step, we extend each text by one character. In the following, v and w are the current and the next character in the current text T_j and i is a record offset. If v is the last character of the text (the endmarker is at $T_j[0]$), we set $w = \$$. In each step, we:

1. **Rebuild records:** The texts are sorted by positions (v, i) such that the endmarker of that text should be at $\mathsf{BWT}_v[i]$. (We do not write the temporary endmarkers to the records.) We process all texts at the same node v to rebuild the record.
   a. If the record does not contain the edge (v, w), we add $(w, 0)$ to the header.
   b. We add BWT runs and haplotype identifiers until offset i to the new record. If we have inserted k characters so far, we replace haplotype identifier (i', j') with $(i' + k, j')$.
   c. If $w = \$$ or the text position is divisible by d, we insert haplotype identifier (i, j).
   d. We insert w to the BWT and set $i \leftarrow \mathsf{BWT}_v.\mathsf{rank}(i, w)$.
   e. If $w \neq \$$, we increment the number of paths from v to w in the incoming edges of w.
2. **Sort:** We sort the texts by (w, v, i), which is the order we need in the next step. If $w = \$$, the text is now fully inserted, and we remove it from further processing.
3. **Rebuild offsets:** For each distinct node w, we rebuild the $\mathsf{BWT}.\mathsf{rank}(v', w)$ fields in the outgoing edges of predecessor nodes v' using the path counts in the incoming edges of w. Then we set $i \leftarrow i + \mathsf{BWT}.\mathsf{rank}(v, w)$ to have the correct offset in the next step.

4.2 Construction in VG

GBWT construction in VG requires a VCF file with *phasing information*. We expect a diploid genome, though some regions may be haploid. Because we need two layers of buffering, we process the VCF file in batches of s samples (default 200) in order to save memory.

At each variant site and for every haplotype, we determine the path from the previous site to the current site, and extend the buffered path for that haplotype with it. If there is no phasing information at the current site or if we cannot otherwise extend the path P, we insert both the path P and its reverse $\overline{P}$ into the GBWT construction buffer and start a new path. Once the GBWT construction buffer is full (the default size is 100 million), we launch a background thread to insert the batch into the index.

We can *merge* GBWT indexes quickly if the node identifiers do not overlap (e.g. indexes for different chromosomes). The records for all nodes $v \in V$ can be reused in the merged index. In the endmarker record \$, we merge the local alphabets and concatenate the record bodies in some order. When we interleave the global haplotype identifier structures, we have to update the identifiers according to the order we used in the endmarker.

5 Haplotype-aware graph simplification

VG uses *pruning heuristics* to simplify graphs for k-mer indexing. First we remove edges on k-mers that make too many edge choices (e.g. more than 3 choices in a 24-mer). Edges on unary paths are not deleted, as there is no choice in taking them. Then we delete *connected components* with too little sequence (e.g. less than 33 bases). Finally, if the graph contains reference paths, we may restore them to the pruned graph.

Heuristic pruning often breaks paths taken by known haplotypes. This may cause errors in read mapping, if we cannot find candidate positions for a read in the correct graph region. On the other hand, indexing too many recombinations may increase the number of false positives. Hence we would like to prune recombinations while leaving the haplotypes intact.

We describe an algorithm that *unfolds* the haplotype paths in pruned regions, *duplicating* nodes when necessary. Our algorithm works with any pruning algorithm that removes nodes from the graph. See Figure 2 for an example. We work with bidirected VG graphs, unless otherwise noted. Reference paths can also be unfolded with a similar algorithm.

Let $G_i = (V_i, E_i)$ be the graph induced by GBWT paths and $G_p = (V_p, E_p)$ be a pruned graph. We build a *complement graph* induced by edges $E_i \setminus E_p$ and consider each connected

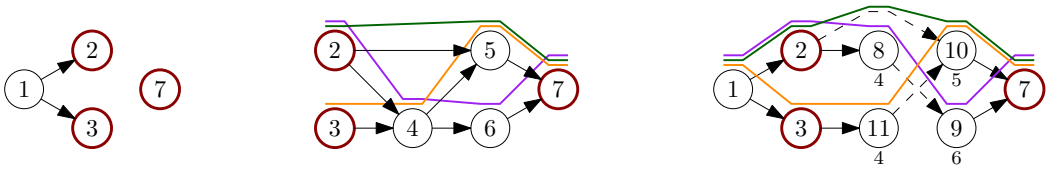


Figure 2 Unfolding the paths in the graph in Figure 1. Border nodes have been highlighted. Left: The graph after removing nodes 4, 5, and 6. Center: Complement graph. The maximal paths are $(2, 4 \mid 6, 7)$, $(2 \mid 5, 7)$, and $(3, 4 \mid 5, 7)$, with the bar splitting a path into a prefix and a suffix. Right: Unfolded graph. Dashed edges cross from prefixes to suffixes. Duplicated nodes have the original ids below the node.

component $G_c = (V_c, E_c)$ in it separately. The set $V_b = V_c \cap V_p$ is the *border* of the component, as the nodes exist both in the component and in the pruned graph. Nodes in the set $V_c \setminus V_b$ are *internal nodes*.

Each connected component G_c represents a graph region that was removed from the original graph. We build an unfolded component consisting of the paths in G_c supported by GBWT paths and insert it into the pruned graph G_p. We achieve this by duplicating the internal nodes that would otherwise cause recombinations.

In order to build the unfolded component, we must find all *maximal* paths P of length $|P| \geq 2$ supported by GBWT paths in the component. A path starting from a border node is maximal if it reaches the border again or cannot be extended any further. GBWT paths consisting entirely of internal nodes of the component are also maximal.

Let v be a GBWT node and $vg(v) \in V_c$ the corresponding VG node. If $vg(v)$ is a border node, we create a *search state* $(v, \mathsf{find}(v))$ consisting of a pattern and a range. For internal nodes, we create state $(v, \mathsf{find}(\$v))$. Then, for each search state $(X, [sp, ep])$, with $x = X[|X| - 1]$:

1. If $|X| \geq 2$ and the last node $vg(x)$ is a border node, we stop the search for this state. If $vg(X[0])$ is also a border node, X is a maximal path, and we output it.
2. We try to extend the search with all GBWT nodes v corresponding to the successors $u \in V_c$ of $vg(x)$, taking the orientation of v from the VG edge. If $[sp', ep'] = [\mathsf{LF}((x, sp), v), \mathsf{LF}((x, ep + 1), v) - 1] \neq \emptyset$, we create a new state $(Xv, [sp', ep'])$.
3. If no extension was successful and $|X| \geq 2$, path X is maximal, and we output it.

Let P be a maximal path we output. If P is not a border-to-border path, we try to extend the lexicographically smaller of P and $\overline{P}$ with reference paths, replacing P with the extended path. To avoid having the same path in both orientations, we replace each path P with the smaller of P and $\overline{P}$.

We could create new duplicates of all internal nodes on P and insert the path into G_p, but this would create too much nondeterminism for GCSA2.[6] Instead, we split each path into a prefix and a suffix of equal length and build a trie of the prefixes and a trie of the reverse suffixes. Every edge in the tries becomes a node in the unfolded component.

Let v be the label of a trie edge starting from the root. If $vg(v)$ is a border node, it already exists in G_p. Otherwise we add a new duplicate of $vg(v)$. Now let v and v' be the

[6] If VG node v has predecessors u and u' with identical labels, k-mers starting from u and u' and passing through v cannot be distinguished. GCSA2 construction has to extend these k-mers until the order of the index (e.g. $k = 256$), which may increase the size of the temporary files significantly.

labels of two successive trie edges, and let u be the VG node we used for v. We create a new duplicate u' of $vg(v')$ and add node u' to G_p. We also add edge (u, u') or (u', u), depending on whether we are in a prefix or a suffix. Finally, if we used VG node u for the end of a prefix and VG node u' for the start of the corresponding suffix, we add edge (u, u') to G_p.

After we have handled all components, the simplified graph G_p contains all GBWT paths. The GCSA2 index of G_p contains all k-mers (e.g. 256-mers) in the haplotypes. This allows us to prune the graph more aggressively, removing more k-mers corresponding to recombinations. In order to map reads to the original graph $G = (V, E)$ instead of the simplified graph G_p, we replace the node identifiers $v \in V_p$ in the GCSA2 index with the original identifiers $v' \in V$.

6 Experiments

We have implemented GBWT in C++ using the SDSL library [10]. The following experiments were done using VG v1.7.0 with prerelease versions of GBWT v0.4 and GCSA2 v1.2. All code was compiled using GCC 5.4. We used a single Amazon EC2 i3.8xlarge instance with 16 physical (32 logical) cores of an Intel Xeon E5 2686 v4 and 244 GiB[7] of memory. The system was running Ubuntu 16.04 with Linux kernel 4.4.0. The temporary files in GCSA2 construction were stored on a local RAID 0 volume consisting of four 1.9 TB SSDs.

6.1 GBWT construction

We built VG graphs from the GRCh37 human reference genome and the *1000 Genomes Project* (1000GP) final phase data [27]. The VCF files had phasings for 2504 humans over approximately 80 million variants. VG transformed the phasings into 29.3 million paths of total length 1.62 trillion in a graph with 493 million nodes (including the reverse paths).

GBWT construction is space-efficient and uses two threads. We first built separate GBWTs for each chromosome, running 12 jobs in parallel. The jobs were ordered $X, 1, \ldots, 22, Y$, as large chromosomes take longer to finish. Total construction time was 29.0 hours. The longest job was 27.1 hours for chromosome 2. The bottleneck was generating haplotype paths, as the insertion threads were running less than half of the total time.

Merging the GBWTs into a single index took less than 9 minutes. The merged index took 14.6 GiB, out of which 7.4 GiB was for the GBWT itself and 7.2 GiB for the haplotype identifiers ($d = 1024$). The dynamic GBWT was roughly 10x larger. Its exact size is not well-defined due to a large number of memory allocations and unused space in the arrays.

All the assumptions in Section 3.3 were valid for our dataset:

1. The average outdegree of a record is 1.34 and the maximum is 13, excluding the endmarker.
2. As the graphs built from a VCF file are acyclic, no haplotype can visit the same node twice.
3. The GBWT takes 0.04 bits per character, excluding the haplotype identifiers.
4. VG construction avoids leaving gaps between node identifiers.
5. The VG graphs built from a VCF file are almost in topological order.

6.2 GBWT benchmarks

For various pattern lengths $|X|$ from 2 to 50, we extracted 100,000 patterns from the whole-genome index and used them for find() queries. The average query times in the compressed GBWT start from 460 ns/character with $|X| = 2$ and go down to 300 ns/character with

[7] Sizes measured in MiB, GiB, and TiB are based on 1024-byte kibibytes. Sizes measured in MB, GB, and TB are based on 1000-byte kilobytes.

Table 1 GCSA2 indexes for simplified 1000GP graphs. Index size in GiB and construction time in hours. Number of distinct 64-mers (in billions) shared with pruned-256, additional haplotype 64-mers, and additional recombination 64-mers.

	GCSA2 index		64-mers		
Graph	Size	Construction	Shared	Haplotype	Recombination
pruned-128	35.4 GiB	25.4 h	27.0 G	3.11 G	11.4 G
pruned-256	29.3 GiB	25.5 h	27.0 G	–	–
unfolded-256	33.7 GiB	28.9 h	27.0 G	3.46 G	–

$|X| = 50$ due to memory locality. This is somewhat slower than in FM-indexes over small alphabets [25]. For the dynamic GBWT, query times were 130 ns/character with $|X| = 2$ and 80 ns/character with $|X| = 50$, or 2–3 times faster than in ordinary FM-indexes.

The locate() performance suffers from the long distance $d = 1024$ between stored identifiers. We extracted 20,000 patterns of length 20 from the index and used the ranges returned by find() queries for locate() benchmarks. The total length of the ranges was 69.1 million. The average query times in the compressed GBWT were 110 µs/position (direct algorithm) and 14 µs/position (fast algorithm). For the dynamic index, the times were 19 µs/position (direct) and 11 µs/position (fast). FM-indexes for non-repetitive text typically use $d = 16$ or $d = 32$ and take a few microseconds to locate each position.

We also extracted 100,000 paths of total length 5.50 billion from the index. The average time per character was 1,800 ns in the compressed index and 410 ns in the dynamic index. This is several times slower than in find() queries: find() uses $\mathsf{LF}((v, i), c)$ instead of the slower $\mathsf{LF}(v, i)$, while extract() queries start from the very large record for the endmarker \$. While the direct locate() algorithm also uses $\mathsf{LF}(v, i)$, it benefits from memory locality, as it traverses the same graph region for each position in the query range.

6.3 Haplotype-aware graphs

The typical pruning parameters for a 128-mer GCSA2 index are 4 edge choices in a 16-mer. When building 256-mer indexes, we need to prune more aggressively: 3 edge choices in a 24-mer. This removes more k-mers corresponding to both haplotypes and their recombinations. In the following, graph pruned-k has been pruned with the parameters for a k-mer index, and the reference paths have been restored afterwards. Similarly, unfolded-k is a graph, where the haplotype paths and reference paths have been unfolded after pruning.

We created simplified whole-genome graphs pruned-128, pruned-256, and unfolded-256. The simplification took 3–4 hours for unfolded-256 and slightly less for the other graphs. We then built GCSA2 indexes for both orientations of the simplified graphs. The results can be seen in Table 1. The index for pruned-256 is a few GiB smaller than the others, while the index for unfolded-256 takes a few hours longer to build. Graph pruned-128 contains 90 % of the haplotype k-mers missing from pruned-256 but included in unfolded-256. (Some haplotype k-mers may be recombinations crossing between simple and unfolded regions.) It also contains a large number of additional recombination k-mers not present in unfolded-256.

7 Discussion

We have developed GBWT, a scalable implementation of the graph extension of the PBWT. The earlier implementation used 9.3 hours and 278 GiB of memory to index the 1000GP chromosome 22 using a single thread [21]. In comparison, our implementation takes 4.1 hours and less than 10 GiB of memory using two threads. We also reduced the final index size from

321 MiB to 110 MiB (without haplotype identifiers). By running multiple jobs in parallel, we were able to build a whole-genome GBWT in less than 30 hours on a single system.

Contemporary sequencing projects are sequencing in excess of 100,000 diploid genomes. We intend to scale GBWT to allow working with such large collections, providing a compressed, indexed and searchable representation that should fit into the memory of a single server. Potential applications in genome inference and imputation, as well as for powering population genomic queries, are myriad. The main bottleneck here is construction time. We currently parse the VCF file and find the paths between variant sites once for every 200 samples, which takes the bulk of the time. By parsing the file once and storing the information in a directly usable format, we should be able to double the construction speed.

Storing the haplotype identifiers for locate() queries is another bottleneck. With 5,000 haplotypes, the identifiers use roughly as much space as the GBWT itself. If we increase the number of haplotypes to 50,000, GBWT size should not increase too much, while the identifiers will take 10x more space. We can save space by increasing the distance between stored identifiers, at the expense of increased query times. There is a theoretical proposal for supporting fast locate() queries in space proportional to the size of the run-length encoded BWT [8]. Building the proposed structure for large text collections is still an open problem.

We used the haplotype information in GBWT to simplify VG graphs for k-mer indexing. This allowed us to prune the k-mers corresponding to recombinations more aggressively, while still having all k-mers from the haplotypes in the index. CHOP, the other haplotype-aware graph indexing approach, can only use short-range haplotype information in read mapping. Because VG graphs are connected, we can use the long-range information in the GBWT for mapping long reads and paired-end reads. We will investigate this in a subsequent paper.

References

1. Markus J. Bauer, Anthony J. Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theoretical Computer Science*, 483:134–148, 2013. doi:10.1016/j.tcs.2012.02.002.
2. Michael Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994. URL: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.html.
3. Ho-Leung Chan et al. Compressed indexes for dynamic text collections. *ACM Transactions on Algorithms*, 3(2):21, 2007. doi:10.1145/1240233.1240244.
4. 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.
5. Hannes P. Eggertsson et al. Graphtyper enables population-scale genotyping using pangenome graphs. *Nature Genetics*, 49:1654–1660, 2017. doi:10.1038/ng.3964.
6. Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.
7. Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical Computer Science*, 698:67–78, 2017. doi:10.1016/j.tcs.2017.06.016.
8. Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proc. ALENEX 2018*, pages 1459–1477. SIAM, 2018. doi:10.1137/1.9781611975031.96.
9. Erik Garrison et al. Sequence variation aware genome references and read mapping with the variation graph toolkit. bioRxiv, 2017. doi:10.1101/234856.

10 Simon Gog et al. From theory to practice: Plug and play with succinct data structures. In *Proc. SEA 2014*, volume 8504 of *LNCS*, pages 326–337. Springer, 2014. doi:10.1007/978-3-319-07959-2_28.

11 Wing-Kai Hon et al. A space and time efficient algorithm for constructing compressed suffix arrays. *Algorithmica*, 48(1):23–36, 2007. doi:10.1007/s00453-006-1228-8.

12 Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):i361–i370, 2013. doi:10.1093/bioinformatics/btt215.

13 Songbo Huang et al. Indexing similar DNA sequences. In *Proc. AAIM 2010*, volume 6124 of *LNCS*, pages 180–190. Springer, 2010. doi:10.1007/978-3-642-14355-7_19.

14 Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012. doi:10.1093/bioinformatics/bts280.

15 Heng Li. Fast construction of FM-index for long sequence reads. *Bioinformatics*, 30(22):3274–3275, 2014. doi:10.1093/bioinformatics/btu541.

16 Sorina Maciuca et al. A natural encoding of genetic variation in a Burrows-Wheeler transform to enable mapping and genome inference. In *Proc. WABI 2016*, volume 9838 of *LNCS*, pages 222–233. Springer, 2016. doi:10.1007/978-3-319-43681-4_18.

17 Tom O. Mokveld et al. CHOP: Haplotype-aware path indexing in population graphs. bioRxiv, 2018. doi:10.1101/305268.

18 Veli Mäkinen et al. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010. doi:10.1089/cmb.2009.0169.

19 Joong Chae Na et al. FM-index of alignment: A compressed index for similar strings. *Theoretical Computer Science*, 638:159–170, 2016. doi:10.1016/j.tcs.2015.08.008.

20 Joong Chae Na et al. FM-index of alignment with gaps. *Theoretical Computer Science*, 710(148-157), 2018. doi:10.1016/j.tcs.2017.02.020.

21 Adam Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional Burrows–Wheeler transform and its applications. *Algorithms for Molecular Biology*, 12:18, 2017. doi:10.1186/s13015-017-0109-9.

22 Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proc. ALENEX 2007*, pages 60–70. SIAM, 2007. doi:10.1137/1.9781611972870.6.

23 Goran Rakocevic et al. Fast and accurate genomic analyses using genome graphs. bioRxiv, 2017. doi:10.1101/194530.

24 Korbinian Schneeberger et al. Simultaneous alignment of short reads against multiple genomes. *Genome Biology*, 10(9):R98, 2009. doi:10.1186/gb-2009-10-9-r98.

25 Jouni Sirén. Indexing variation graphs. In *Proc. ALENEX 2017*, pages 13–27. SIAM, 2017. doi:10.1137/1.9781611974768.2.

26 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014. doi:10.1109/TCBB.2013.2297101.

27 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–64, 2015. doi:10.1038/nature15393.

28 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018. doi:10.1093/bib/bbw089.

Reconciling Multiple Genes Trees via Segmental Duplications and Losses

Riccardo Dondi
Dipartimento di Filosofia, Lettere, Comunicazione, Università degli Studi di Bergamo, Bergamo, Italy,
riccardo.dondi@unibg.it

Manuel Lafond
Department of Computer Science, Université de Sherbrooke, Québec, Canada,
manuel.lafond@USherbrooke.ca

Celine Scornavacca
ISEM, CNRS, Université de Montpellier, IRD, EPHE, Montpellier, France,
celine.scornavacca@umontpellier.fr

Abstract

Reconciling gene trees with a species tree is a fundamental problem to understand the evolution of gene families. Many existing approaches reconcile each gene tree independently. However, it is well-known that the evolution of gene families is interconnected. In this paper, we extend a previous approach to reconcile a set of gene trees with a species tree based on segmental macro-evolutionary events, where segmental duplication events and losses are associated with cost δ and λ, respectively. We show that the problem is polynomial-time solvable when $\delta \leq \lambda$ (via LCA-mapping), while if $\delta > \lambda$ the problem is NP-hard, even when $\lambda = 0$ and a single gene tree is given, solving a long standing open problem on the complexity of the reconciliation problem. On the positive side, we give a fixed-parameter algorithm for the problem, where the parameters are δ/λ and the number d of segmental duplications, of time complexity $O(\lceil \frac{\delta}{\lambda} \rceil^d \cdot n \cdot \frac{\delta}{\lambda})$. Finally, we demonstrate the usefulness of this algorithm on two previously studied real datasets: we first show that our method can be used to confirm or refute hypothetical segmental duplications on a set of 16 eukaryotes, then show how we can detect whole genome duplications in yeast genomes.

2012 ACM Subject Classification Applied computing → Computational biology, Theory of computation → Fixed parameter tractability, Theory of computation → Problems, reductions and completeness

Keywords and phrases Gene trees/species tree reconciliation, phylogenetics, computational complexity, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.5

Related Version A full version of the paper is available at `https://arxiv.org/abs/1806.03988`.

Acknowledgements The authors would like to thank Mukul Bansal for providing the eukaryotes data set for the experiments section.

1 Introduction

It is nowadays well established that the evolution of a gene family can differ from that of the species containing these genes. This can be due to quite a number of different reasons, including gene duplication, gene loss, horizontal gene transfer or incomplete lineage sorting,

to only name a few [17]. While this discongruity between the gene phylogenies (the *gene trees*) and the species phylogeny (the *species tree*) complicates the process of reconstructing the latter from the former, every cloud has a silver lining: "plunging" gene trees into the species tree and analyzing the differences between these topologies, one can gather hints to unveil the macro-evolutionary events that shaped gene evolution. This is the rationale behind the *species tree-gene tree reconciliation*, a concept introduced in [9] and first formally defined in [18]. Gathering intelligence on these macro-evolutionary events permits us to better understand the mechanisms of evolution with applications ranging from orthology detection [14, 21] to ancestral genome reconstruction [7], and recently in dating phylogenies [5, 4].

It is well-known that the evolution of gene families is interconnected. However, in current pipelines, each gene tree is reconciled independently with the species tree, even when posterior to the reconciliation phase the genes are considered as related, e.g. [7]. A more pertinent approach would be to reconcile the set of gene trees at once and consider *segmental* macro-evolutionary events, i.e. that concern a chromosome segment instead of a single gene.

Some work has been done in the past to model segmental gene duplications and three models have been considered: the EC (*Episode Clustering*) problem, the ME (*Minimum Episodes*) problem [10, 1], and the MGD (*Multiple Gene Duplication*) problem [8]. The EC and MGD problems both aim at clustering duplications together by minimizing the number of locations in the species tree where at least one duplication occurred, with the additional requirement that a cluster cannot contain two gene duplications from a same gene tree in the MGD problem. The ME problem is more biologically-relevant, because it aims at minimizing the actual number of segmental duplications (more details in Section 2.3). Most of the exact solutions proposed for the ME problem [1, 15, 20] deal with a constrained version, since the possible mappings of a gene tree node are limited to given intervals, see for example [1, Def. 2.4]. In [20], a simple $O^*(2^k)$ time algorithm is presented for the unconstrained version (here O^* hides polynomial factors), where k is the number of speciation nodes that have descending duplications under the LCA-mapping This shows that the problem is fixed-parameter tractable (FPT) in k. But since the LCA-mapping maximizes the number of speciation nodes, there is no reason to believe that k is a small parameter, and so more practical FPT algorithms are needed.

In this paper, we extend the unconstrained ME model to gene losses and provide a variety of new algorithmic results. We allow weighing segmental duplication events and loss events by separate costs δ and λ, respectively. We show that if $\delta \leq \lambda$, then an optimal reconciliation can be obtained by reconciling each gene tree separately under the usual LCA-mapping, even in the context of segmental duplications. On the other hand, we show that if $\delta > \lambda$ and both costs are given, reconciling a set of gene trees while minimizing segmental gene duplications and gene losses is NP-hard. The hardness also holds in the particular case that we ignore losses, i.e. when $\lambda = 0$. This solves a long standing open question on the complexity of the reconciliation problem under this model (in [1], the authors already said *"it would be interesting to extend the [...] model of Guigó et al. (1996) by relaxing the constraints on the possible locations of gene duplications on the species tree"*. The question is stated as still open in [20]). The hardness holds also when only a single gene tree is given. On the positive side, we describe an algorithm that is practical when δ and λ are not too far apart. More precisely, we show that multi-gene tree reconciliation is fixed-parameter tractable in the ratio δ/λ and the number d of segmental duplications, and can be solved in time $O(\lceil \frac{\delta}{\lambda} \rceil^d \cdot n \cdot \frac{\delta}{\lambda})$. The algorithm has been implemented and tested and is

available[1] at `https://github.com/manuellafond/Multrec`. We first evaluate the potential of multi-gene reconciliation on a set of 16 eukaryotes, and show that our method can find scenarios with less duplications than other approaches. While some previously identified segmental duplications are confirmed by our results, it casts some doubt on others as they do not occur in our optimal scenarios. We then show how the algorithm can be used to detect whole genome duplications in yeast genomes. Further work includes incorporating in the model segmental gene losses and segmental horizontal gene transfers, with a similar flavor than the heuristic method discussed in [6].

2 Preliminaries

2.1 Basic notions

For our purposes, a *rooted phylogenetic tree* $T = (V(T), E(T))$ is an oriented tree, where $V(T)$ is the set of nodes, $E(T)$ is the set of arcs, all oriented away from $r(T)$, the root. Unless stated otherwise, all trees in this paper are rooted phylogenetic trees. A *forest* $F = (V(F), E(F))$ is a directed graph in which every connected component is a tree. Denote by $t(F)$ the set of trees of F that are formed by its connected components. Note that a tree is itself a forest. In what follows, we shall extend the usual terminology on trees to forests.

For an arc (x, y) of F, we call x the *parent* of y, and y a *child* of x. If there exists a path that starts at x and ends at y, then x is an *ancestor* of y and y is a *descendant* of x. We say y is a *proper* descendant of x if $y \neq x$, and then x is a proper ancestor of y. This defines a partial order denoted by $y \leq_F x$, and $y <_F x$ if $x \neq y$ (we may omit the F subscript if clear from the context). If none of $x \leq y$ and $y \leq x$ holds, then x and y are *incomparable*. The set of children of x is denoted $ch(x)$ and its parent x is denoted $par(x)$ (which is defined to be x if x itself is a root of a tree in $t(F)$). For some integer $k \geq 0$, we define $par^k(x)$ as the k-th parent of x. Formally, $par^0(x) = par(x)$ and $par^k(x) = par(par^{k-1}(x))$ for $k > 0$. The number of children $|ch(x)|$ of x is called the *out-degree* of x. Nodes with no children are *leaves*, all others are *internal nodes*. The set of leaves of a tree F is denoted by $L(F)$. The leaves of F are bijectively labeled by a set $\mathcal{L}(F)$ of labels. A forest is *binary* if $|ch(x)| = 2$ for all internal nodes x. Given a set of nodes X that belong to the same tree $T \in t(F)$, the *lowest common ancestor* of X, denoted $LCA_F(X)$, is the node z that satisfies $x \leq z$ for all $x \in X$ and such that no child of z satisfies this property. We leave $LCA_F(X)$ undefined if no such node exists (when elements of X belong to different trees of $t(F)$). We may write $LCA_F(x, y)$ instead of $LCA_F(\{x, y\})$. The *height* of a forest F, denoted $h(F)$, is the number of nodes of a longest directed path from a root to a leaf in a tree of F (note that the height is sometimes defined as the number of arcs on such a path - here we use the number of nodes instead). Observe that since a tree is a forest, all the above notions also apply on trees.

2.2 Reconciliations

A reconciliation usually involves two rooted phylogenetic trees, a *gene tree* G and a *species tree* S, which we always assume to be both binary. In what follows, we will instead define reconciliation between a gene forest $\mathcal{G}$ and a species tree. Here $\mathcal{G}$ can be thought of as a set of gene trees. Each leaf of $\mathcal{G}$ represents a distinct extant gene, and $\mathcal{G}$ and S are related by a function $s : \mathcal{L}(\mathcal{G}) \to \mathcal{L}(S)$, which means that each extant gene belongs to an extant species.

[1] To our knowledge, this is the first publicly available reconciliation software for segmental duplications.

Note that s does not have to be injective (in particular, several genes from a same gene tree G of $\mathcal{G}$ can belong to the same species) or surjective (some species may not contain any gene of $\mathcal{G}$). Given $\mathcal{G}$ and S, we will implicitly assume the existence of the function s.

In a $\mathbb{DL}$ reconciliation, each node of $\mathcal{G}$ is associated to a node of S and an event – a speciation ($\mathbb{S}$), a duplication ($\mathbb{D}$) or a contemporary event ($\mathbb{C}$) – under some constraints. A contemporary event $\mathbb{C}$ associates a leaf u of $\mathcal{G}$ with a leaf x of S such that $s(u) = x$. A speciation in a node u of $\mathcal{G}$ is constrained to the existence of two separated paths from the mapping of u to the mappings of its two children, while the only constraint given by a duplication event is that evolution of $\mathcal{G}$ cannot go back in time. More formally:

▶ **Definition 1** (Reconciliation)**.** Given a gene forest $\mathcal{G}$ and a species tree S, a *reconciliation* between $\mathcal{G}$ and S is a function α that maps each node u of $\mathcal{G}$ to a pair $(\alpha_r(u), \alpha_e(u))$ where $\alpha_r(u)$ is a node of $V(S)$ and $\alpha_e(u)$ is an event of type $\mathbb{S}, \mathbb{D}$ or $\mathbb{C}$, such that:

1. if u is a leaf of $\mathcal{G}$, then $\alpha_e(u) = \mathbb{C}$ and $\alpha_r(u) = s(u)$;
2. if u is an internal node of $\mathcal{G}$ with children u_1, u_2, then exactly one of following cases holds:
   - $\alpha_e(u) = \mathbb{S}$, $\alpha_r(u) = LCA_S(\alpha_r(u_1), \alpha_r(u_2))$ and $\alpha_r(u_1), \alpha_r(u_2)$ are incomparable;
   - $\alpha_e(u) = \mathbb{D}$, $\alpha_r(u_1) \leq \alpha_r(u)$ and $\alpha_r(u_2) \leq \alpha_r(u)$

Note that if $\mathcal{G}$ consists of one tree, this definition coincides with the usual one given in the literature (first formally defined in [18]). We say that α is an *LCA-mapping* if, for each internal node $u \in V(\mathcal{G})$ with children u_1, u_2, $\alpha_r(u) = LCA_S(\alpha_r(u_1), \alpha_r(u_2))$. Note that there may be more than one LCA-mapping, since the $\mathbb{S}$ and $\mathbb{D}$ events on internal nodes can vary. The number of duplications of α, denoted by $d(\alpha)$ is the number of nodes u of $\mathcal{G}$ such that $\alpha_e(u) = \mathbb{D}$. For counting the losses, first define for $y \leq x$ the distance $dist(x, y)$ as the number of arcs on the path from x to y. Then, for every internal node u with children $\{u_1, u_2\}$, the number of losses associated with u in a reconciliation α, denoted by $l_\alpha(u)$, is defined as follows:

- if $\alpha_e(u) = \mathbb{S}$, then $l_\alpha(u) = dist(\alpha_r(u), \alpha_r(u_1)) + dist(\alpha_r(u), \alpha_r(u_2)) - 2$;
- if $\alpha_e(u) = \mathbb{D}$, then $l_\alpha(u) = dist(\alpha_r(u), \alpha_r(u_1)) + dist(\alpha_r(u), \alpha_r(u_2))$.

The number of losses of a reconciliation α, denoted by $l(\alpha)$, is the sum of $l_\alpha(\cdot)$ for all internal nodes of $\mathcal{G}$. The usual cost of α, denoted by $cost(\alpha)$, is $d(\alpha) \cdot \delta + l(\alpha) \cdot \lambda$ [16], where δ and λ are respectively the cost of a duplication and a loss event (it is usually assumed that speciations do not incur cost). A *most parsimonious reconciliation*, or MPR, is a reconciliation α of minimum cost. It is not hard to see that finding such an α can be achieved by computing a MPR for each tree in $t(\mathcal{G})$ separately. This MPR is the unique LCA-mapping α in which $\alpha_e(u) = \mathbb{S}$ whenever it is allowed according to Definition 1 [3].

2.3 Reconciliation with segmental duplications

Given a reconciliation α for $\mathcal{G}$ in S, and given $s \in V(S)$, write $D(\mathcal{G}, \alpha, s) = \{u \in V(\mathcal{G}) : \alpha_e(u) = \mathbb{D} \text{ and } \alpha_r(u) = s\}$ for the set of duplications of $\mathcal{G}$ mapped to s. We define $\mathcal{G}[\alpha, s]$ to be the subgraph of $\mathcal{G}$ induced by the nodes in $D(\mathcal{G}, \alpha, s)$. Note that $\mathcal{G}[\alpha, s]$ is a forest.

Here we want to tackle the problem of reconciling several gene trees at the same time and counting segmental duplications only once. Given a set of duplications nodes $\mathcal{D} \in V(\mathcal{G})$ occurring in a given node s of the species tree, it is easy to see that the minimum the number of segmental duplications associated with s is the minimal number of parts in a partition of $\mathcal{D}$ in which each part does not contain comparable nodes. See Figure 1.(4) for an example. This number coincides [1] with $h_\alpha(s) := h(\mathcal{G}[\alpha, s])$, i.e. the height of the forest of the duplications in s. Now, denote $\hat{d}(\alpha) = \sum_{s \in V(S)} h_\alpha(s)$. For instance in Figure 1, under the mapping μ in

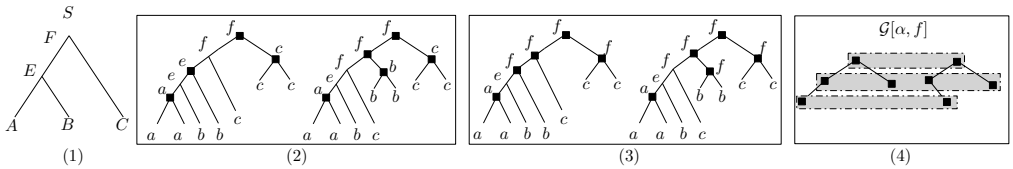


Figure 1 (1) A species tree S. (2) A gene forest $\mathcal{G}$ with two gene trees reconciled under the MPR that we denote μ. The nodes are labeled by the lowercase name of the species they are mapped to. Black squares indicate duplication nodes. Losses are not shown. (3) The same forest $\mathcal{G}$ but with another reconciliation α for the internal nodes. (4) The forest $\mathcal{G}[\alpha, f]$, along with a partition into (possible) segmental duplications.

(2), we have $\hat{d}(\mu) = 6$, because $h_\mu(s) = 1$ for $s \in \{A, B, C, E\}$ and $h_\mu(F) = 2$. But under the mapping α in (3), $\hat{d}(\alpha) = 4$, since $h_\alpha(A) = 1$ and $h_\alpha(F) = 3$. Note that this does not consider losses though – the α mapping has more losses than μ.

The cost of α is $cost^{SD}(\mathcal{G}, S, \alpha) = \delta \cdot \hat{d}(\alpha) + \lambda \cdot l(\alpha)$. If $\mathcal{G}$ and S are unambiguous, we may write $cost^{SD}(\alpha)$. We have the following problem :

Most Parsimonious Reconciliation of a Set of Trees with Segmental Duplications (MPRST-SD)
Instance: a species tree S, a gene forest $\mathcal{G}$, costs δ for duplications and λ for losses.
Output: a reconciliation α of $\mathcal{G}$ in S such that $cost^{SD}(\mathcal{G}, S, \alpha)$ is minimum.

Note that, when $\lambda = 0$, $cost^{SD}$ coincides with the unconstrained ME score defined in [20] (where it is called the FHS model).

2.4 Properties of multi-gene reconciliations

We finish this section with some additional terminology and general properties of multi-gene reconciliations that will be useful throughout the paper. The next basic result states that in a reconciliation α, we should set the events of internal nodes to $\mathbb{S}$ whenever it is allowed.

▶ **Lemma 2.** *Let α be a reconciliation for $\mathcal{G}$ in S, and let $u \in V(\mathcal{G})$ such that $\alpha_e(u) = \mathbb{D}$. Let α' be identical to α, with the exception that $\alpha'_e(u) = \mathbb{S}$, and suppose that α' satisfies the requirements of Definition 1. Then $cost^{SD}(\alpha') \leq cost^{SD}(\alpha)$.*

Proof. Observe that changing $\alpha_e(u)$ from $\mathbb{D}$ to $\mathbb{S}$ cannot increase $\hat{d}(\alpha)$. Moreover, as $dist(\alpha'_r(u), \alpha'_r(u_1))$ and $dist(\alpha'_r(u), \alpha'_r(u_2))$ are the same as in α for the two children u_1 and u_2 of u, by definition of duplications and losses this decreases the number of losses by 2. Thus $cost^{SD}(\alpha') \leq cost^{SD}(\alpha)$, and this inequality is strict when $\lambda > 0$. ◀

Since we are looking for a most parsimonious reconciliation, by Lemma 2 we may assume that for an internal node $u \in V(\mathcal{G})$, $\alpha_e(u) = \mathbb{S}$ whenever allowed, and $\alpha_e(u) = \mathbb{D}$ otherwise. Therefore, $\alpha_e(u)$ is implicitly defined by the α_r mapping. To alleviate notation, we will treat α as simply as a mapping from $V(\mathcal{G})$ to $V(S)$ and thus write $\alpha(u)$ instead of $\alpha_r(u)$. We will assume that the events $\alpha_e(u)$ can be deduced from this mapping α by Lemma 2.

Therefore, treating α as a mapping, we will say that α is *valid* if for every $v \in V(\mathcal{G})$, $\alpha(v) \geq \alpha(v')$ for all descendants v' of v. We denote by $\alpha[v \to s]$ the mapping obtained from α by remapping $v \in V(\mathcal{G})$ to $s \in V(S)$, i.e. $\alpha[v \to s](w) = \alpha(w)$ for every $w \in V(\mathcal{G}) \setminus \{v\}$, and $\alpha[v \to s](v) = s$. Since we are assuming that $\mathbb{S}$ and $\mathbb{D}$ events can be deduced from α, the LCA-mapping is now unique: we denote by $\mu : V(\mathcal{G}) \to V(S)$ the LCA-mapping, defined as $\mu(v) = s(v)$ if $v \in L(\mathcal{G})$, and otherwise $\mu(v) = LCA_S(\mu(v_1), \mu(v_2))$, where v_1 and v_2 are the children of v. Note that for any valid reconciliation α, we have $\alpha(v) \geq \mu(v)$ for all $v \in V(\mathcal{G})$. We also have the following, which will be useful to establish our results.

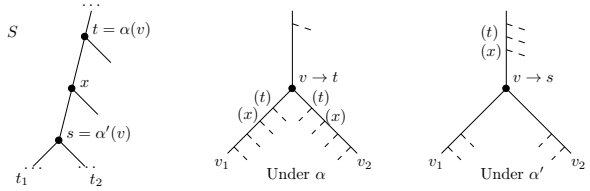


Figure 2 The Shift-down lemma in action. Here $t = par^2(s)$, and remapping v from t to s saves 2 losses – 4 losses are saved below v and 2 are added above.

▶ **Lemma 3.** *Let α be a mapping from $\mathcal{G}$ to S. If $\alpha(v) > \mu(v)$, then v is a $\mathbb{D}$ node under α.*

Proof. Let v_1 and v_2 be the two children of v. If $\alpha(v) \neq LCA_S(\alpha(v_1), \alpha(v_2))$, then v must be a duplication, by the definition of $\mathbb{S}$ events. The same holds if $\alpha(v_1)$ and $\alpha(v_2)$ are not incomparable. Thus assume $\alpha(v) = LCA_S(\alpha(v_1), \alpha(v_2)) > \mu(v)$ and that $\alpha(v_1)$ and $\alpha(v_2)$ are incomparable. This implies that one of $\alpha(v_1)$ or $\alpha(v_2)$ is incomparable with $\mu(v)$, say $\alpha(v_1)$ w.l.o.g.. But $\mu(v_1) \leq \alpha(v_1)$, implying that $\mu(v_1)$ is also incomparable with $\mu(v)$, a contradiction to the definition of $\mu = LCA_S(\mu(v_1), \mu(v_2))$. ◀

▶ **Lemma 4.** *Let α be a mapping from $\mathcal{G}$ to S, and let $v \in V(\mathcal{G})$. Suppose that there is some proper descendant v' of v such that $\alpha(v') \geq \mu(v)$. Then v is a duplication under α.*

Proof. If $\alpha(v) = \mu(v)$, we get $\mu(v) \leq \alpha(v') \leq \alpha(v) = \mu(v)$, and so $\alpha(v') = \mu(v)$. We must then have $\alpha(v'') = \mu(v)$ for every node v'' on the path between v' and v. In particular, v has a child v_1 with $\alpha(v) = \alpha(v_1)$ and thus v is a duplication. If instead $\alpha(v) > \mu(v)$, then v is a duplication by Lemma 3. ◀

The *Shift-down lemma* will prove very useful to argue that we should shift mappings of duplications down when possible, as it saves losses (see Figure 2). For future reference, do note however that this may increase the height of some duplication forest $\mathcal{G}[\alpha, s]$.

▶ **Lemma 5** (Shift-down lemma)**.** *Let α be a mapping from $\mathcal{G}$ to S, let $v \in V(\mathcal{G})$, let $s \in V(S)$ and $k > 0$ be such that $par^k(s) = \alpha(v)$. Suppose that $\alpha[v \to s]$ is a valid mapping. Then $l(\alpha[v \to s]) \leq l(\alpha) - k$.*

Proof. Let v_1 and v_2 be the children of v, and denote $t := \alpha(v), t_1 := \alpha(v_1)$ and $t_2 := \alpha(v_2)$. Moreover denote $\alpha' := \alpha[v \to s]$. Let P be the set of nodes that appear on the path between s and t, excluding s but including t (note that s is a proper descendant of t but an ancestor of both t_1 and t_2, by the validity of α'). For instance in Figure 2, $P = \{x, t\}$. Observe that $|P| = k$. Under α, there is a loss for each node of P on both the (v, v_1) and (v, v_2) branches. (noting that v is a duplication by Lemma 3). These $2k$ losses are not present under α'. On the other hand, there are at most k losses that are present under α' but not under α, which consist of one loss for each node of P on the $(par(v), v)$ branch (in the case that v is not the root of its tree - otherwise, no such loss occurs). This proves that $l(\alpha') \leq l(\alpha) - k$. ◀

An illustration of the Shift-down lemma can be found in Figure 2.

3 The computational complexity of the MPRST-SD problem

We separate the study of the complexity of the MPRST-SD problem into two subcases: when $\lambda \geq \delta$ and when $\lambda < \delta$.

3.1 The case of $\lambda \geq \delta$.

The following theorem states that, when $\lambda \geq \delta$, the MPR (ie the LCA-mapping) is a solution to the MPRST-SD problem.

▶ **Theorem 6.** *Let $\mathcal{G}$ and S be an instance of* MPRST-SD*, and suppose that $\lambda \geq \delta$, Then the LCA-mapping μ is a reconciliation of minimum cost for $\mathcal{G}$ and S. Moreover if $\lambda > \delta$, μ is the unique reconciliation of minimum cost.*

Proof. Let α be a mapping of $\mathcal{G}$ into S of minimum cost. Let $v \in V(\mathcal{G})$ be a minimal node of $\mathcal{G}$ with the property that $\alpha(v) \neq \mu(v)$ (i.e. all proper descendants v' of v satisfy $\alpha(v) = \mu(v)$). Note that v must exists since, for every leaf $l \in \mathcal{L}(\mathcal{G})$, we have $\alpha(l) = \mu(l)$. Because $\alpha(v) \geq \mu(v)$, it follows that $\alpha(v) > \mu(v)$. Denote $s = \mu(v)$ and $t = \alpha(v)$. Then there is some $k \geq 1$ such that $t = par^k(s)$. Consider the mapping $\alpha' = \alpha[v \to s]$. This possibly increases the sum of duplications by 1, so that $\hat{d}(\alpha') \leq \hat{d}(\alpha) + 1$. But by the Shift-down lemma, $l(\alpha') \leq l(\alpha) - 1$. Thus we have at most one duplication but save at least one loss.

If $\lambda > \delta$, this contradicts the optimality of α, implying that v cannot exist and thus that $\alpha = \mu$. This proves the uniqueness of μ in this case.

If $\delta = \lambda$, then $\delta\hat{d}(\alpha') + \lambda l(\alpha') \leq \delta\hat{d}(\alpha) + \lambda l(\alpha)$. By applying the above transformation successively on the minimal nodes v that are not mapped to $\mu(v)$, we eventually reach the LCA-mapping μ with an equal or better cost than α. ◀

3.2 The case of $\delta > \lambda$.

We show that, in contrast with the $\lambda \geq \delta$ case, the MPRST-SD problem is NP-hard when $\delta > \lambda$ and the costs are given as part of the input. More specifically, we show that the problem is NP-hard when one only wants to minimize the sum of duplication heights, i.e. $\lambda = 0$. Note that if $\lambda > 0$ but is small enough, the effect will be the same and the hardness result also holds – for instance, putting $\delta = 1$ and say $\lambda < \frac{1}{2|V(\mathcal{G})||V(S)|}$ ensures that even if a maximum number of losses appears on every branch of $\mathcal{G}$, it does not even amount to the cost of one duplication. The hardness proof is quite technical, and we refer the interested reader to the Appendix of the arXiv version of the paper (`https://arxiv.org/abs/1806.03988`) for the details.

We briefly outline the main ideas of the reduction. The reduction is from the Vertex Cover problem, where we are given a graph G and must find a subset of vertices $V' \subseteq V(G)$ of minimum size such that each edge has at least one endpoint in V'. The species tree S and the forest $\mathcal{G}$ are constructed so that, for each vertex $v_i \in V(G)$, there is a gene tree A_i in $\mathcal{G}$ with a long path of duplications, all of which could either be mapped to a species called y_i or another species z_i. We make it so that mapping to y_i introduces one more duplication than mapping to z_i, hence we have to "pay" for each y_i. We also have a gene tree C_h in $\mathcal{G}$ for each edge $e_h \in E(G)$, with say v_i and v_j being the endpoints of e_h. In C_h, there is a large set of duplications $\mathcal{D}$ under the LCA-mapping μ. We make it so that if we either mapped the duplications in A_i or A_j to y_i or y_j, respectively, then we may map all the $\mathcal{D}$ nodes to y_i or y_j without adding more duplications. However if we did not choose y_i nor y_j, then it will not be possible to remap the $\mathcal{D}$ nodes without incurring a large duplication cost. Therefore, the goal becomes to choose a minimum number of y_i's from the A_i trees so that for each edge $e_h = \{v_i, v_j\}$, one of y_i or y_j is chosen for the tree C_h. This establishes the correspondence with the vertex cover instance.

▶ **Theorem 7.** *The* MPRST-SD *problem is NP-hard for $\lambda = 0$ and for given $\delta > \lambda$.*

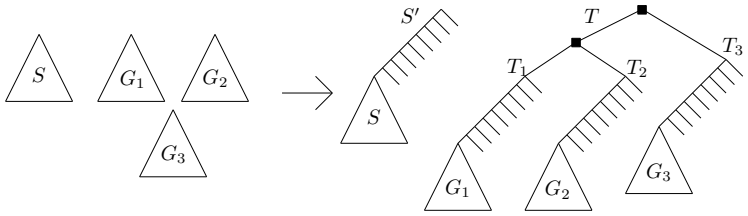


Figure 3 The construction of S' and T from S and the set of gene trees $G_1, \ldots, G_k$ (here $k = 3$). The black squares indicate the path of $k-1$ duplications that must be mapped to $r(S')$.

The above hardness supposes that δ and λ can be arbitrarily far apart. This leaves open the question of whether MPRST-SD is NP-hard when δ and λ are fixed constants – in particular when $\delta = 1 + \epsilon$ and $\lambda = 1$, where $\epsilon < 1$ is some very small constant. We end this section by showing that the above hardness result persists even if only one gene tree is given. The idea is to reduce from the MPRST-SD show hard just above. Given a species tree S and a gene forest $\mathcal{G}$, we make $\mathcal{G}$ a single tree by incorporating a large number of speciations (under μ) above the root of each tree of $\mathcal{G}$ (modifying S accordingly), then successively joining the roots of two trees of $\mathcal{G}$ under a common parent until $\mathcal{G}$ has only one tree.

▶ **Theorem 8.** *The* MPRST-SD *problem is NP-hard for $\lambda = 0$ and for given $\delta > \lambda$, even if only one gene tree is given as input.*

Proof. We reduce from the MPRST-SD problem in which multiple trees are given. We assume that $\delta = 1$ and $\lambda = 0$ and only consider duplications – we use the same argument as before to justify that the problem is NP-hard for very small λ. Let S be the given species tree and $\mathcal{G}$ be the given gene forest. As we are working with the decision version of MPRST-SD, assume we are given an integer t and asked whether $cost^{SD}(\mathcal{G}, S, \alpha) \leq t$ for some α. Denote $n = |\mathcal{L}(\mathcal{G})|$ and let $G_1, \ldots, G_k$ be the $k > 1$ trees of $\mathcal{G}$. We construct a corresponding instance of a species tree S' and a single gene tree T as follows (the construction is illustrated in Figure 3). Let S' be a species tree obtained by adding $2(t + k)$ nodes "above" the root of S. More precisely, first let C be a caterpillar with $2(t + k)$ internal nodes. Let l be a deepest leaf of C. Obtain S' by replacing l by the root of S. Then, obtain the gene tree T by taking k copies $C_1, \ldots, C_k$ of C, and for each leaf l' of each C_i other than l, put $s(l')$ as the corresponding leaf in S'. Then for each $i \in [k]$, replace the l leaf of C_i by the tree G_i (we keep the leaf mapping s of G_i), resulting in a tree we call T_i. Finally, let T' be a caterpillar with k leaves $h_1, \ldots, h_k$, and replace each h_i by the T_i tree. The resulting tree is T. We show that $cost(\mathcal{G}, S, \alpha) \leq t$ for some α if and only if $cost(T, S', \alpha') \leq t + k - 1$ for some α'.

Notice the following: in any mapping α of T, the $k-1$ internal nodes of the T' caterpillar must be duplications mapped to $r(S')$, so that $h_\alpha(r(S')) \geq k - 1$. Also note that under the LCA mapping μ for T and S', the only duplications other than those $k-1$ mentioned above occur in the G_i subtrees. The ($\Rightarrow$) is then easy to see: given α such that $cost(\mathcal{G}, S, \alpha) \leq t$, we set $\alpha'(v) = \alpha(v)$ for every node v of T that is also in $\mathcal{G}$ (namely the nodes of $G_1, \ldots, G_k$), and set $\alpha'(v) = \mu(v)$ for every other node. This achieves a cost of $t + k - 1$.

As for the ($\Leftarrow$) direction, suppose that $cost^{SD}(T', S', \alpha') \leq t + k - 1$ for some mapping α'. Observe that under the LCA-mapping in T, each root of each G_i subtree has a path of $2(t + k)$ speciations in its ancestors. If any node in a G_i subtree of T is mapped to $r(S')$, then all these speciations become duplications (by Lemma 4), which would contradict $cost^{SD}(T', S', \alpha') \leq t + k - 1$. We may thus assume that no node belonging to a G_i subtree is mapped to $r(S')$. Since $h_{\alpha'}(r(S')) \geq k - 1$, this implies that the restriction of α' to the G_i subtrees has cost at most t.

More formally, consider the mapping α'' from $\mathcal{G}$ to S' in which we put $\alpha''(v) = \alpha'(v)$ for all $v \in V(\mathcal{G})$. Then $cost^{SD}(\mathcal{G}, S', \alpha'') \leq cost^{SD}(T, S', \alpha') - (k-1) \leq t$, because α'' does not contain the top $k-1$ duplications of α', and cannot introduce longer duplication paths than in α'.

We are not done, however, since α'' is a mapping from $\mathcal{G}$ to S', and not from $\mathcal{G}$ to S. Consider the set $Q \subseteq V(\mathcal{G})$ of nodes v of $\mathcal{G}$ such that $\alpha''(v) \in \overline{V(S)} := V(S') \setminus V(S)$. We will remap every such node to $r(S)$ and show that this cannot increase the cost. Observe that if $v \in Q$, then every ancestor of v in $\mathcal{G}$ is also in Q. Also, every node in Q is a duplication (by invoking Lemma 3).

Consider the mapping α^* from $\mathcal{G}$ to S' in which we put $\alpha^*(v) = \alpha''(v)$ for all $v \notin Q$, and $\alpha^*(v) = r(S)$ for all $v \in Q$. It is not difficult to see that α^* is valid.

Now, $h_{\alpha^*}(s) = 0$ for all $s \in \overline{V(S)}$ and $h_{\alpha^*}(s) = h_{\alpha''}(s)$ for all $s \in V(S) \setminus \{r(S)\}$. Moreover, the height of the $r(S)$ duplications under α^* cannot be more than the height of the forest induced by Q and the duplications mapped to $r(S)$ under α''. In other words,

$$\begin{aligned}
h_{\alpha^*}(r(S)) &\leq \max_{G_i}(h_{\alpha''}(r(S)) + \sum_{s' \in \overline{V(S)}} h(G_i[\alpha'', s'])) \\
&= h_{\alpha''}(r(S)) + \max_{G_i}(\sum_{s' \in \overline{V(S)}} h(G_i[\alpha'', s'])) \\
&\leq h_{\alpha''}(r(S)) + \sum_{s' \in \overline{V(S)}} \max_{G_i}(h(G_i[\alpha'', s'])) \\
&= h_{\alpha''}(r(S)) + \sum_{s' \in \overline{V(S)}} h(\mathcal{G}[\alpha'', s']) \\
&= h_{\alpha''}(r(S)) + \sum_{s' \in \overline{V(S)}} h_{\alpha''}(s')
\end{aligned}$$

Therefore, the sum of duplication heights cannot have increased. Finally, because α^* is a mapping from $\mathcal{G}$ to S, we deduce that $cost^{SD}(\mathcal{G}, S, \alpha^*) \leq cost^{SD}(\mathcal{G}, S', \alpha'') \leq t$, as desired. ◀

4 An FPT algorithm

In this section, we show that for costs $\delta > \lambda$ and a parameter $d > 0$, if there is an optimal reconciliation α of cost $cost^{SD}(\mathcal{G}, S)$ satisfying $\hat{d}(\alpha) \leq d$, then α can be found in time $O(\lceil \frac{\delta}{\lambda} \rceil^d \cdot n \cdot \frac{\delta}{\lambda})$.

In what follows, we allow mappings to be partially defined, and we use the $\perp$ symbol to indicate undetermined mappings. The idea is to start from a mapping in which every internal node is undetermined, and gradually determine those in a bottom-up fashion. We need an additional set of definitions. We will assume that $\delta > \lambda > 0$ (although the algorithm described in this section can solve the $\lambda = 0$ case by setting λ to a very small value).

We say that the mapping $\alpha : V(\mathcal{G}) \to V(S) \cup \{\perp\}$ is a *partial mapping* if $\alpha(l) = s(l)$ for every leaf $l \in \mathcal{L}(\mathcal{G})$, and it holds that whenever $\alpha(v) \neq \perp$, we have $\alpha(v') \neq \perp$ for every descendant v' of v. That is, if a node is determined, then all its descendants also are. This also implies that every ancestor of a $\perp$-node is also a $\perp$-node. A node $v \in V(\mathcal{G})$ is a *minimal $\perp$-node* (under α) if $\alpha(v) = \perp$ and $\alpha(v') \neq \perp$ for each child v' of v. If $\alpha(v) \neq \perp$ for every $v \in V(\mathcal{G})$, then α is called *complete*. Note that if α is partial and $\alpha(v) \neq \perp$, one can already determine whether v is an $\mathbb{S}$ or a $\mathbb{D}$ node, and hence we may say that v is a speciation or a duplication under α. Also note that the definitions of $\hat{d}(\alpha), l(\alpha)$ and $h_\alpha(s)$ extend naturally to a partial mapping α by considering the forest induced by the nodes not mapped to $\perp$.

If α is a partial mapping, we call α' a *completion* of α if α' is complete, and $\alpha(v) = \alpha'(v)$ whenever $\alpha(v) \neq \perp$. Note that such a completion always exists, as in particular one can map every $\perp$-node to the root of S (such a mapping must be valid, since all ancestors of a $\perp$-node are also $\perp$-nodes, which ensures that $r(S) = \alpha'(v) \geq \alpha'(v')$ for every descendant v' of a newly mapped $\perp$-node v). We say that α' is an *optimal completion of* α if $cost^{SD}(\alpha')$ is minimum among every possible completion of α. For a minimal $\perp$-node v with children v_1 and v_2, we denote $\mu_\alpha(v) = LCA_S(\alpha(v_1), \alpha(v_2))$, i.e. the lowest species of S to which v can possibly be mapped to in any completion of α. Observe that $\mu_\alpha(v) \geq \mu(v)$. Moreover, if v is a minimal $\perp$-node, then in any completion α' of α, $\alpha'[v \to \mu_\alpha(v)]$ is a valid mapping. A minimal $\perp$-node v is called a *lowest minimal $\perp$-node* if, for every minimal $\perp$-node w distinct from v, either $\mu_\alpha(v) \leq \mu_\alpha(w)$ or $\mu_\alpha(v)$ and $\mu_\alpha(w)$ are incomparable.

The following Lemma forms the basis of our FPT algorithm, as it allows us to bound the possible mappings of a minimal $\perp$-node.

▶ **Lemma 9.** *Let α be a partial mapping and let v be a minimal $\perp$-node. Then for any optimal completion α^* of α, $\alpha^*(v) \leq par^{\lceil \delta/\lambda \rceil}(\mu_\alpha(v))$.*

Proof. Let α^* be an optimal completion of α and let $\alpha' := \alpha^*[v \to \mu_\alpha(v)]$. Note that $\hat{d}(\alpha') \leq \hat{d}(\alpha^*) + 1$. Now suppose that $\alpha^*(v) > par^{\lceil \delta/\lambda \rceil}(\mu_\alpha(v))$. Then by the Shift-down lemma, $l(\alpha^*) - l(\alpha') > \lceil \delta/\lambda \rceil \geq \delta/\lambda$. Thus $cost^{SD}(\alpha^*) - cost^{SD}(\alpha') > -\delta + \lambda(\delta/\lambda) = 0$. This contradicts the optimality of α^*. ◀

A node $v \in V(\mathcal{G})$ is a *required duplication* (under α) if, in any completion α' of α, v is a duplication under α'. We first show that required duplications are easy to find.

▶ **Lemma 10.** *Let v be a minimal $\perp$-node under α, and let v_1 and v_2 be its two children. Then v is a required duplication under α if and only if $\alpha(v_1) \geq \mu(v)$ or $\alpha(v_2) \geq \mu(v)$.*

Proof. Suppose that $\alpha(v_1) \geq \mu(v)$, and let α' be a completion of α. If $\alpha'(v) = \alpha'(v_1)$, then v is a duplication by definition. Otherwise, $\alpha'(v) > \alpha'(v_1) = \alpha(v_1) \geq \mu(v)$, and v is a duplication by Lemma 3. The case when $\alpha(v_2) \geq \mu(v)$ is identical.

Conversely, suppose that $\alpha(v_1) < \mu(v)$ and $\alpha(v_2) < \mu(v)$. Then $\alpha(v_1)$ and $\alpha(v_2)$ must be incomparable descendants of $\mu(v)$ (because otherwise if e.g. $\alpha(v_1) \leq \alpha(v_2)$, then we would have $\mu(v) = LCA_S(\mu(v_1), \mu(v_2)) \leq LCA_S(\alpha(v_1), \alpha(v_2)) = \alpha(v_2)$, whereas we are assuming that $\alpha(v_2) < \mu(v)$). Take any completion α' of α such that $\alpha'(v) = \mu(v)$. To see that v is a speciation under α', it remains to argue that $\alpha'(v) = \mu(v) = LCA_S(\alpha(v_1), \alpha(v_2))$. Since $\mu(v)$ is an ancestor of both $\alpha(v_1)$ and $\alpha(v_2)$, we have $LCA_S(\alpha(v_1), \alpha(v_2)) \leq \mu(v)$. We also have $\mu(v) = LCA_S(\mu(v_1), \mu(v_2)) \leq LCA_S(\alpha(v_1), \alpha(v_2))$, and equality follows. ◀

Lemma 11 and Lemma 12 allow us to find minimal $\perp$-nodes of $\mathcal{G}$ that are the easiest to deal with, as their mapping in an optimal completion can be determined with certainty.

▶ **Lemma 11.** *Let v be a minimal $\perp$-node under α. If v is not a required duplication under α, then $\alpha^*(v) = \mu_\alpha(v)$ for any optimal completion α^* of α.*

Proof. Let v_1, v_2 be the children of v, and let α^* be an optimal completion of α. Since v is not a required duplication, by Lemma 10 we have $\alpha(v_1) < \mu(v)$ and $\alpha(v_2) < \mu(v)$ and, as argued in the proof of Lemma 10, $\alpha(v_1)$ and $\alpha(v_2)$ are incomparable. We thus have that $\mu_\alpha(v) = \mu(v)$. Then $\alpha^*[v \to \mu(v)]$ is a valid mapping, and v is a speciation under this mapping. Hence $\hat{d}(\alpha^*[v \to \mu(v)]) \leq \hat{d}(\alpha^*)$. Then by the Shift-down lemma, this new mapping has fewer losses, and thus attains a lower cost than α^*. ◀

▶ **Lemma 12.** *Let v be a minimal $\perp$-node under α, and let $\alpha_v := \alpha[v \to \mu_\alpha(v)]$. If $\hat{d}(\alpha) = \hat{d}(\alpha_v)$, then $\alpha^*(v) = \mu_\alpha(v)$ for any optimal completion α^* of α.*

Proof. Let α^* be an optimal completion of α. Denote $s := \mu_\alpha(v)$, and assume that $\alpha^*(v) > s$ (as otherwise, we are done). Let $\alpha' = \alpha^*[v \to s]$. We have that $l(\alpha') < l(\alpha^*)$ by the Shift-down lemma. To prove the Lemma, we then show that $\hat{d}(\alpha') \leq \hat{d}(\alpha^*)$. Suppose otherwise that $\hat{d}(\alpha') > \hat{d}(\alpha^*)$. As only v changed mapping to s to go from α^* to α', this implies that $h_{\alpha'}(s) > h_{\alpha^*}(s)$ because of v. Since under α^*, no ancestor of v is mapped to s, it must be that under α', v is the root of a subtree T of height $h_{\alpha'}(s)$ of duplications in s. Since T contains only descendants of v, it must also be that $h_{\alpha_v}(s) = h_{\alpha'}(s)$ (here α_v is the mapping defined in the Lemma statement). As we are assuming that $h_{\alpha'}(s) > h_{\alpha^*}(s)$, we get $h_{\alpha_v}(s) > h_{\alpha^*}(s)$. This is a contradiction, since $h_{\alpha^*}(s) \geq h_\alpha(s) = h_{\alpha_v}(s)$ (the left inequality because α^* is a completion of α, and the right equality by the choice of α_v). Then $l(\alpha') < l(\alpha^*)$ and $\hat{d}(\alpha') \leq \hat{d}(\alpha^*)$ contradicts the fact that α^* is optimal. ◀

We say that a minimal $\perp$-node $v \in V(\mathcal{G})$ is *easy* (under α) if v falls into one of the cases described by Lemma 11 or Lemma 12. Formally, v is easy if either v is a speciation mapped to $\mu_\alpha(v)$ under any optimal completion of α (Lemma 11), or $\hat{d}(\alpha) = \hat{d}(\alpha[v \to \mu_\alpha(v)])$ (Lemma 12). Our strategy will be to "clean-up" the easy nodes, meaning that we map them to $\mu_\alpha(v)$ as prescribed above, and then handle the remaining non-easy nodes by branching over the possibilities. We say that a partial mapping α is *clean* if every minimal $\perp$-node v satisfies the two following conditions:

[C1] v is not easy;

[C2] for all duplication nodes w (under α with $\alpha(w) \neq \perp$), either $\alpha(w) \leq \mu_\alpha(v)$ or $\alpha(w)$ is incomparable with $\mu_\alpha(v)$.

Roughly speaking, C2 says that all further duplications that may "appear" in a completion of α will be mapped to nodes "above" the current duplications in α. The purpose of C2 is to allow us to create duplication nodes with mappings from the bottom of S to the top. Our goal will be to build our α mapping in a bottom-up fashion in $\mathcal{G}$ whilst maintaining this condition. The next lemma states that if α is clean and some *lowest* minimal $\perp$-node v gets mapped to species s, then v brings with it every minimal $\perp$-node that can be mapped to s.

▶ **Lemma 13.** *Suppose that α is a clean partial mapping, and let α^* be an optimal completion of α. Let v be a lowest minimal $\perp$-node under α, and let $s := \alpha^*(v)$. Then for every minimal $\perp$-node w such that $\mu_\alpha(w) \leq s$, we have $\alpha^*(w) = s$.*

Proof. Denote $\alpha' := \alpha[v \to s]$. Suppose first that $s = \mu_\alpha(v)$. Note that since α is clean, v is not easy, which implies that $h_{\alpha'}(s) = h_\alpha(s) + 1$. Since v is a lowest minimal $\perp$-node, if w is a minimal $\perp$-node such that $\mu_\alpha(w) \leq s$, we must have $\mu_\alpha(w) = s$, as otherwise v would not have the 'lowest' property. Moreover, because v and w are both minimal $\perp$-nodes under the partial mapping α, one cannot be the ancestor of the other and so v and w are incomparable. This implies that mapping w to s under α' cannot further increase $h_{\alpha'}(s)$ (because we already increased it by 1 when mapping v to s). Thus $\hat{d}(\alpha') = \hat{d}(\alpha'[w \to s])$, and w is easy under α' and must be mapped to s by Lemma 12. This proves the $\alpha^*(v) = \mu_\alpha(s)$ case.

Now assume that $s > \mu_\alpha(v)$, and let w be a minimal $\perp$-node with $\mu_\alpha(w) \leq s$. Let us denote $s' := \alpha^*(w)$. If $s' = s$, then we are done. Suppose that $s' < s$, noting that $h_{\alpha^*}(s') > 0$ (because w must be a duplication node, due to α being clean). If $s' = \mu_\alpha(v)$, then w is also a lowest minimal $\perp$-node. In this case, using the arguments from the previous paragraph and swapping the roles of v and w, one can see that v is easy in $\alpha[w \to s']$ and must be mapped

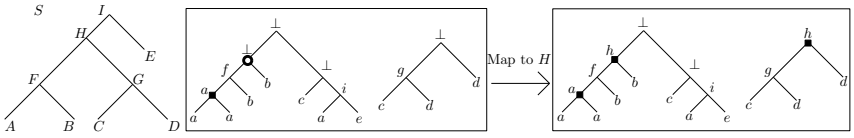


Figure 4 An illustration of one pass through the algorithm. The species tree S is left and $\mathcal{G}$ has two trees (middle) and has partial mapping α (labels are the lowercase of the species). Here α is in a clean state, and the algorithm will pick a lowest minimal $\perp$-node (white circle) and try to map it to, say, H. The forest on the right is the state of α after applying this and cleaning up.

to $s' < s$, a contradiction. Thus assume $s' > \mu_\alpha(v)$. Under α^*, for each child v' of v, we have $\alpha^*(v') \leq \mu_\alpha(v) < s'$, and for each ancestor v'' of v, we have $\alpha^*(v'') \geq \alpha^*(v) = s > s'$. Therefore, by remapping v to s', v is the only duplication mapped to s' among its ancestors and descendants. In other words, because $h_{\alpha^*}(s') > 0$, we have $\hat{d}(\alpha^*[v \to s']) \leq \hat{d}(\alpha^*)$. Moreover by the Shift-down lemma, $l(\alpha^*[v \to s']) < l(\alpha^*)$, which contradicts the optimality of α^*.

The remaining case is $s' > s$. Note that $h_{\alpha^*}(s) > 0$ (because v must be a duplication node, due to α being clean). Since it holds that v is a minimal $\perp$-node, that α is clean and that $s > \mu_\alpha(v)$, it must be the case that α has no duplication mapped to s (by the second property of cleanness). In particular, w has no descendant that is a duplication mapped to s under α (and hence under α^*). Moreover, as $s' = \alpha^*(w) > s$, w has no ancestor that is a duplication mapped to s. Thus $\hat{d}(\alpha^*[w \to s]) \leq \hat{d}(\alpha^*)$, and the Shift-down lemma contradicts the optimality of α^*. This concludes the proof. ◀

We are finally ready to describe our algorithm. We start from a partial mapping α with $\alpha(v) = \perp$ for every internal node v of $\mathcal{G}$. We gradually "fill-up" the $\perp$-nodes of α in a bottom-up fashion, maintaining a clean mapping at each step and ensuring that each decision leads to an optimal completion α^*. To do this, we pick a lowest minimal $\perp$-node v, and "guess" $\alpha^*(v)$ among the $\lceil \delta/\lambda \rceil$ possibilities. This increases some $h_\alpha(s)$ by 1. For each such guess s, we use Lemma 13 to map the appropriate minimal $\perp$-nodes to s, then take care of the easy nodes to obtain another clean mapping. We repeat until we have either found a complete mapping or we have a duplication height higher than d. An illustration of a pass through the algorithm is shown in Figure 4.

Notice that the algorithm assumes that it receives a clean partial mapping α. In particular, the initial mapping α that we pass to the first call should satisfy the two properties of cleanness. To achieve this, we start with a partial mapping α in which every internal node is a $\perp$-node. Then, while there is a minimal $\perp$-node v that is not a required duplication, we set $\alpha(v) = \mu_\alpha(v)$, which makes v a speciation. It is straightforward to see that the resulting α is clean: C1 is satisfied because we cannot make any more minimal $\perp$-nodes become speciations, and we cannot create any duplication node without increasing $cost^{SD}$ because α has no duplication. C2 is met because there are no duplications at all.

See below for the proof of correctness. The complexity follows from the fact that the algorithm creates a search tree of degree $\lceil \delta/\lambda \rceil$ of depth at most d. The main technicality is to show that the algorithm maintains a clean mapping before each recursive call[2].

[2] There is a subtlety to consider here. What we have shown is that if there exists a mapping α of minimum cost $cost^{SD}(\mathcal{G}, S)$ with $\hat{d}(\alpha) \leq d$, then the algorithm finds it. It might be that a reconciliation α satisfying $\hat{d}(\alpha) \leq d$ exists, but that the algorithm returns no solution. This can happen in the case that α is not of cost $cost^{SD}(\mathcal{G}, S)$.

Algorithm 1 FPT algorithm for parameter d.

1: **procedure** SUPERRECONCILE($\mathcal{G}, S, \alpha, d$)
$\mathcal{G}$ is the set of input trees, S is the species tree, α is a *clean* partial mapping, d is the maximum value of $\hat{d}(\alpha)$.
2: **if** $\hat{d}(\alpha) > d$ **then**
3: Return ∞
4: **else if** α is a complete mapping **then**
5: Return $cost^{SD}(\alpha)$
6: **else**
7: Let v be a lowest minimal $\perp$-node
8: $bestCost \leftarrow \infty$
9: **for** s such that $\mu_\alpha(v) \leq s \leq par^{\lceil \delta/\lambda \rceil}(\mu_\alpha(v))$ **do**
10: Let $\alpha' = \alpha[v \rightarrow s]$
11: **for** minimal $\perp$-node $w \neq v$ under α such that $\mu_\alpha(w) \leq s$ **do**
12: Set $\alpha' = \alpha'[w \rightarrow s]$
13: **while** there is a minimal $\perp$-node w that is easy under α' **do**
14: Set $\alpha' = \alpha'[w \rightarrow \mu_{\alpha'}(w)]$
15: $cost \leftarrow superReconcile(\mathcal{G}, S, \alpha', d)$
16: **if** $cost < bestCost$ **then** $bestCost \leftarrow cost$
17: Return $bestCost$

▶ **Theorem 14.** *Algorithm 1 is correct and finds a minimum cost mapping α^* satisfying $\hat{d}(\alpha^*) \leq d$, if any, in time $O(\lceil \frac{\delta}{\lambda} \rceil^d \cdot n \cdot \frac{\delta}{\lambda})$.*

Proof. We show by induction over the depth of the search tree that, in any recursive call made to Algorithm 1 with partial mapping α, the algorithm returns the cost of an optimal completion α^* of α having $\hat{d}(\alpha^*) \leq d$, or ∞ if no such completion exists - assuming that the algorithm receives a clean mapping α as input. Thus in order to use induction, we must also show that at each recursive call done on line 15, α' is a clean mapping. We additionally claim that the search tree created by the algorithm has depth at most d. To show this, we will also prove that every α' sent to a recursive call satisfies $\hat{d}(\alpha') = \hat{d}(\alpha) + 1$.

The base cases of lines 3 - 5 are trivial. For the induction step, let v be the lowest minimal $\perp$-node chosen on line 7. By Lemma 9, if α^* is an optimal completion of α and $s = \alpha^*(v)$, then $\mu_\alpha(v) \leq s \leq par^{\lceil \delta/\lambda \rceil}(\mu_\alpha(v))$. We try all the $\lceil \delta/\lambda \rceil$ possibilities in the for-loop on line 9. The for-loop on line 11 is justified by Lemma 13, and the for-loop on line 13 is justified by Lemma 11 and Lemma 12. Assuming that α' is clean on line 15, by induction the recursive call will return the cost of an optimal completion α^* of α' having $\hat{d}(\alpha^*) \leq d$, if any such completion exists. It remains to argue that for every α' sent to a recursive call on line 15, α' is clean and $\hat{d}(\alpha') = \hat{d}(\alpha) + 1$.

Let us first show that such a α' is clean, for each choice of s on line 9. There clearly cannot be an easy node under α' after line 13, so we must show C2, i.e. that for any minimal $\perp$-node w under α', there is no duplication z under α' satisfying $\alpha'(z) > \mu_{\alpha'}(w)$. Suppose instead that $\alpha'(z) > \mu_{\alpha'}(w)$ for some duplication node z. Let w_0 be a descendant of w that is a minimal $\perp$-node in α (note that $w_0 = w$ is possible). We must have $\mu_{\alpha'}(w) \geq \mu_\alpha(w_0)$. By our assumption, we then have $\alpha'(z) > \mu_{\alpha'}(w) \geq \mu_\alpha(w_0)$. Then z cannot be a duplication under α, as otherwise α itself could not be clean (by C2 applied on z and w_0). Thus z is a newly introduced duplication in α', and so z was a $\perp$-node under α. Note that Algorithm 1 maps $\perp$-nodes of $\mathcal{G}$ one after another, in some order $(z_1, z_2, \ldots, z_k)$. Suppose without loss of generality that z is the first duplication node in this ordering that gets mapped to $\alpha'(z)$.

There are two cases: either $\alpha'(z) \neq s$, or $\alpha'(z) = s$.

Suppose first that $\alpha'(z) \neq s$. Lines 10 and 11 can only map $\perp$-nodes to s, and line 13 either maps speciation nodes, or easy nodes that become duplications. Thus when $\alpha'(z) \neq s$, we may assume that z falls into the latter case, i.e. z is easy before being mapped, so that mapping z to $\alpha'(z)$ does not increase $h_{\alpha'}(\alpha'(z))$. Because z is the first $\perp$-node that gets mapped to $\alpha'(z)$, this is only possible if there was already a duplication z_0 mapped to $\alpha'(z)$ in α. This implies that $\alpha(z_0) = \alpha'(z) > \mu_\alpha(w_0)$, and that α was not clean (by C2 applied on z_0 and w_0). This is a contradiction.

We may thus assume that $\alpha'(z) = s$. This implies $\mu_{\alpha'}(w) < \alpha'(z) = s$. If w was a minimal $\perp$-node in α, it would have been mapped to s on line 11, and so in this case w cannot also be a minimal $\perp$-node in α', as we supposed. If instead w was not a minimal $\perp$-node in α, then w has a descendant w_0 that was a minimal $\perp$-node under α. We have $\mu_\alpha(w_0) \leq \mu_{\alpha'}(w) < s$, which implies that w_0 gets mapped to s on line 11. This makes $\mu_{\alpha'}(w) < s$ impossible, and we have reached a contradiction. We deduce that z cannot exist, and that α' is clean.

It remains to show that $\hat{d}(\alpha') = \hat{d}(\alpha) + 1$. Again, let s be the chosen species on line 9. Suppose first that $s = \mu_\alpha(v)$. Then $h_{\alpha[v \to s]}(s) = h_\alpha(s) + 1$, as otherwise v would be easy under α, contradicting its cleanness. In this situation, as argued in the proof of Lemma 13, each node w that gets mapped to s on line 11 or on line 13 is easy, and thus cannot further increase the height of the duplications in s. If $s > \mu_\alpha(v)$, then $h_{\alpha[v \to s]}(s) = 1 = h_\alpha(s) + 1$, since by cleanness no duplication under α maps to s. Here, each node w that gets mapped on line 11 has no descendant nor ancestor mapped to s, and thus the height does not increase. Noting that remapping easy nodes on line 13 cannot alter the duplication heights, we get in both cases that $\hat{d}(\alpha[v \to s]) = \hat{d}(\alpha) + 1$. This proves the correctness of the algorithm.

As for the complexity, the algorithm creates a search tree of degree $\lceil \delta/\lambda \rceil$ and of depth at most d. Each pass can easily seen to be feasible in time $O(\delta/\lambda \cdot n)$ (with appropriate pre-parsing to compute $\mu_\alpha(v)$ in constant time, and to decide if a node is easy or not in constant time as well), and so the total complexity is $O(\lceil \delta/\lambda \rceil^d n \cdot \frac{\delta}{\lambda})$. ◀

5 Experiments

We used our software to reanalyze a data set of 53 gene trees for 16 eukaryotes presented in [10] and already reanalyzed in [19, 1]. In [1], the authors showed that, if segmental duplications are not accounted for, we get a solution having $\hat{d}$ equal to 9, while their software (ExactMGD) returns a solution with $\hat{d}$ equal to 5. We were able to retrieve the solution with maximum height of 5 fixing $\delta \in [28, 61]$ and $\lambda = 1$, but, as soon as $\delta > 61$, we got a solution with maximum height of 4 where no duplications are placed in the branch leading to the Tetrapoda clade (see [19, Fig. 1]) while the other locations of segmental duplications inferred in [10] are confirmed[3]. This may sow some doubt on the actual existence of a segmental duplication in the LCA of the Tetrapoda clade.

We also reanalyzed the data set of yeast species described in [2]. First, we selected from the data set the 2379 gene trees containing all 16 species and refined unsupported branches using the method described in [12] and implemented in ecceTERA [11] with a bootstrap threshold of 0.9 and $\delta = \lambda = 1$. Using our method with $\delta = 1.5$, $\lambda = 1$ we were able to detect the ancient genome duplication in Saccharomyces cerevisiae already established using synteny information [13], with 216 gene families supporting the event. Other nodes with a

[3] Note that for this data set we used a high value for $\frac{\delta}{\lambda}$ since, because of the sampling strategy, we expect that all relevant genes have been sampled (recall that in ExactMGD, λ is implicitly set to 0).

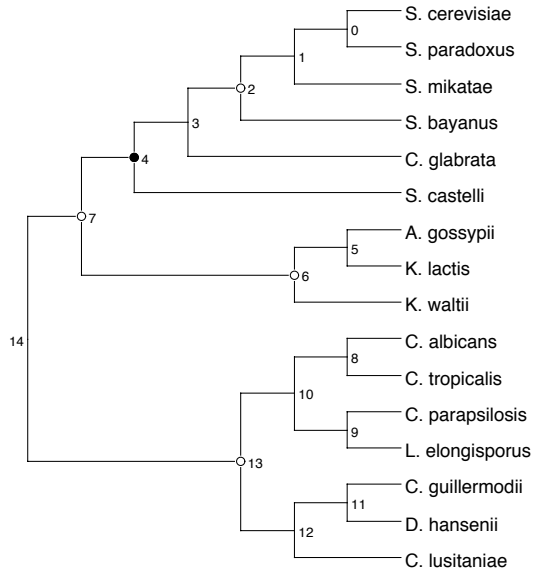


Figure 5 The species tree phylogeny for the yeast data set described in [2]. Numbers at the internal nodes are meaningless and are only used to refer to the nodes in the main text.

signature of segmental duplication are nodes 7, 6, 13 and 2 (refer to Fig. 5) with respectively 190, 157, 148 and 136 gene families supporting the event. It would be interesting to see if the synteny information supports these hypotheses.

6 Conclusion

This work poses a variety of questions that deserve further investigation. The complexity of the problem when δ/λ is a constant remains an open problem. Moreover, our FPT algorithm can handle data sets with a sum of duplication height of about $d = 30$, but in the future, one might consider whether there exist fast approximation algorithms for MPRST-SD in order to attain better scalability. Other future directions include a multivariate complexity analysis of the problem, in order to understand whether it is possible to identify other parameters that are small in practice. Finally, we plan to extend the experimental analysis to other data sets, for instance for the detection of whole genome duplications in plants.

References

1 Mukul S Bansal and Oliver Eulenstein. The multiple gene duplication problem revisited. *Bioinformatics*, 24(13):i132–i138, 2008.

2 Geraldine Butler, Matthew D Rasmussen, Michael F Lin, Manuel AS Santos, Sharadha Sakthikumar, Carol A Munro, Esther Rheinbay, Manfred Grabherr, Anja Forche, Jennifer L Reedy, et al. Evolution of pathogenicity and sexual reproduction in eight candida genomes. *Nature*, 459(7247):657, 2009.

3 Cedric Chauve and Nadia El-Mabrouk. New perspectives on gene family evolution: losses in reconciliation and a link with supertrees. In *Annual International Conference on Research in Computational Molecular Biology*, pages 46–58. Springer, 2009.

4 Cedric Chauve, Akbar Rafiey, Adrian A. Davin, Celine Scornavacca, Philippe Veber, Bastien Boussau, Gergely Szollosi, Vincent Daubin, and Eric Tannier. Maxtic: Fast ranking

of a phylogenetic tree by maximum time consistency with lateral gene transfers. *bioRxiv*, 2017. URL: https://www.biorxiv.org/content/early/2017/11/07/127548.

5 Adrián A Davín, Eric Tannier, Tom A Williams, Bastien Boussau, Vincent Daubin, and Gergely J Szöllősi. Gene transfers can date the tree of life. *Nature ecology & evolution*, page 1, 2018.

6 Wandrille Duchemin. *Phylogeny of dependencies and dependencies of phylogenies in genes and genomes*. PhD thesis, Université de Lyon, 2017.

7 Wandrille Duchemin, Yoann Anselmetti, Murray Patterson, Yann Ponty, Sèverine Bérard, Cedric Chauve, Celine Scornavacca, Vincent Daubin, and Eric Tannier. Decostar: Reconstructing the ancestral organization of genes or genomes using reconciled phylogenies. *Genome biology and evolution*, 9(5):1312–1319, 2017.

8 Michael Fellows, Michael Hallett, and Ulrike Stege. On the multiple gene duplication problem. In *International Symposium on Algorithms and Computation*, pages 348–357. Springer, 1998.

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

10 Roderic Guigo, Ilya Muchnik, and Temple F Smith. Reconstruction of ancient molecular phylogeny. *Molecular phylogenetics and evolution*, 6(2):189–213, 1996.

11 Edwin Jacox, Cedric Chauve, Gergely J. Szöllősi, Yann Ponty, and Celine Scornavacca. ecceteta: comprehensive gene tree-species tree reconciliation using parsimony. *Bioinformatics*, 32(13):2056–2058, 2016.

12 Edwin Jacox, Mathias Weller, Eric Tannier, and Céline Scornavacca. Resolution and reconciliation of non-binary gene trees with transfers, duplications and losses. *Bioinformatics*, 33(7):980–987, 2017.

13 Manolis Kellis, Bruce W Birren, and Eric S Lander. Proof and evolutionary analysis of ancient genome duplication in the yeast saccharomyces cerevisiae. *Nature*, 428(6983):617, 2004.

14 Manuel Lafond, Mona Meghdari Miardan, and David Sankoff. Accurate prediction of orthologs in the presence of divergence after duplication. *Bioinformatics*, In press, 2018.

15 Cheng-Wei Luo, Ming-Chiang Chen, Yi-Ching Chen, Roger WL Yang, Hsiao-Fei Liu, and Kun-Mao Chao. Linear-time algorithms for the multiple gene duplication problems. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 8(1):260–265, 2011.

16 Bin Ma, Ming Li, and Louxin Zhang. From gene trees to species trees. *SIAM Journal on Computing*, 30(3):729–752, 2000.

17 Wayne P Maddison. Gene trees in species trees. *Systematic biology*, 46(3):523–536, 1997.

18 Roderic DM Page. Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Systematic Biology*, 43(1):58–77, 1994.

19 Roderic DM Page and JA Cotton. Vertebrate phylogenomics: reconciled trees and gene duplications. In *Pacific Symposium on Biocomputing*, volume 7, pages 536–547, 2002.

20 Jaroslaw Paszek and Pawel Gorecki. Efficient algorithms for genomic duplication models. *IEEE/ACM transactions on computational biology and bioinformatics*, 2017.

21 Ikram Ullah, Joel Sjöstrand, Peter Andersson, Bengt Sennblad, and Jens Lagergren. Integrating sequence evolution into probabilistic orthology analysis. *Systematic Biology*, 64(6):969–982, 2015.

Protein Classification with Improved Topological Data Analysis

Tamal K. Dey
Department of Computer Science and Engineering, The Ohio State University, Columbus, USA
http://web.cse.ohio-state.edu/~dey.8/
dey.8@osu.edu

Sayan Mandal
Department of Computer Science and Engineering, The Ohio State University, Columbus, USA
http://web.cse.ohio-state.edu/~mandal.25/
mandal.25@osu.edu

Abstract

Automated annotation and analysis of protein molecules have long been a topic of interest due to immediate applications in medicine and drug design. In this work, we propose a topology based, fast, scalable, and parameter-free technique to generate protein signatures.

We build an initial simplicial complex using information about the protein's constituent atoms, including its radius and existing chemical bonds, to model the hierarchical structure of the molecule. Simplicial collapse is used to construct a filtration which we use to compute persistent homology. This information constitutes our signature for the protein. In addition, we demonstrate that this technique scales well to large proteins. Our method shows sizable time and memory improvements compared to other topology based approaches. We use the signature to train a protein domain classifier. Finally, we compare this classifier against models built from state-of-the-art structure-based protein signatures on standard datasets to achieve a substantial improvement in accuracy.

2012 ACM Subject Classification Applied computing → Life and medical sciences

Keywords and phrases topological data analysis, persistent homology, simplicial collapse, supervised learning, topology based protein feature vector, protein classification

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.6

Supplement Material http://web.cse.ohio-state.edu/~dey.8/proteinTDA

Acknowledgements This work has been supported by NSF grants CCF-1318595, CCF-1526513, and CCF-1733798.

1 Introduction

Proteins are by far the most anatomically intricate and functionally sophisticated molecules known. The benchmarking and classification of unannotated proteins have been done by researchers for quite a long time. This effort has direct influence in understanding behavior of unknown proteins or in more advanced tasks as genome sequencing. Since the sheer volume of protein structures is huge, up till the last decade, it had been a cumbersome task for scientists to manually evaluate and classify them. For the last decade, several works aiming at automating the classification of proteins have been developed. The majority of annotation and classification techniques are based on sequence comparisons (for example in BLAST [19], HHblits [2] and [18]) that try to align protein sequences to find homologs or a common ancestor. However, since those methods focus on finding sequence similarity, they are more

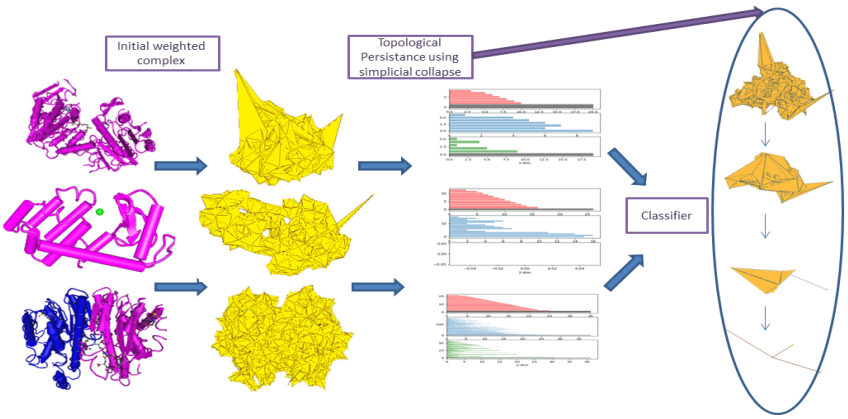


Figure 1 Workflow of our technique.

efficient in finding close homologs. Some domains such as remote homologs are known to have less than 25% sequential similarity and yet have common ancestors and are functionally similar. So, we miss out important information on structural variability while classifying proteins solely based on sequences. Even though, sometimes, homology is established by comparing structural alignment [14], accurate and fast structural classification techniques for the rapidly expanding Protein Data Bank remains a challenge.

Several works on the classification of protein structures exist in the literature. The main intuition behind these works draws upon a heuristic that generates a signature for each protein strand so that structurally close proteins have similar signatures. Essentially, the signature alignment quantifies the similarity between two protein structures. The problem, however, remains with the speed of computing these signatures and the degree of their representative power. We want a fingerprint for the protein that can be computed fast and can tell whether two proteins are dissimilar or even marginally similar.

Some works use vector of frequencies to describe structural features while others take various physical properties into account such as energy, surface area, volume, flexibility/rigidity or use other features from geometric modeling. The "Bag-Of-Word" (BOW) representation to describe an object has been used in computer vision, natural language processing and various other fields. The work by Budowski-Tal [3] have described protein structure using a fragment library in a similar context. Since we use this work for comparison, we shall discuss its details later.

Topological data analysis [10], a newly developed data analysis technique has been shown to give some encouraging results in protein structure analysis. Topological signatures, particularly based on Persistent Homology, enjoy some nice theoretical properties including their robustness and scale invariance. These features are global and more resilient to local perturbations. Moreover, they are invariant to scaling and any isometric transformation of the input. The authors in [23] extract topological fingerprints based on the alignment of atoms and molecules in three dimensional space. Their work shows the impact of persistent homology in the modeling of protein flexibility which is ultimately used in protein B-factor analysis. This work also characterizes the evolution of topology during protein folding and thereby predicts its stability. For this task, the authors have introduced a coarse grain (CG) representation of proteins by considering an amino acid molecule as an atom C_α. This helps them describe the higher level protein structures using the topological fingerprint perfectly. However, since the CG homology may be inconsistent due to ambiguity in choosing the CG particle, we present a similar study on secondary structures using our signature and show that our method does not require such a representation as it is inherently scaling independent.

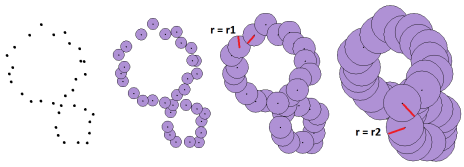


Figure 2 Persistence of a point cloud in $\mathbb{R}^2$ and its corresponding barcode

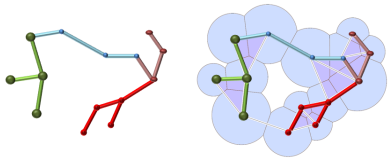


Figure 3 Weighted Alpha complex for protein structure

The authors in [4] have used persistent homology to generate feature vector in the context of machine learning algorithms applied to protein structure explorations. We explore further to improve upon the technique to eliminate its deficiencies. First, the approach in [4] does not differentiate between atoms belonging to different elements. Also, it does not account for the existing chemical bonds between the atoms while building the signature. Most importantly it uses Vietoris Rips(VR) complex to generate the topological features for protein complex which suffers from the well-known problem of scalability. As we will describe later, the VR complex developed in the early 20th century grows rapidly in size even for moderate size protein structures. Current state-of-the-art techniques, which have addressed the problem to some extent, are still very cumbersome and slow especially for structures having about 30,000 atoms on an average. Among the several methods that generate persistence signature from a point cloud, the PHAT toolbox [1] based on several efficient matrix reduction strategies and GUDHI [22] library based on some compression techniques have been popular because of their space and time efficiencies. A recent software called SimBa [8] published last year, has been shown to work faster for large datasets. Yet, for our application, SimBa falls short as we shall see later.

The algorithm that we present here is a fast technique to generate a topological signature for protein structures. We build our signature based on the coordinates of the atoms in $\mathbb{R}^3$ using their radius as weights. Since we also consider existing chemical bonds between the atoms while building the signature, we believe that the hierarchical convoluted structure of protein is captured in our features. Finally, we have developed a new technique to generate persistence that is much quicker and uses less space than even the current state-of-the-art such as SimBa. It helps us generate the signature even for reasonably large protein structures. In sum, in this paper, we focus on three problems: (1) effectively map a protein structure into a suitable complex; (2) develop a technique to generate fast persistent signature from this complex; (3) use this signature to train a machine learning model for classification and compare against other techniques. Our entire method is summarized in figure 1. We also illustrate this method using a supplementary video available at `https://youtu.be/yfcf9UWgdTo`.

2 Methods

We use the theory of topological persistence to generate features for protein structures. These topological features serve as a distinct signature for each protein strand. In this section, we give some background on persistent homology followed by how we construct our signature.

2.1 Persistence signature of point cloud data

We start with a point cloud data in any n-dimensional Euclidean space. These will essentially be the centers of protein atoms in the three dimensional space. However, to illustrate the theory of persistent homology, we consider a toy example of taking a set of points in two

dimensions sampled uniformly from a two-hole structure (Fig. 2). We start growing balls around each point, increasing their radius r continually and tracking the behavior of the union of these growing balls. If we start increasing r from zero, we notice that at $r = r_1$ (third from left in Fig 2) both holes are prominent in the union of ball structure. Further increasing r to r_2, leads to filling of the smaller hole (fourth figure from left). This continues till the value of r is large enough for the union of balls to fill the entire structure. During the change in the structure of the union of balls due to increase in radius, the larger of the two holes *'persists'* for a larger range of r compared to the smaller one. Hence features that are more prominent are expected to persist for longer periods of increasing r. This is the basic intuition for topological persistence. The holes in this example are captured by calculating a set of *birth-death* pairs of homology cycle classes that indicate at which value of r the class is born and where it dies. The persistence is visualized in $\mathbb{R}^2$ using horizontal line segments that connect two points whose x-coordinates coincide with the birth and death values of the homology classes. These collection of line segments, as shown in Figure 2, are called barcodes [5]. The length of each line segment corresponds to the persistence of a cycle in the structure. Hence, the short blue line segments correspond to the tiny holes that are formed intermittently as the radius increases. The two long red line segments correspond to the two holes in the structure, the largest being the bigger hole. For computational purposes, the growing sequence of the union of balls is converted to a growing sequence of triangulations, simplicial complexes in general, called a *filtration*. In some cases, some cycles called the *'essential cycles'* persists till the end of the filtration.

The rank of the persistent homology group called the persistent Betti numbers capture the number of persistent features. For n-dimensional homology group, we denote this number as β_n. This means β_0 counts the number of connected components that arise in the filtration. Similarly, β_1 counts the number of *circular* holes being born as we proceed through the filtration. It is due to this fact that all the folds in the tertiary structure, as well as the helix and strands in the secondary structure of proteins, are recorded in our signature.

With the above technique, difficulties are faced as r increases. An average protein in a database such as CATH [20] has 20,0000~30,000 atoms, thus creating a point cloud of the same size in $\mathbb{R}^3$. Furthermore, the initial complex including 3-simplices (or tetrahedra) becomes quite large. On an average, this complex size grows to (50~100)x10^4 simplices of dimension upto 4 and becomes quite difficult to process. Building a filtration using this growing sequence of balls is thus not scalable. We attack the problem with two strategies: (1) we only consider simplices on the boundary of the entire simplicial complex in our algorithm and (2) compute a new filtration technique that is based on collapsing simplices rather than growing their numbers by addition.

Topological persistence

Traditionally, given a point cloud, its persistence signature is calculated by building a filtration over a simplicial complex called *Vietoris-Rips*(VR). This technique is also used in [4] which takes the 3D position of the centers of the atoms as points in the point cloud. Given a parameter α, we can define VR complex over a point cloud P as: $\mathcal{VR}^{\alpha}(\mathbf{P}) = \{\sigma \mid \mathbf{d}(\mathbf{p}, \mathbf{q}) < \alpha \ \forall \ \mathbf{p}, \mathbf{q} \in \sigma\}$.

As the value of α increases, more edges and higher order simplices are introduced, and a *filtration* is obtained. Finally, the persistence of this *filtration* is computed. For a better representation of protein molecules, we take into account the radius of different atoms as weight of the points. So, we replace each point $p \in P$ with a tuple $\hat{p} = (p, r_p)$ where r_p is the radius of the atom represented by p. For the resulting weighted point cloud $\hat{P} = \{(p, r_p)\}$, we consider the weighted VR complex: $\mathcal{VR}^{\alpha}(\hat{\mathbf{P}}) = \{\sigma \mid \mathbf{d}(\mathbf{p}, \mathbf{q}) < \alpha(\mathbf{r_p} + \mathbf{r_q}) \ \forall \ \mathbf{p}, \mathbf{q} \in \sigma\}$.

The VR complex is easy to implement, but its size can become a hindrance for an even a moderate size protein molecule. Thus, instead of a VR complex, we use the (weighted) alpha complex that is sparser and has been used to model molecules in earlier works [11].

Alpha complex AC(α): For a given value of α, a simplex $\sigma \in AC(\alpha)$ if:

- The circumball of σ is empty and has radius $< \alpha$, *or*
- σ is a face of some other higher dimensional simplex in AC(α).

Weighted Alpha Complex $WAC_{\hat{P}}(\alpha)$: Let $B_k(\hat{p})$ be a k-dimensional closed ball with center p, and weight r_p. It is orthogonal or sub-orthogonal to a weighted point $(p',r_{p'})$ iff $||\mathbf{p}-\mathbf{p'}||^2 = \mathbf{r_p^2} + \mathbf{r_{p'}^2}$ **or** $||\mathbf{p}-\mathbf{p'}||^2 < \mathbf{r_p^2} + \mathbf{r_{p'}^2}$ respectively.

An *orthoball* of a k-simplex $\sigma = \{\hat{p_0}, \ldots, \hat{p_k}\}$ is a k-dimensional ball that is orthogonal to every vertex p_i. A simplex is in the weighted alpha complex $WAC_{\hat{P}}(\alpha)$ iff its orthoball has radius less than α and is suborthogonal to all other weighted points in $\hat{P}$.

2.2 Collapse-induced persistent homology from point clouds

The following procedure computes a topological signature for a weighted point cloud $\hat{P} = \{p, r_p\}$ using subsamples and subsequent collapses:

1. Compute a weighted alpha complex $\mathbb{C}^0$ on the point set $\hat{P} = \{p, r_p\}$ using the algorithm described in [22]. Let V^0 be the vertex set of $\mathbb{C}^0$.
2. Compute a sequence of subsamples $V^0 \supset V^1 \supset \ldots \supset V^k$ of the initial vertex set V^0 based on the Morton Ordering as discussed later. (For every V^i, we remove every n^{th} point in the Morton Ordering from V^i to form V^{i+1}. We choose 'n' based on the number of initial points).
3. This sequence of subsets of V^i allows us to define a simplicial map between any two adjacent subsets V^i and V^{i+1}: $f^i(p) = \begin{cases} p & \text{if } p \in V^{i+1} \\ \underset{v \in V^{i+1}}{\operatorname{argmin}}\, d(p,v) & \text{otherwise} \end{cases}$
4. This vertex map $f^i : V^i \to V^{i+1}$ in turn generates a sequence of collapsed complexes: $\mathbb{C}^0, \mathbb{C}^1, \ldots, \mathbb{C}^n$. Each vertex map induces a simplicial map $f^i : \mathbb{C}^{i-1} \to \mathbb{C}^i$ that associates simplices in $\mathbb{C}^{i-1}$ to simplices in $\mathbb{C}^i$(see Figure 4)
5. Compute the persistence for the simplicial maps in the sequence $\mathbb{C}^0 \xrightarrow{f_1} \mathbb{C}^1 \xrightarrow{f_2} \ldots \xrightarrow{f_k} \mathbb{C}^k$ to generate the topological signature of the point set $\hat{P}$.

In step 1 of the procedure, weighted points alone lead to disconnected weighted atoms in $\mathbb{C}^0$ rather than capturing the actual protein structure. To sidestep this difficulty, we increase the weights of these points based on the existence of covalent or ionic bonds in the structure. That is, if there exists a chemical bond between two atoms (which we get from the input .pdb file), we scale-up the weight of each point so that they are connected in the weighted alpha complex $WAC_{\hat{P}}(\alpha)$ (see Fig. 3). We determine a global multiplying factor $\rho \geq 1$ for this purpose. As mentioned earlier, we take the boundary of this weighted complex which forms our initial simplicial complex $\mathbb{C}^0$.

In step 2, in order to generate the sequence of subsamples, we pick vertices uniformly from the simplicial complex to be collapsed to their respective nearest neighbors. To choose a subsample that respects local density, we use a space curve generation technique called Morton Ordering [15]. The Morton curve generates a total ordering on the point set V^0.

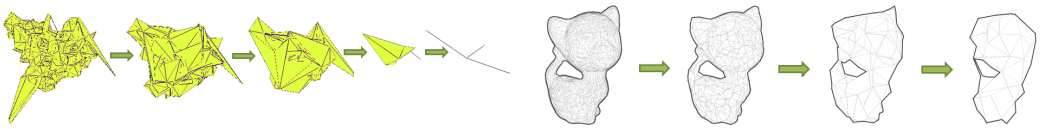


Figure 4 (a) Collapse of weighted alpha complex generated from protein structure via simplicial map. (b) Same algorithm applied to a kitten model in $\mathbb{R}^3$.

This ordering is explicitly defined by the Morton Ordering map $M : \mathbb{Z}^N \mapsto \mathbb{Z}$ given by:

$$\mathbf{M(p)} = \bigvee_{\mathbf{b=0}}^{\mathbf{B}} \bigvee_{\mathbf{i=0}}^{\mathbf{N}} \mathbf{x_2^{i,b}} \ll \mathbf{N(b+1)} - \mathbf{(i+1)},$$

where $x_2^{i,b}$: b^{th} bit value of the binary representation of the i^{th} component of x.
This map merely interleave bits of the different components of p. Application of M to V^0 yields a total ordering on our initial point set. To generate a new subset $V^1 \subset V^0$, we simply choose a value n such that $1 < n \leq \|V^0\|$. Then, V^{i+1} is taken as:

$$\mathbf{V^{i+1}} = \{\mathbf{x_j} \mid \mathbf{x_j} \in \mathbf{V^i}, \mathbf{j} \not\equiv \mathbf{0} \bmod \mathbf{n}\},$$

where x_j is the j^{th} vertex in the Morton Ordering of V^i. We choose $n = 12$ as it has procured good results for the datasets we experimented on (having 20,000~30,000 atoms on an average). Following this approach, the process can be repeated to create a sequence of subsets $V^0 \supset V^1 \supset ... \supset V^n, \|V^n\| \leq k$ as done in step 2 of our procedure above.

Finally, as described in step 3, instead of constructing the filtration by increasing the value of α, we perform a series of successive collapses starting with the initial simplicial complex. This leads to a sequence of complexes that decreases in size instead of growing as we proceed forward. Effectively, it generates a sequence called *tower* of simplicial complexes where successive complexes are connected by *simplicial maps*. These maps which are the counterpart of continuous maps for the combinatorial setting extend maps on vertices (vertex maps) to simplices (see [16] for details). In our case, collapses of vertices generate these simplicial maps between a simplicial complex in the tower to the one it is collapsed to. Persistence for towers under simplicial maps can be computed by the algorithm introduced in [7]. We use the package called Simpers that the authors have reported for the same.

To summarize, the algorithm generates an initial weighted alpha complex. It then proceeds by recursively choosing vertices based on Morton Ordering to be collapsed to their nearest neighbors resulting in vertex maps. These vertex maps are then extended to higher order simplices (such as triangles and tetrahedra) using the simplicial map. Finally given the simplicial map, we generate the persistence and get the barcodes for the zero and one dimensional homology groups.

2.3 Feature vector generation

We discuss how we generate a feature vector given a protein structure. We take protein data bank (*.pdb) files as input to extract protein structures. It contains the coordinates of every atom, their name, chemical bond with neighboring atoms and other meta-data such as helix, sheet and peptide residue information. We introduce a weighted point for each atom in the protein where the point is the center of the atom and its weight is the specified radius. For instance, for a Nitrogen atom in the amino acid, we assign a weight equal to its covalent

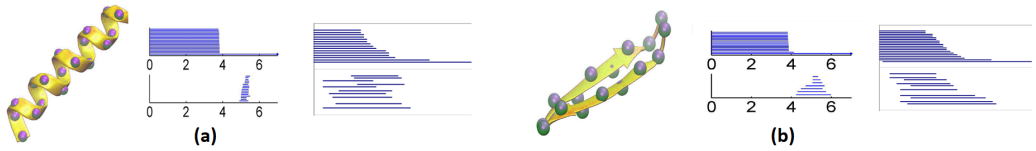


Figure 5 (a) Left: Alpha helix from PCB 1C26 , Middle: Barcode of [23], Right: Our Barcode, (b) Left: Beta sheet from PCB 2JOX, Middle: Barcode of [23], Right: Our Barcode. Each segment of the barcodes shows $\beta 0$(top) and $\beta 1$(bottom).

radius of 71(pm). On this weighted point cloud $\hat{p} = (p, r_p)$, if two atoms $\hat{p}$ and $\hat{q}$ are involved in a chemical bond, we increase their weights so that p and q get connected in the alpha complex. We compute the persistence by generating the initial alpha complex and undergoing a series of collapses as described in the previous section. For computational efficiency, we only consider the barcodes in zero and one dimensional homology groups. Note that some of the barcodes can have death time equal to infinity indicating an essential feature. For finite barcodes, shorter lengths $(death - birth)$ indicate noise. Elimination of these intermittent features serves some interesting purpose as we will see in section 3. To find relatively long barcodes, we sort them in descending order of their lengths. Let $\{l_1, l_2, ..., l_k\}$ be this sorted sequence. Consider the sequence $\{l'_1, l'_2, ..., l'_{k-1}\}$ where $l'_i = l_{i+1} - l_i$ and let l'_m be a maximal element for $1 \leq m \leq k-1$. All barcodes with the lengths $[l_1..l_m]$ form part of the feature vector. Essentially we remove all barcodes whose lengths are shorter than the largest gap between two consecutive barcodes when sorted according to their lengths. A similar technique used in [13] has shown improved results in image segmentation over other heuristics and parameterizations. Since the feature vector needs to be of a fixed length for feeding into a classifier, we compute the index m of l_m over all protein structures and take an average. The feature vector also includes the number of *essential* zero and one dimensional cycles. Therefore, we have a feature vector of length $2 \times m + 2$: $\{l^0_1, l^0_2, ...l^0_m, l^1_1, l^1_2, ...l^1_m, c_{\beta_0}, c_{\beta_1}\}$. Here l^0_i and l^1_i are the lengths of zero and one dimensional homology cycles respectively whereas c_{β_i} are the total number of essential cycles in i-dimensional homology.

3 Experiments and results

We perform several experiments to establish the utility of the generated topological signature. First, we show how our feature vector captures various connections in the single strands of secondary structures and compare them against the signatures obtained in [23]. Then we investigate if there is a correlation between the count of such secondary structures and our feature vector. Next, we describe the topological feature vector obtained from two macromolecular proteins structures. We also compare the size and time needed by our algorithm (software) over the other commonly used persistence software (as in [4]). Lastly, we show the effectiveness of our approach in classifying protein structures using machine learning models.

Topological description of alpha helix and beta sheet

It is known that barcodes can explain the structure of an alpha helix and a beta sheet [23]. The authors in [23] use a coarse-grain(CG) representation of the protein by replacing each amino acid molecule with a single atom. This representation removes the short barcodes corresponding to the edges and cycles of the chemical bonds inside the amino acid molecule.

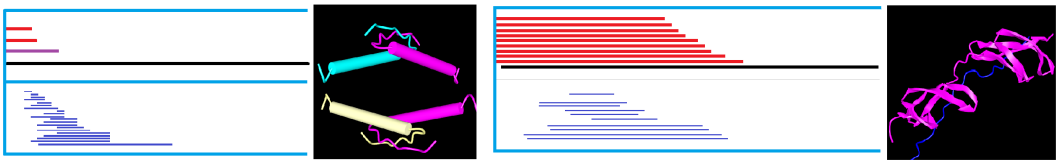


Figure 6 Barcode and Ribbon diagram of (Left): PDB: 1c26. (Right): PDB: 1o9a. Diagram courtesy NCBI [17].

We do not need this CG representation as our procedure can implicitly determine a threshold l_m and therefore delete all barcodes of length shorter than the largest gap between two consecutive barcodes (as described in section 2.3). So, we get a barcode that describes the essential features of the secondary structures without including noise or short lived cycles from the amino acids. For a fair comparison, we compute our barcodes on the same alpha helix residue as in [23] with 19 residues extracted from the protein strand having PDB ID 1C26 (see figure 5). Analogous to the barcode of [23] (as shown in the middle diagram of figure 5a), we have 19 bars in the zero-dimensional homology for the alpha helix representing the nineteen initial residues. These components die as edges are introduced in the weighted alpha complex which gets them connected. For one-dimensional homology, an initial ring with 4 residues is formed followed by additional rings resulting from the growing connections in each amino acid. These cycles eventually die by the collapse operations in our algorithm.

The same process is followed for beta sheets after we extract two parallel beta sheet strands from the protein structure with PDB ID 2JOX. The zero-dimensional homology cycles are killed when individual disconnected amino acid residues belonging to the same beta sheet strand are connected by edges, as represented in the top 17 barcodes (leftmost figure of 5b). However, other than these barcodes and the longest bar corresponding to the *essential cycle*, there is one bar in the zero-dimensional homology which is longer than the top 17 bars. This bar represents the component which is killed by joining the closest adjacent amino acid molecules from the two parallel beta strands. The one dimensional homology bars are formed as more adjacent amino acid molecules are connected and killed once the collapse operation starts. Note that the two barcodes shown in figure 5 comparing our work with [23] are not to scale. This is because, in contrast to [23], the barcodes in our figure are not plotted against Euclidean distance rather the step at which each insertion and collapse operation occurs.

A caveat

Our aim is to compute signatures that capture discriminating structural information useful for classifying proteins. Even though we can use our signature to describe secondary structures, we do not want our signature to be directly correlated to the *number of* alpha helix or beta sheet as it would mean they are redundant. We generate a 2×12 matrix where each cell contains the correlation value between beta-sheet(top row) and alpha-helix(bottom row) with each individual component in the feature vector: $\{l_1^0, l_2^0, ...l_m^0, l_1^1, l_2^1, ...l_m^1, c_{\beta 0}, c_{\beta 1}\}$. We use proteins in the PCB00020 record of the PCBC database to compute this matrix and depict it by a heatmap (Fig 7). Essentially, we first generate two vectors v_α and v_β of the number of alpha helices and beta sheets respectively in each protein over all entries in the database. Similarly, we produce a vector for each value in the feature vector: $\{v_{l_1^0}, ..., v_{c_{\beta 1}}\}$. Now we populate the matrix by calculating the correlation between each of these individual vectors with v_α and v_β. For example, row 1 and column 1 of the matrix contain the correlation value between the vectors $v_{l_1^0}$ and v_β. The heat map color ranges from blue for

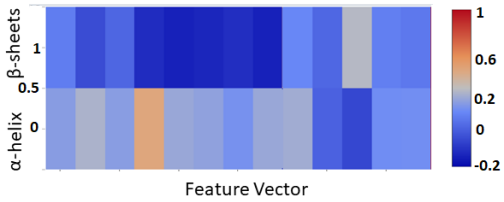


Figure 7 Heatmap correlating secondary structure against our feature vector. Each column in the heatmap is the feature vector.

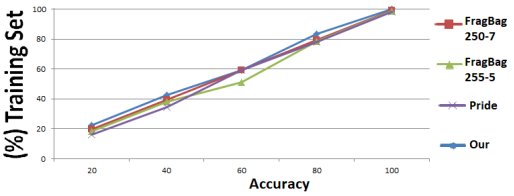


Figure 8 Plot showing accuracy against varying training data size. 100(%) indicates the entire training and test data.

zero correlation to dark-red for complete correlation. As we can see from the figure, almost all matrix entries have a blue tinge indicating low correlation. This shows that our feature vector is non-redundant over the frequency of secondary structures.

Topological Description of macromolecular structures

In the previous section, we use our signature to describe the secondary structures and compare it with the work in [23]. In this section, we further show how our signature works by describing two macromolecular protein structures that are built on multiple secondary structures. We start by describing the tetrametric protein: 1C26. The ribbon diagram and associated barcode after noise removal is given in figure 6 . It essentially contains four monomers, associated pairwise to form two dimers. These two dimers, in turn, join across a distinct parallel helix-helix interface to form the tetramer. When we build the filtration on this protein structure, two monomers on opposite sides are killed first by connecting to their adjacent monomers to form two distinct dimers. This is evident as there are two short bars in the zero dimensional barcode (Fig. 6 right: shown in red). We now have two dimers, one of which is killed when it joins with the other to form a third slightly longer non-essential barcode (shown in purple). The second dimer lives on as the tetramer and forms an essential barcode (shown in black). Next, if we look into the one dimensional homology (shown as blue lines), we notice that the most notable feature for the protein is the tetramer structure which contains a large loop when the two dimers are connected. This is evident in our 1D-barcode as there is a distinct long bar representing the large one dimensional cycle. Note that the birth time of this cycle in 1D corresponds with the death time of the non-essential dimer in 0D.

Next, we consider the protein structure 1O9A. The structure contains several antiparallel beta-strands and is an example of a tandem beta-zipper. As we can see from the ribbon diagram in Fig. 6, there are six beta sheets on one side and five on another, connected together to form a fibronectin. This is evident as there are ten non essential and one essential bar in the zero dimensional homology owing to the six beta sheets on one side and five on the other. Each component is killed as the beta sheets join with another as the filtration proceeds. Note that the last connected component after joining all beta sheets forms an essential bar. Moreover, since there is no distinct cycle in the structure, we do not get any distinct long bar in the one dimensional homology. The presence of multiple one dimensional bars of similar size are probably due to the antiparallel beta-strands on either side which form a ring once joined. Thus, we can see that using the same signature generation method, we can describe secondary structures (as in the previous section) as well as macromolecular proteins without any change in the parameter. It is therefore evident that our signature is intrinsic and scale independent.

Table 1 Time comparison of our algorithm against SimBa [8] and VR complex.

		Size			Time (in sec)		
Data	Dim	VR	SimBa	Our	VR	SimBa	Our
CATH	3	–	1422	443	–	1.75	0.35
Soneko	3	324802	10188	576	32	6.77	2.05
Surv-l	150	–	3.1×10^6	1.09×10^6	–	5.08×10^3	884
PrCl-I	25	–	10.2×10^6	0.22×10^6	–	585	141.3

Table 2 Accuracy comparison with FragBag and Cang.

	SVM			KNN		
	FB	Cang	Our	FB	Cang	Our
Class	91.08	89.07	92.36	86.01	86.40	86.39
Architecture	90.26	91.11	92.20	88.17	87.47	89.11
Topology	92.19	94.87	96.71	91.54	94.02	96.20
Homology	93.33	94.06	94.17	90.28	91.11	93.30

Time and space comparison with VR-complex and SimBa

The method in [4] uses persistent homology as feature vectors for machine learning. However, as mentioned earlier, the use of Veitoris-Rips (VR) complex leads to a size blow up that not only increases runtime but also in most cases, causes the operating system to abort the process due to space requirements. Results in [4] procure good results as the datasets are of moderate size, but the same could not be reported for larger and real life protein structures. In table 1, we show a size and time comparison of our approach with the original feature generation technique used in [4]. We also tabulate the size and time to generate the same feature vector in [4] using a state-of-the-art persistence algorithm called SimBa [8]. Table 2 contains a mix of protein databases and other higher dimensional datasets. As we see in the table, our algorithm is faster even when the features in [4] are generated with SimBa.

3.1 Supervised learning based classification models

Classification model. For the purpose of protein classification, we train two classifiers: an SVM model and a k-nn model on some protein databases. Once the model is trained, we test it to find accuracy, precision, and recall. The reason behind choosing Support Vector Machine and k-nn based supervised learning technique over other sophisticated and state-of-the-art classifiers is their basic nature. Results obtained from basic learning techniques prove the effectiveness of the feature vectors rather than that of the classifier. We can further improve the classification accuracy for proteins using some advanced supervised learning or Neural Network based classifiers using our proposed features.

Benchmark techniques. In order to test the effectiveness of our protein signature, we need to compare it against some of the state-of-the-art protein structure classification techniques. We generate feature vectors through these techniques and train and test the same classification models as before. The first technique, known as PRIDE [9], classifies proteins at the Homology level in the CATH database. It represents protein structures by distance distributions and compares two sets of distributions via contingency table analysis. The more recent work by Budowski-Tal et al. [3], which has achieved significant improvement in protein classification is used as our second benchmark technique. Their work, known as FragBag mimics the bag of word representation used in natural language processing and computer vision. They maintain protein fragments as a benchmark. Given a large protein strand, they generate a vector which is essentially a count of the number of times each fragment approximates a segment in this strand. This vector now acts as a signature for the protein structure and that is what forms the basis for their feature vector which we use to train and test our classifier. The protein fragment benchmark is available from the library [12]. We choose 250 protein fragments of length 7. The third work that we test against is the topological approach to generate a protein fingerprint [4]. However, as we saw earlier, it is not possible to generate

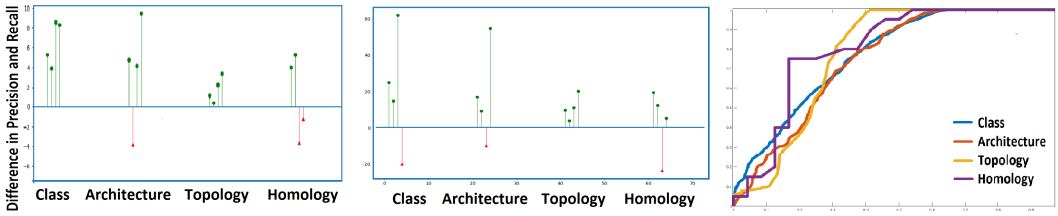


Figure 9 Left:a) Difference in precision and recall from FragBag. Middle: b) Difference in precision and recall from [4]. Right: c) ROC curve for SVM classification of our algorithm.

all the protein signatures using the original algorithm used by the authors. Therefore, we replace the Vietoris-Rips filtration by the state-of-the-art SimBa and generate feature vectors the same way as mentioned in their paper.

Database. The database that we use is called Protein Classification Benchmark Collection (PCBC) [21]. It has 20 classification tasks, each derived from either the SCOP or CATH databases, each containing about 12000 protien structures. The classification experiment involves the group of the domain as positive set, and the positive test set is the sub-domain group. The negative set includes the rest of the database outside the superfamily divided into a negative test or negative training set. The result for some of the classification tasks for the database is given in Table 3. As evident from the table, the accuracy obtained by using our signature has a considerable improvement over the state of the art techniques. The only classification task in which our algorithm under-performs is with the protein domain CATH95_Class_5fold_Filtered (fourth row of table 3). The class domain is randomly sub-divided into 5 subclasses in this task. Since the class is divided randomly into subclasses, we believe some proteins belonging to different sub-classes have generated a similar initial complex resulting in a similar filtration and ultimately a decrease in performance.

The PCBC dataset, even though suitable for learning algorithms, suffers from being skewed as the number of negative instances in any classification is much larger than the number of positive instances, leading to probable incorrect classifications. Therefore, we test on one of the most popular protein databases known as CATH [6]. The CATH database contains proteins divided into different domains (C: class; A: architecture; T: topology; H: homologous superfamily). For each domain, we get protein structures and their labels in accordance with the sub-domain they belong to. For any classification task, we randomly choose positive instances from one sub-domain and the same number of negative instances sampled equally from the other sub-domains. Each such task, on average has 400 protein structures containing approximately 30,000 atoms each. We then divide this into 80%-training and 20%-test set. The result of classification on the CATH database averaged over several such randomly chosen sub-domains as positive classes, are illustrated in table 2. We see yet again that for each case, there is an improvement of about 3-4% over the benchmark techniques.

3.1.1 Classification result

We have listed our main results in tables 2 and 3 showing the improvement in accuracy using our method over the state-of-the-art techniques of FragBag, PRIDE and the preceding work on topology by Cang et al. [4]. We provide further evidence of the efficiency of our algorithm by comparing the precision and recall in figures 9a and b. In these plots, we show

Table 3 Classification accuracy for different techniques on Protein dataset. SC: SCOP95, CA: CATH95, Sf: Superfamily, Fm: Family, F: Filtered, T: Topology, H: Homology, C: Class, 5f: 5fold, A: Architecture, Si: Similarity.

	SVM				k-NN			
	Pride	Fragbag	Cang	Our	Pride	Fragbag	Cang	Our
SC_Sf_Fm_F	90.09	93.01	93.39	95.24	89.58	87.31	89.83	91.66
CA_T_5f	94.23	92.97	94.87	99.53	90.96	91.16	94.57	97.87
CA_T_H_F	90.15	89.89	95.06	98.80	84.98	81.11	86.65	95.51
CA_C_5f_F	85.09	84.76	80.98	82.36	80.18	84.74	83.83	78.81
CA_H_Si_F	98.60	95.89	98.24	99.05	95.45	91.11	79.469	97.56
CA_A_T_F	87.56	91.58	74.58	90.95	67.47	89.00	68.90	87.00

the difference between the precision and recall obtained using our algorithm against that of FragBag(9a) and Cang(9b). A green bar indicates that our algorithm performed better and the difference is positive while a red bar suggests the opposite. This experiment is done on the CATH database and the figure shows the precision and recall for each domain: class(C), architecture(A), topology(T) and homology(H). Notice that, since the classification is binary, we get two precision and two recall for every class in each domain. Thus, there are four bars for each of C,A,T,H. Yet again, other than a few marginal cases, our algorithm largely performs better. Finally, we calculate the ROC curve using SVM on a subset of the CATH dataset, the result of which is shown in figure 9c. The ROC curve is a plot of the true positive rate against false positive rate obtained by changing the input size and parameter. This means that the further the lines are away from the diagonal, the better is the classifier.

For the positive test cases, we investigate further the trend of the output. We try to see the correlation of accuracy with the change in training set size. We therefore change the training and test set sizes by taking a fraction of the entire dataset and trace the accuracy in each case. This is done over all the test cases shown in Table 3 and the average is shown in Fig 8. We have plotted the output of our algorithm in blue with two instances of FragBag with (fragment, library) sizes (5,225) and (7,250) in red and green respectively. In addition, we have plotted the output of PRIDE as well. Ideally, the accuracy should decrease uniformly with a decrease in training set size and we should get a straight line across the diagonal. In this case, all the trendlines are almost close to the diagonal and hence we can say that they are correlated. Moreover, we observe that even as the training data size decreases, the accuracy of our algorithm remains better or comparable to the other algorithms. This indicates that topological features work better with a lower number of samples as well.

4 Conclusion

We present a practical topological technique to generate signatures for protein molecules that can be used as feature vectors for its classification. Since we investigated the descriptive power of our signature, we believe it can be used for other purposes such as protein energy computation, or finding protein B-factor. We believe that this signature can be extended to other biomolecular data such as DNA or enzymes.

References

1 Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat - persistent homology algorithms toolbox. *J. Symb. Comput.*, 78(C):76–90, 2017.

2 Juliana Bernardes, Gerson Zaverucha, Catherine Vaquero, Alessandra Carbone, and Levitt Michael. Improvement in protein domain identification is reached by breaking consensus,

with the agreement of many profiles and domain co-occurrence. *PLoS Computational Biology*, 12, 07 2016.

3 Inbal Budowski-Tal, Yuval Nov, and Rachel Kolodny. Fragbag, an accurate representation of protein structure, retrieves structural neighbors from the entire pdb quickly and accurately. *PNAS*, 107(8):3481–3486, February 2010.

4 Zixuan Cang, Lin Mu, Kedi Wu, Kristopher Opron, Kelin Xia, and Guo-Wei Wei. A topological approach for protein classification. In *Computational and Mathematical Biophysics*. MBMB, Nov 2015.

5 Gunnar Carlsson, Afra Zomorodian, Anne Collins, and Leonidas Guibas. Persistence barcodes for shapes. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 124–135. ACM, 2004.

6 Natalie Dawson, Tony E Lewis, Sayoni Das, Jonathan Lees, David Lee, Paul Ashford, Christine Orengo, and Ian Sillitoe. Cath: An expanded resource to predict protein function through structure and sequence. *Nucleic Acids Research*, 45, 11 2016.

7 Tamal K. Dey, Fengtao Fan, and Yusu Wang. Computing topological persistence for simplicial maps. *Symposium on Computational Geometry*, pages 345–354, june 2014.

8 Tamal K. Dey, Dayu Shi, and Yusu Wang. Simba: An efficient tool for approximating rips-filtration persistence via simplicial batch-collapse. In *ESA*, volume 57 of *LIPIcs*, 2016.

9 Zoltán Gáspári, Kristian Vlahovicek, and Sándor Pongor. Efficient recognition of folds in protein 3d structures by the improved pride algorithm. *Bioinformatics*, 21(15), 2005.

10 Edelsbrunner Herbert and John Harer. *Computational topology: an introduction*. American Mathematical Society, 2010.

11 Liang J, Edelsbrunner H, Fu P, Sudhakar PV, and Subramaniam S. Analytical shape computation of macromolecules: Ii. molecular area and volume through alpha shape. In *Proteins*, volume 33, pages 18–29, 1998.

12 Rachel Kolodny, Patrice Koehl, Leonidas Guibas, and Michael Levitt. Small libraries of protein fragments model native protein structures accurately. *JMB*, 323, 2002.

13 Vitaliy Kurlin. A fast persistence-based segmentation of noisy 2D clouds with provable guarantees. *Pattern Recognition Letters*, 83:3–12, 2015.

14 Holm Liisa and Rosenström Päivi. Dali server: conservation mapping in 3d. *Nucleic Acids Research*, 38:W545–W549, 2010. doi:10.1137/070711669.

15 G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. *International Business Machines Co.*, 1966.

16 J. R. Munkres. *Elements of Algebraic Topology*, chapter 1. CRC Press, 1 edition, 1984.

17 USA National Institutes of Health, 1988. URL: https://www.ncbi.nlm.nih.gov/.

18 M Remmert, A Biegert, and Söding J. Hauser A. Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment. *Nature Methods*, 9, Dec 2011.

19 Altschul S.F., Gish W., Miller W., Myers E.W., and Lipman D.J.n. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

20 Ian Sillitoe, Tony E Lewis, and et al. Cath: Comprehensive structural and functional annotations for genome sequences. *Nucleic Acids Research*, 43, 01 2015.

21 Paolo Sonego, Mircea Pacurar, Somdutta Dhir, Attila Kertesz-Farkas, András Kocsor, Zoltán Gáspári, Jack A M Leunissen, and Sándor Pongor. A protein classification benchmark collection for machine learning. *Nucleic acids research*, 35:D232–6, 02 2007.

22 The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. URL: http://gudhi.gforge.inria.fr/doc/latest/.

23 Kelin Xia and Guo-Wei Wei. Persistent homology analysis of protein structure, flexibility and folding. *IJNMBE*, 30(8):814–844, 2014. URL: doi:10.1002/cnm.2655.

A Dynamic Algorithm for Network Propagation

Barak Sternberg
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
barakolo@gmail.com
https://orcid.org/0000-0002-1803-6437

Roded Sharan[1]
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
roded@post.tau.ac.il

Abstract

Network propagation is a powerful transformation that amplifies signal-to-noise ratio in biological and other data. To date, most of its applications in the biological domain employed standard techniques for its computation that require $O(m)$ time for a network with n vertices and m edges. When applied in a dynamic setting where the network is constantly modified, the cost of these computations becomes prohibitive. Here we study, for the first time in the biological context, the complexity of dynamic algorithms for network propagation. We develop a vertex decremental algorithm that is motivated by various biological applications and can maintain propagation scores over general weights at an amortized cost of $O(m/n^{1/4})$ per update. In application to real networks, the dynamic algorithm achieves significant, 50- to 100-fold, speedups over conventional static methods for network propagation, demonstrating its great potential in practice.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Network propagation, Dynamic graph algorithm, protein-protein interaction network

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.7

Supplement Material `https://github.com/barakolo/dygraph_bio`

Funding R. S. was supported a grant from the Ministry of Science, Technology and Space of the State of Israel and the Helmholtz Centers, Germany.

1 Introduction

Network propagation has become a central technique in biology, as in other domains, to rank the relevance of genes to a process under investigation [7]. However, its complexity is becoming a bottleneck in dynamic settings where the network is subjected to multiple changes. In the biological domain, dynamic computations are essential not only because the network is updated with time but also because certain applications involve the systematic evaluation of propagation results under many network modifications. For example, in [12] the propagation over a network with n nodes is compared to n other propagations that are performed on modifications of the network where each time a different vertex is removed (simulating a knockout). Another application of the dynamic setting is when working with tissue-specific networks. As an example, in [14] multiple tissue-specific networks are formed from a given protein-protein interaction network by removing vertices with low expression, and propagation computations are applied to each.

[1] Corresponding author

While the computation of network propagation requires matrix inversion, a common and more efficient alternative utilizes the power iteration method [6]. This method approximates the propagation scores to within some additive constant at a cost of $O(m)$. An alternative local approach for obtaining approximate propagation was also suggested [3], yielding $O(m)$ time for a single propagation at worst case.

Focusing on a dynamic setting in which vertices are removed one by one [12], the total complexity of maintaining the propagation vectors after each removal becomes $O(mn)$ for n vertices when computing each propagation afresh. There is relatively scarce prior work regarding the computation of network propagation in a dynamic fashion which can be applied to the above setting. Specifically, Zhang et al. [5] and Ihsaka et al. [10] provide a fully dynamic propagation algorithm whose expected time per edge update is $O(1)$. However, both algorithms are limited to unweighted graphs and expected time analysis and might yield $O(mn)$ time under the settings considered here (of n vertex removals). This is true also for Yoon et al. [8], who provide a fully dynamic algorithm for network propagation but may lead to $O(mn)$ time under the settings considered here.

To tackle the dynamic computation challenge, we propose a novel algorithm that can handle general weights and several normalizations, including a symmetric normalization, a variant of which has been shown to be powerful in the biological domain [11, 12]. Our algorithm can handle n vertex removals in $O(mn^{3/4})$ total time. This yields a speedup of $\Omega(n^{1/4})$ over previous work. For real biological networks, this leads to a 50-fold to $100-fold$ speedup in the computations.

2 Preliminaries

We focus on undirected and weighted networks that represent protein-protein interactions. For a network G with n vertices and m edges, we denote by w the symmetric weighted adjacency matrix of the network. For a vertex u, we denote its set of neighbors by $N(u)$ and their number, i.e., the *degree* of u, by $d(u)$. The *weighted degree* $w(u)$ of u is the sum of weights of its adjacent edges. Two common normalizations of w to form a normalized matrix W are as follows: (i) normalizing each column of w to sum to 1, henceforth *weighted degree normalization*, and (ii) dividing each entry of w by the squared product of the weighted degrees of the corresponding nodes, i.e., $W_{ij} = w_{ij}/(\sqrt{w(i)}\sqrt{w(j)})$, henceforth *symmetric normalization* as used in [11].

Given a network G, a prior vector p of node relevance values (in [0,1]; typically 0 or 1), and a parameter $0 < \alpha < 1$, the *network propagation* transformation computes a score $s(v)$ for every node v that is a linear combination of its prior value and the average score of its network neighbors, reflecting its network proximity to the a-priori relevant nodes. Formally,

$$s(v) = \alpha p(v) + (1-\alpha) \sum_{u \in N(v)} s(u) W_{uv}$$

where $0 \leq \alpha \leq 1$ controls the tradeoff between prior information and network smoothing [7]. s, the propagation vector, can be computed analytically via matrix inversion or approximated using the "power iteration" algorithm which works as follows ([1, 9]):

1. Define $v_0 = p, i = 0$.
2. Compute $v_{i+1} = \alpha p + (1-\alpha) W v_i$.
3. While $i < \frac{\log(\epsilon)}{\log(1-\alpha)}$ increment i and goto (2).

The "power iteration" process is known to converge to the propagation vector ($\lim_{i\to\infty} v_i = s$), whenever the eigenvalues of W are at most 1 in absolute value, a condition satisfied

Algorithm 1 ForwardPush for weighted degree normalization.

```
1: procedure PUSH(v, R, P, α, W)
2:     P(v)+ = αR(v)
3:     for n ∈ N(v) do
4:         R(n)+ = (1 − α)R(v)W_vn
5:     R(v) = 0
6: procedure ForwardPush(W, α, ϵ, R, P)
7:     while ∃u|R(u)| > ϵd(u) do
8:         push(u, R, P, α, W)
9:     return P
```

by both normalizations presented above [7]. Here $\epsilon > 0$ is the required approximation bound on the sum of differences in absolute value (L1 error) between the computed and true propagation score. ϵ is typically chosen to be a constant smaller than 0.01. The following lemma characterizes the resulting propagation vector.

▶ **Lemma 2.1** ([2, 7]). *The propagation vector converges to* $\alpha(I - (1-\alpha)W)^{-1}p$.

Our algorithmic approach is motivated by the following lemma:

▶ **Lemma 2.2** ([6]). *Consider a random walk from prior distribution* p *using the weighted-degree normalized adjacency matrix* W*, where at each node* u *the walk stops with probability* α*. Then the total probability of the walk to stop at* u *is* $s(u)$*.*

3 The Forward-Push algorithm

In the following we describe our dynamic algorithms for the case of symmetric and weighted degree normalization. Our algorithm builds on the Forward-Push algorithm for the static case [3] which can be viewed as "simulating" random walks, “pushing” walks from one node to another and taking into account walks that stopped at any node, adding their probability to the node’s score. The algorithm is run until the residual walks have neglible effects (ϵ). For clarity, we fix the prior vector to p. The algorithm maintains two estimates per node u: the current estimate $P(u)$ of the probability to stop at u and the remaining probability $R(u)$ of walks that have reached u without stopping. The algorithm is given below and is called by initializing $P = 0^n$ (the zero vector) and $R = p$. For any nodes s, t we denote by $\pi(s,t)$ the score of t when propagating from s. Generalizing it, for any prior vector p and node u we denote by $\pi(p,u)$ the score of u when propagating from p. In the symmetric normalization case, $\pi(s,t) = \pi(t,s)$ for any two nodes s, t, while in the weighted degree normalization case it can be shown that $\pi(s,t)w(s) = \pi(t,s)w(t)$ (see, e.g., Lemma 1 in [13]). The following lemma is key in proving the correctness of the algorithm.

▶ **Lemma 3.1.** *The following equalities (algorithm’s invariants) are equivalent:*

1. $P(u) + \alpha R(u) = (1-\alpha)\sum_{x \in N(u)} P(x)W_{xu} + \alpha p_u$
2. $\pi(p,u) = P(u) + \sum_{x \in V} R(x)\pi(x,u)$

Proof. Denote $\pi = \alpha(I - (1-\alpha)W)^{-1}$. Note that $(\pi p)_u = \sum_{x \in V} \pi(x,u)p_x$, $(\pi R)_u = \sum_{x \in V} \pi(x,u)R(x)$. We can write (2) in vector form as: $\pi p = P + \pi R$. Multiplying both

sides by π^{-1} from the left we get: $p = \pi^{-1}P + R \leftrightarrow \alpha p = P - (1-\alpha)WP + \alpha R$. Rearranging terms we get the desired result. ◀

Using this lemma we can now justify the propagation approximation achieved by the ForwardPush algorithm.

▶ **Lemma 3.2.** *The ForwardPush algorithm maintains the invariants in 3.1.*

Proof. It suffices to prove that the first invariant holds. On initialization, $P = 0^n$ and $R = p$, hence the invariant holds. Suppose an arbitrary node u is pushed and denote by P', R' the updated values of P and R. Since $P'(u) = P(u) + \alpha R(u)$ and $R'(u) = 0$, we have $P'(u) + \alpha R'(u) = P(u) + \alpha R(u)$. Clearly, the right hand side of invariant (1) did not change for u as for any $x \in N(u)$ only $R(x)$ is changed while pushing u. For any other node x that is adjacent to u, $R'(x) = R(x) + (1-\alpha)R(u)W_{xu}$. Thus, the addition to the left hand side of the equation is $\alpha(1-\alpha)R(u)W_{xu}$ which is exactly the addition to the right hand side due to the update of $P(u)$. ◀

▶ **Lemma 3.3.** *For weighted-degree normalization, when ForwardPush() terminates:*

$$\sum_{t \in V} |P(t) - \pi(p,t)| \leq 2\epsilon m$$

Proof. Consider any node t and let e_t be the unit vector with 1 at coordinate t and 0 elsewhere. It follows from lemma 2.2 that for any node $u \in V$, $\sum_{t \in V} \pi(u,t) \leq 1$. By Lemma 3.1 and since $|R|_\infty \leq \epsilon$ upon termination,

$$\sum_{t \in V} |P(t) - \pi(p,t)| = \sum_{t \in V} \sum_{u \in V} |R(u)| \pi(u,t) \leq \sum_{t \in V} \sum_{u \in V} \epsilon d(u) \pi(u,t) = \sum_{u \in V} \epsilon d(u) \sum_{t \in V} \pi(u,t) \leq \sum_{u \in V} \epsilon d(u) = 2\epsilon m$$

◀

Note that the above lemma bounds the overall L1 approximation of the algorithm at $2\epsilon m$, hence ϵ is typically chosen to be smaller than $1/m$ to guarantee a constant overall error. After proving the correctness of the algorithm, we turn to bound its complexity:

▶ **Lemma 3.4.** *Every push of the algorithm, $|R|_1$, the sum of residuals in absolute value, decreases by at least $\alpha\epsilon d(v)$ where v is the node being pushed.*

Proof. Let R, R' denote the residuals before and after the push, respectively. Then

$$\sum_{u \in V} |R(u)| - \sum_{u \in V} |R'(u)| = |R(v)| + \sum_{u \in N(v)} |R(u)| - |R'(u)| \geq \quad (1)$$

$$|R(v)| - \sum_{u \in N(v)} |R'(u) - R(u)| = |R(v)| - (1-\alpha)|R(v)| \sum_{u \in N(v)} |W_{vu}| \geq \quad (2)$$

$$|R(v)| - (1-\alpha)|R(v)| = |R(v)|(1-(1-\alpha)) = \alpha|R(v)| \geq \alpha\epsilon d(v) \quad (3)$$

◀

▶ **Lemma 3.5.** *The ForwardPush algorithm with weighted degree normalization takes $O(\frac{|p|_1}{\alpha\epsilon})$ time.*

Proof. On initialization $\sum_{v \in V} |R(v)| = |p|_1$. Denote by u_i the vertex being pushed at step i of the algorithm. By Lemma 3.4, $\sum_{v \in V} |R(v)|$ decreases by at least $\alpha\epsilon d(u_i)$ at step i, thus the following holds: $\sum_i \alpha\epsilon d(u_i) \leq |p|_1$. Hence, $\sum_i d(u_i) \leq \frac{|p|_1}{\alpha\epsilon}$. As each push step of a node u_i can be implemented in $O(d(u_i))$ time, the total time is $O(\frac{|p|_1}{\alpha\epsilon})$. The residuals with absolute values greater than $\epsilon d(u_i)$ can be maintained in a linked list and an associated array at the same cost. ◀

We can generalize the above algorithm to any similar matrix $W = L^{-1}SL$ where L is an invertible diagonal matrix, S is a stochastic matrix, i.e: $S = wD^{-1}$, and D is the diagonal weighted degree matrix. For example, choosing $L = D^{1/2}$ yields the symmetric normalization. Note that such matrices satisfy the convergence condition as their eigenvalues are the same as those of S. The generalization is based on the following lemma:

▶ **Lemma 3.6.** *(Stochastic Lemma) Let $\psi(W, p, \alpha)$ be the result of network propagation with matrix $W = L^{-1}SL$ over prior p. Then $\psi(W, p, \alpha) = L^{-1}\psi(S, Lp, \alpha)$.*

Proof. Denote by v_n and v'_n the propagation vectors on W and S, respectively, after the nth step of the propagation process. We prove by induction that for every n, $v_n = L^{-1}v'_n$. For the base case $v_0 = p$ and $v'_0 = Lp$ so the claim trivially holds. Suppose the claim is true for n, then: $v_{n+1} = \alpha p + (1-\alpha)Wv_n = \alpha p + (1-\alpha)L^{-1}SL(L^{-1}v'_n) = L^{-1}(\alpha(Lp) + (1-\alpha)Sv'_n) = L^{-1}v'_{n+1}$. ◀

4 A dynamic decremental algorithm for vertex removals

Lemma 3.6 naturally suggests an adaptation for the *ForwardPush* algorithm for any normalization matrix W which is similar to a stochastic matrix. For concreteness and clarity, we will present a dynamic decremental algorithm for removing nodes under weighted symmetric normalization as used in [12]. The main idea is first to maintain a valid propagation result for weighted-degree normalization, and on deletion, using lemma 3.6, we fix the invariant. If this leads to high residuals, we will re-push them over the graph locally, to fix the total scores. In the following we set $L = D^{1/2}$ for symmetric normalization and $p' = LP$. For a removal of any node v, denote by S', D' the resulting matrices, by w' the updated network weights and by π' the updated propagation scores. Further denote by Leaf$N(v)$ the set of neighbors of v of degree 1. W.l.o.g. v has at least one neighbor which is of degree greater than 1, otherwise the connected component of v will vanish after its removal. The algorithm is given below (Algorithm 2). *ForwardPushSym* computes the initial propagation. *FixRemoveEdge* handles edge removals and is used as a subroutine by *VertexRemoveProp* which handles the node removals.

The following lemmas, proved in the Appendix, establish the correctness and accuracy of the algorithm. Denote $\phi := \max_v\{\sqrt{w(v)}, 1/\sqrt{w(v)}\}$. In all our applications reported below, for both yeast and human, $\phi < 36$.

▶ **Lemma 4.1.** *When $ForwardPushSym(W, \alpha, \epsilon, p)$ returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w(t)}} - \pi(p, t)| < 2\epsilon m\phi$ where $\pi(p, t)$ is the propagation score for node t with prior p under symmetric normalization.*

▶ **Lemma 4.2.** *When $VertexRemoveProp(v, R, P, W, \epsilon)$ returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w'(t)}} - \pi'(p, t)| < 2\epsilon m\phi$ where $\pi'(p, t)$ is the propagation score for node t with prior p under symmetric normalization after node v removal.*

Algorithm 2 ForwardPush for symmetric normalization.

1: **procedure** $ForwardPushSym(W = D^{-1/2}SD^{1/2}, \alpha, \epsilon, p)$
2: $R \leftarrow D^{1/2}p, P \leftarrow 0^n$
3: $R, P \leftarrow ForwardPush(S, R, P, \alpha, \epsilon)$
4: **return** R, P
5: **procedure** $FixRemoveEdge(u, v, R, P, W = D^{-1/2}SD^{1/2})$
6: $w(v), w_u(v) := \sum_{x \in V} w_{xv}, \sum_{x \in V, x \neq u} w_{xv}$
7: $F_u(v) = \frac{w(v)}{w_u(v)}$
8: $R(v) += \frac{1}{\alpha}(1 - \frac{1}{F_u(v)})P(v) - \frac{(1-\alpha)}{\alpha}P(u)S_{uv} + (\sqrt{w_u(v)} - \sqrt{w(u)})p_v$
9: $P(v) /= F_u(v)$
10: Remove (u, v) from S and normalize v.
11: **procedure** $VertexRemoveProp(v, R, P, W = D^{-1/2}SD^{1/2}, \epsilon)$
12: **for** $u \in \text{Leaf}N(v)$ **do**
13: $R(u) = 0, P(u) = \alpha p_u$
14: **for** $u \in N(v) \backslash \text{Leaf}N(v)$ **do**
15: $FixRemoveEdge(v, u, R, P, W)$
16: $R(v) = 0, P(v) = \alpha p_v$
17: $ForwardPush(S', R, P, \alpha, \epsilon)$
18: **return** R, P

Algorithm 3 Multiple-vertex removal

1: **procedure** $RemoveNodes(node\_list, W = D^{-1/2}SD^{1/2}, \alpha, \epsilon_0, \epsilon_1, p)$
2: $R, P \leftarrow ForwardPushSym(W, \alpha, \epsilon_0, p)$
3: **for** $u \in node\_list$ **do**
4: Backup R,P
5: $R, P \leftarrow VertexRemoveProp(u, R, P, W, \epsilon_1)$
6: $res(u) \leftarrow D^{-1/2}P$
7: Restore R,P
8: **return** res

Given the vertex removal routine, we can perform n removals and corresponding propagations using Algorithm 3. For efficiency, we can choose different accuracies ($\epsilon_0, \epsilon_1 = \epsilon$) for different stages of the propagation with $\epsilon_0 \leq \epsilon_1$. This allows us to invest more time in the initial propagation in order to reduce the resulting residual sum, leading to time savings in subsequent computations. In order to bound the complexity of the algorithm we denote by R^u the vector of residuals after fixing invariants in $VertexRemoveProp$ (line (16) completion) and define the total sum in absolute value of residual changes following the removal of a vertex u:

$$\Delta R_u := \sum_{v \in V} |R^u(v) - R(v)|$$

▶ **Lemma 4.3.** *The total residual sum being pushed over n vertex removals is bounded by:*

$$T := \sum_{u \in V} (\sum_{v \in V} |R(v)| + \Delta R_u)$$

Proof. The total sum of residuals being pushed when removing a vertex u is bounded by

$\sum_{v\in V\setminus\{u\}} |R^u(v)|$. By the triangle inequality, $\sum\limits_{v\in V\setminus\{u\}} |R^u(v)| \leq \sum\limits_{v\in V} |R(v)| + \Delta R_u$. ◀

▶ **Corollary 4.4.** *The removal of n vertices and subsequent propagations take $O(\frac{T}{\alpha\epsilon_1})$ time.*

▶ **Lemma 4.5.** *Let u be a node being removed, and $F_u(v)$ the expression described in Algorithm (2). Then,*

$$\Delta R_u \leq R(u) + \sum_{v\in LeafN(u)} R(v) + \sum_{v\in N(u)\setminus LeafN(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v +$$
$$(\frac{1-\alpha}{\alpha})P(u) \sum_{v\in N(u)\setminus LeafN(u)} S_{uv} + \frac{1}{\alpha} \sum_{v\in N(u)\setminus LeafN(u)} P(v)(1 - \frac{1}{F_u(v)})$$

Proof. Note that $R(v), P(v)$ are positive for any v as they represent the initial ForwardPush result. Assume that u is removed and consider the changes to its neighbors in VertexRemove(). Clearly, for each leaf neighbor v, the change in residuals is $R(v)$. For non-leaf neighbors v, we can upper bound the residual change by:

$$\frac{1}{\alpha}\left(1 - \frac{1}{F_u(v)}\right) P(v) + |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v + \frac{1-\alpha}{\alpha} P(u) S_{uv}$$

Hence, summing over all of these neighbors and u itself leads to the required result. ◀

To bound the complexity of the algorithm we need the following notation and auxiliary lemmas.

▶ **Lemma 4.6.** $1 - \frac{1}{F_u(v)} = S_{uv}$

Proof. $1 - \frac{1}{F_u(v)} = \frac{\sum\limits_{x\in V} w_{xv}}{\sum\limits_{x\in V} w_{xv}} - \frac{\sum\limits_{x\neq u} w_{xv}}{\sum\limits_{x\in V} w_{xv}} = S_{uv}$ ◀

▶ **Lemma 4.7.** *After the initial ForwardPushSym, $\sum\limits_{v\in V} P(v) \leq \phi$*

Proof. Initially, $\sum_{v\in V} P(v) = 0$ and $\sum_{v\in V} |R(v)| = |p'|_1$. Anytime we increase $P(u)$ by $\alpha R(u)$ for some u, $\sum_{v\in V} |R(v)|$ is reduced by at least $\alpha|R(u)|$ by Lemma 3.4. Therefore, $\sum_{v\in V} |R(v)| \leq |p'|_1$ throughout, implying that $\sum\limits_{v\in V} P(v) \leq |p'|_1$.

By definition, $|D^{1/2}|_1 = \sup\limits_{v\neq 0} \frac{|D^{1/2}v|_1}{|v|_1}$. Hence, $|p'|_1 = |D^{1/2}p|_1 \leq \frac{|D^{1/2}p|_1}{|p|_1}|p|_1 \leq |D^{1/2}|_1|p|_1 \leq \phi$. ◀

▶ **Lemma 4.8.** $\sum\limits_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| \leq \sqrt{w(v)}$

Proof. First, note that:

$$|\sqrt{w_u(v)} - \sqrt{w(v)}| = |\frac{(\sqrt{w_u(v)} - \sqrt{w(v)})(\sqrt{w_u(v)} + \sqrt{w(v)})}{\sqrt{w_u(v)} + \sqrt{w(v)}}|$$
$$= \frac{|w_u(v) - w(v)|}{|\sqrt{w_u(v)} + \sqrt{w(v)}|} \leq \frac{w_{vu}}{\sqrt{w(v)}}$$

Then, summing over all neighbors of v:

$$\sum_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| \leq \sum_{u\in N(v)} \frac{w_{vu}}{\sqrt{w(v)}} = \sqrt{w(v)}$$ ◀

▶ **Lemma 4.9.** $\sum_{u\in V} \Delta R_u \leq \frac{5\phi}{\alpha}$.

Proof. Let R, P be the updated residuals and the estimates after the initial application of ForwardPush() in line 2 of the RemoveNodes() algorithm. By Lemma 4.5, the total changes to the residuals are

$$\overbrace{\sum_{u\in V} R(u) + \sum_{u\in V}\sum_{v\in \text{Leaf}N(u)} R(v)}^{\text{part 1}} + \overbrace{\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v}^{\text{part 2}} +$$
$$\overbrace{\sum_{u\in V}(\frac{1-\alpha}{\alpha})P(u)\sum_{v\in N(u)\setminus \text{Leaf}N(u)} S_{uv}}^{\text{part 3}} + \frac{1}{\alpha}\overbrace{\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} P(v)(1-\frac{1}{F_u(v)})}^{\text{part 4}}$$

We will bound each part separately.

1. By definition, $\sum_{u\in V} R(u) \leq |D^{1/2}p|_1$. For a leaf $v \in V$, its residual will be summed once as its degree is 1. Overall, the sum is bounded by $2\sum_{u\in V} R(u) \leq 2|D^{1/2}p|_1 \leq 2\phi$.
2. By changing the summation order we get
$$\begin{aligned}\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v &= \\ \sum_{v\in V,\text{ not a leaf}} p_v \sum_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| &\overset{4.8}{\leq} \\ \sum_{v\in V,\text{ not a leaf}} p_v\sqrt{w(v)} &\leq |p|_1\phi = \phi\end{aligned}$$
3. we bound this part and get:
$$\sum_{u\in V}(\frac{1-\alpha}{\alpha})P(u)\sum_{v\in N(u)} S_{uv} \leq \frac{(1-\alpha)}{\alpha}\sum_{u\in V} P(u) \overset{4.7}{\leq} \frac{(1-\alpha)}{\alpha}\phi$$
4. Similar to part (1), by changing the order of summation we get that:
$$\begin{aligned}\frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v)\sum_{u\in N(v)}(1-\frac{1}{F_u(v)}) &\overset{4.6}{=} \\ \frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v)\sum_{u\in N(v)} S_{uv} &\leq \\ \frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v) &\leq \phi/\alpha\end{aligned}$$

Overall we obtain a bound of $\frac{5\phi}{\alpha}$. ◀

We are now ready to state our main result:

▶ **Lemma 4.10.** *The total complexity of the propagation algorithm with n vertex removals is* $(\frac{\phi}{\alpha\epsilon_0} + \frac{mn\epsilon_0}{\alpha\epsilon_1} + \frac{\phi}{\epsilon_1}) = O(m\sqrt{n\phi})$

Proof. The initialization takes $O(\phi/(\alpha\epsilon_0))$ time. By Lemmas 4.3 and 4.9, the total time for n vertices removals and subsequent propagations is bounded by:

$$\frac{\sum_{u\in V}(\sum_{v\in V}|R(v)|) + \Delta R_u}{\alpha\epsilon_1} \leq \frac{\sum_{u\in V}(\sum_{v\in V}\epsilon_0 d(v)) + \Delta R_u}{\alpha\epsilon_1} = \frac{\sum_{u\in V}(2m\epsilon_0) + \Delta R_u}{\alpha\epsilon_1} = \frac{2mn\epsilon_0}{\alpha\epsilon_1} + \frac{\frac{5\phi}{\alpha}}{\alpha\epsilon_1}$$

Table 1 Tested Networks and their properties.

Graph	#Nodes	#Edges	Maximal Degree	Average Degree	ϕ
Human (HIPPIE) [4]	19,796	339,788	2173	17.1	35.67
Yeast (ANAT [15])	5138	76,467	2040	13.82	19.9
Human (ANAT [15])	15,517	259,161	2086	15.73	27.75

Table 2 Performance evaluation on a human network (ANAT) upon 1,000 vertex removals.

Algorithm	DynamicFP	ForwardPush	Power Iteration
Time	10.1s	9417.89s	886.54s
#node visits per update	302.24	55,175,737	2,618,256
L1-error	0.00121	0.00119	0.00101

Hence, The complexity is:

$$O(\frac{\phi}{\alpha\epsilon_0}+\frac{mn\epsilon_0}{\alpha\epsilon_1}+\frac{\phi}{\epsilon_1}) \overset{\epsilon_0=\sqrt{\frac{\phi\epsilon_1}{mn}}}{=} O(\sqrt{\frac{mn\phi}{\epsilon_1}}) \overset{\epsilon_1=\Theta(\frac{1}{m})}{=} O(m\sqrt{n\phi})$$

The first equality follows by finding the minimal solution for ϵ_0, ϵ_1, where ϵ_1 is chosen to be c/m to bound the overall L1 error by some additive constant. As the edge weights are at most 1 and at least some positive constant (otherwise, the edges are considered unreliable), $\phi \leq \sqrt{n}$. Hence, the total time is $O(mn^{3/4})$ and the amortized cost per node knockout is $O(\frac{m}{n^{1/4}})$. ◀

5 Results

We benchmark our algorithm, Dynamic ForwardPushSym (DynamicFP), against a static implementation of the power iteration method, as well as against the static ForwardPush algorithm (ForwardPushSym from scratch). We measure the performance of each algorithm in terms of time (seconds) and the number of nodes it visits throughout its execution. In order to evaluate the accuracy of the different algorithms we measured their L1 deviation from the true propagation vector (sum of absolute differences). All tests were done in Intel(R) Xeon(R) E5410 @ 2.33Ghz, 16GB RAM. We used the latest yeast and human protein-protein interaction (PPI) networks from ANAT [15] as well as the human PPI network of [4]. The tested networks and their properties are summarized in Table 1. To perform a comparative analysis of different propagation algorithms, we fixed the propagation parameters to $\alpha = 0.4$ and a prior set size of 100. In order to compute accurate propagation scores against which we could benchmark the different methods, we applied the power iteration method with $\epsilon' = \epsilon/n$. In the comparison itself, we applied the power iteration method with the maximal ϵ that leads to an overall L1 error of at most 10^{-2}. For Forward-Push we used the maximal ϵ_1 that leads to an overall L1 error of at most 10^{-2}. We also set $\epsilon_0 = \sqrt{\frac{\epsilon_1\phi}{mn}}$. The results upon making 1,000 vertex removals, where each vertex is removed separately from the original network, are summarized in Tables 2 and 3. Next, we compared our performance to the previous approach LazyFwdUpdate of [5]. To this end, we used a larger human PPI network from [4] and observed the performance of the two methods upon knockouts of all (non-prior) vertices, simulating a real application of these methods. Since LazyFwdUpdate handles only unweighted networks with degree-based normalization,

Table 3 Performance evaluation on a yeast network (ANAT) upon 1,000 vertex removals.

Algorithm	DynamicFP	ForwardPush	Power Iteration
Time	2.44s	2132.2s	102.91s
#node visits per update	1443.904	22,103,498	765,310
L1-error	0.0013	0.00121	0.0017

Table 4 Performance comparison with $\epsilon_0 = \epsilon_1$ (1) and $\epsilon_0 = \sqrt{\frac{\epsilon_1 \phi}{mn}}$ (2).

Algorithm	LazyFwdUpdate	DynamicFP (1)	DynamicFP (2)
Time	36.01s	23.60	9.71s
#node visits per-update	13,925	8024	2749
L1-error	0.0047	0.0056	0.0061

we used the corresponding variant (unweighted network; weighted-degree normalization) of the *DynamicFP* algorithm. The results, summarized in Table 4, show that our algorithm compares favorably to LazyFwdUpdate, especially when varying the ratio between ϵ_0 and ϵ_1.

6 Conclusions

We have devised a dynamic algorithm for network propagation that can handle a vertex deletion in $O(\frac{m}{n^{1/4}})$ time and provides a speedup of $\Omega(n^{1/4})$ over previous work. Importantly, the algorithm leads to huge speedups of up to a 50-100 fold on real data. Thus, it allows, for the first time, the application of costly methods (such as [12] and [14]) to examine multiple gene knockouts simultaneously. The proposed algorithm can substantially increase the number of different knockout effects that can be simulated in a reasonable time. While our work focused on vertex deletions that are very common in the biological domain, it would be interesting to extend our algorithm to a fully dynamic one. This may result in various applications that concern situations in which edges rather than nodes are perturbed.

References

1 Amy N. Langville and Carl D. Meyer. Deeper Inside PageRank. *Internet Mathematics*, 1(3), jan 2004. doi:10.1080/15427951.2004.10129091.

2 Dengyong Zhou, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Scholkopf. Learning with Local and Global Consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, pages 321–328. MIT Press, 2004. URL: http://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf.

3 Glen Jeh and Jennifer Widom. Scaling Personalized Web Search. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, WWW '03, pages 271–279, New York, NY, USA, 2003. ACM. doi:10.1145/775152.775191.

4 Gregorio Alanis-Lobato, Miguel A. Andrade-Navarro, and Martin H. Schaefer. HIPPIE v2.0: enhancing meaningfulness and reliability of protein–protein interaction networks. *Nucleic Acids Research*, 45(D1):D408–D414, jan 2017. doi:10.1093/nar/gkw985.

5 Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate Personalized PageRank on Dynamic Graphs. *arXiv:1603.07796 [cs]*, 2016. arXiv: 1603.07796. URL: http://arxiv.org/abs/1603.07796.

6 Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web., nov 1999. URL: http://ilpubs.stanford.edu:8090/422/.

7 Lenore Cowen, Trey Ideker, Benjamin J. Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews. Genetics*, 18(9):551–562, 2017. doi:10.1038/nrg.2017.38.

8 Minji Yoon, WooJeong Jin, and U Kang. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. *arXiv:1712.00595 [cs]*, 2017. arXiv: 1712.00595. URL: http://arxiv.org/abs/1712.00595.

9 Monica Bianchini, Marco Gori, and Franco Scarselli. PageRank and Web communities. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pages 365–371, oct 2003. doi:10.1109/WI.2003.1241217.

10 Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient PageRank Tracking in Evolving Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, KDD '15, pages 875–884, New York, NY, USA, 2015. ACM. doi:10.1145/2783258.2783297.

11 Oron Vanunu, Oded Magger, Eytan Ruppin, Tomer Shlomi, and Roded Sharan. Associating Genes and Protein Complexes with Disease via Network Propagation. *PLOS Computational Biology*, 6(1):e1000641, 2010. doi:10.1371/journal.pcbi.1000641.

12 Ortal Shnaps, Eyal Perry, Dana Silverbush, and Roded Sharan. Inference of Personalized Drug Targets via Network Propagation. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 21:156–67, 2016. URL: https://www.semanticscholar.org/paper/Inference-of-Personalized-Drug-Targets-via-Network-Shnaps-Perry/b57104bd662ffeb95bb150d00adb381caffce013.

13 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Bidirectional PageRank Estimation: From Average-Case to Worst-Case. In *Algorithms and Models for the Web Graph - 12th International Workshop, WAW 2015, Eindhoven, The Netherlands, December 10-11, 2015, Proceedings*, WAW 2015, pages 164–176, New York, NY, USA, 2015. Springer-Verlag New York, Inc. doi:10.1007/978-3-319-26784-5_13.

14 Sushant Patkar, Assaf Magen, Roded Sharan, and Sridhar Hannenhalli. A network diffusion approach to inferring sample-specific function reveals functional changes associated with breast cancer. *PLoS Computational Biology 13(11): e1005793, in press*, 13, nov 2017. doi:10.1371/journal.pcbi.1005793.

15 Yomtov Almozlino, Nir Atias, Dana Silverbush, and Roded Sharan. ANAT 2.0: reconstructing functional protein subnetworks. *BMC bioinformatics*, 18(1):495, 2017. doi:10.1186/s12859-017-1932-1.

A Supplementary proofs

▶ **Lemma A.1.** *When ForwardPushSym(W, α, ϵ, p) returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w(t)}} - \pi(p,t)| < 2\epsilon m\phi$ where $\pi(p,t)$ is the propagation score for node t with prior p under symmetric normalization.*

Proof. By Lemma 3.3, *ForwardPush* applied on S with prior p' returns a vector P of

estimated scores, where $\sum_{t\in V}|P(t)-\pi(p',t)| = |P-\pi p'|_1 \le 2\epsilon m$.

$$\begin{aligned} |P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m \rightarrow \\ |P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m|D^{-1/2}|_1|D^{1/2}|_1 \rightarrow \\ |D^{-1/2}|_1|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m|D^{-1/2}|_1 \rightarrow \\ |D^{-1/2}|_1|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m\phi \end{aligned}$$

Since $|D^{-1/2}|_1 = \sup_{v\neq 0}\frac{|D^{-1/2}v|_1}{|v|_1}$ it follows that $|D^{-1/2}v|_1 = \frac{|D^{-1/2}v|_1}{|v|_1}|v|_1 \le |D^{-1/2}|_1|v|_1$. Therefore: $|D^{-1/2}P - D^{-1/2}\psi(S,D^{1/2}p,\alpha)|_1 \le 2\epsilon m\phi$.
Note that $\psi(W,p,\alpha)$ is a vector of propagation scores $\pi(p,t)$ for all nodes t. By the stochastic Lemma 3.6, $\psi(W,p,\alpha) = D^{-1/2}\psi(S,D^{1/2}p,\alpha)$. Thus,

$$|D^{-1/2}P-\psi(W,p,\alpha)|_1 \le 2\epsilon m\phi \rightarrow \sum_{t\in V}|\frac{P(t)}{\sqrt{w(t)}}-\pi(p,t)| \le 2\epsilon m\phi \quad \blacktriangleleft$$

▶ **Lemma A.2.** *When $VertexRemoveProp(v,R,P,W,\epsilon)$ returns, $\sum_{t\in V}|\frac{P(t)}{\sqrt{w'(t)}}-\pi'(p,t)| < 2\epsilon m\phi$ where $\pi'(p,t)$ is the propagation score for node t with prior p under symmetric normalization after node v removal.*

Proof. Denote by R',P',S',D' the changed residuals and estimates, weights and sum of degrees (for an isolated vertex u we define $D^{-1}_{uu} = D_{uu} = 1$), respectively, after line (16) in $VertexRemoveProp$ and $p' := D^{1/2}p$, $p'' := D'^{1/2}p$. Note that $p'_u := \sqrt{w(u)}p_u$, $p''_u := \sqrt{w_v(u)}p_u$ by definition of D' (after node removal). We will first prove the following:

▶ **Lemma A.3.** *The following holds:*
1. *if $x \neq v$ then $P'(x)S'_{xu} = P(x)S_{xu}$*
2. *if $x = v$ and $u \in N(v)$ then $P'(x)S'_{xu} = 0$*
3. *if $x = v$ and $u \notin N(v)$ then $P'(x)S'_{xu} = P(x)S_{xu}$*

Proof.
1. Let $x \neq v$, if $P(x) \neq P'(x)$ then $x \in N(v)$ by algorithm def, hence $P'(x) = P(x)/F_u(v)$ and $S'_{xu} = S_{xu}F_u(v)$. Then for sure $P'(x)S'_{xu} = P(x)S_{xu}$. If $P(x) = P'(x)$ then x is not a neighbor of v, hence by algorithm def. $S'_{xu} = S_{xu}$.
2. Let $x = v$ and $u \in N(v)$, by algorithm def, $S'_{xu} = S'_{vu} = 0$ as the edge was removed, and $P'(x)S'_{xu} = 0$.
3. Let $x = v$ and $u \notin N(v)$, by algorithm def, $P'(x) = P(x)$, as no edge that connects to u was removed, also $S'_{xu} = S_{xu}$ hence $P(x)S_{xu} = P'(x)S'_{xu}$. ◀

We will show that after line (16) in $VertexRemoveProp$ (2), the invariants of $ForwardPush$ (3.1) are kept with respect to p'' as the prior, S' as the weights matrix and α.

- Let $u \in \text{Leaf}N(v)$, then after line (16) node u becomes isolated and $P'(u) + \alpha R'(u) = \alpha p_v = \alpha p''_v$ as required.
- Let $u \in N(v)$, we will show the invariant is kept for u. We know:
$$P(u) + \alpha R(u) = (1-\alpha)\sum_{x\in V}P(x)S_{xu} + \alpha p'_u \tag{4}$$
We want to show:
$$P'(u) + \alpha R'(u) = (1-\alpha)\sum_{x\in V}P'(x)S'_{xu} + \alpha p''_u \tag{5}$$

Hence, it is enough to show that RHS of (5) minus the RHS of (4) equals to the LHS of (5) minus the LHS of (4).
Using Lemma A.3, for any node $x \neq v$ it holds $P'(x)S'_{xu} = P(x)S_{xu}$ and for $x = v$ it holds $P'(x)S'_{xu} = 0$. Then:

$$\begin{aligned}((1-\alpha)\sum_{x\in V} P'(x)S'_{xu} + \alpha p''_u) \quad &- \quad ((1-\alpha)\sum_{x\in V} P(x)S_{xu} + \alpha p'_u) = \\ -(1-\alpha)P(v)S_{vu} \quad &- \quad \alpha(p'_u - p''_u)\end{aligned}$$

By Algorithm 2, line 8:

$$\begin{aligned}(P'(u) + \alpha R'(u)) - (P(u) + \alpha R(u)) \quad &= \quad (P'(u) - P(u)) + \alpha(R'(u) - R(u)) \\ &= \quad -(1 - \frac{1}{F_v(u)})P(u) + \alpha[\frac{1}{\alpha}(1 - \frac{1}{F_v(u)})P(u) \\ &- \quad \frac{(1-\alpha)}{\alpha}P(v)S_{vu} + (\sqrt{w_v(u)} - \sqrt{w(u)})p_u] \\ &= \quad -(1-\alpha)P(v)S_{vu} + \alpha(p''_u - p'_u)\end{aligned}$$

- Let $u \notin N(v)$, using Lemma A.3, $P'(x)S'_{xu} = P(x)S_{xu}$ for any x. By algorithm def. $P'(u) = P(u)$, $R'(u) = R(u)$ and $p'_u = p''_u$. Then:

$$\begin{aligned}P'(u) + \alpha R'(u) \quad &= \quad P(u) + \alpha R(u) = \\ (1-\alpha)\sum_{x\in V} P(x)S_{xu} + \alpha p'_u \quad &= \quad (1-\alpha)\sum_{x\in V} P'(x)S'_{xu} + \alpha p''_u\end{aligned}$$

Therefore, after applying $VertexRemoveProp$ we compute a valid propagation vector over the weights S'' with p'' as prior and α after node v removal. using A.1 we get the same approximation. ◀

New Absolute Fast Converging Phylogeny Estimation Methods with Improved Scalability and Accuracy

Qiuyi (Richard) Zhang[1]
Department of Mathematics, University of California at Berkeley, Berkeley CA 94720
qiuyi@math.berkeley.edu

Satish Rao[2]
Division of Computer Science, University of California at Berkeley, Berkeley CA 94720
satishr@berkeley.edu

Tandy Warnow[3]
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801, USA
warnow@illinois.edu

Abstract

Absolute fast converging (AFC) phylogeny estimation methods are ones that have been proven to recover the true tree with high probability given sequences whose lengths are polynomial in the number of number of leaves in the tree (once the shortest and longest branch lengths are fixed). While there has been a large literature on AFC methods, the best in terms of empirical performance was DCM_{NJ}, published in SODA 2001. The main empirical advantage of DCM_{NJ} over other AFC methods is its use of neighbor joining (NJ) to construct trees on smaller taxon subsets, which are then combined into a tree on the full set of species using a supertree method; in contrast, the other AFC methods in essence depend on quartet trees that are computed independently of each other, which reduces accuracy compared to neighbor joining. However, DCM_{NJ} is unlikely to scale to large datasets due to its reliance on supertree methods, as no current supertree methods are able to scale to large datasets with high accuracy. In this study we present a new approach to large-scale phylogeny estimation that shares some of the features of DCM_{NJ} but bypasses the use of supertree methods. We prove that this new approach is AFC and uses polynomial time. Furthermore, we describe variations on this basic approach that can be used with leaf-disjoint constraint trees (computed using methods such as maximum likelihood) to produce other AFC methods that are likely to provide even better accuracy. Thus, we present a new generalizable technique for large-scale tree estimation that is designed to improve scalability for phylogeny estimation methods to ultra-large datasets, and that can be used in a variety of settings (including tree estimation from unaligned sequences, and species tree estimation from gene trees).

2012 ACM Subject Classification Applied computing → Molecular evolution

Keywords and phrases phylogeny estimation, short quartets, sample complexity, absolute fast converging methods, neighbor joining, maximum likelihood

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.8

[1] Supported by NSF grant 1535989
[2] Supported by NSF grant 1535989
[3] Supported by NSF grant 1535977

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen; Article No. 8; pp. 8:1–8:12

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

1 Introduction

The inference of phylogenies from molecular sequence data is generally approached as a statistical estimation problem, in which a model tree (equipped with a model of sequence evolution) is assumed to have generated the observed data, and the properties of the statistical model are then used to infer the tree. Various statistical approaches can be applied for this estimation, including maximum likelihood, Bayesian techniques, and methods that operate by computing a distance matrix and then computing the tree from the distance matrix.

Many stochastic sequence evolution models have been developed, starting with the Cavender-Farris-Neyman [15, 6, 15] symmetric two-state model (referred to henceforth as "CFN") and including increasingly complex molecular sequence evolution models (with four states for DNA, 20 states for amino acids, and 64 states for codon sequences). However, typically the theory that can be established under the CFN model can also be established under the more complex molecular sequence evolution models used in phylogeny estimation.

Under standard sequence evolution models, many methods are known to be statistically consistent (meaning that they will provably converge to the true tree as the sequence lengths increase), including maximum likelihood [19] and many distance-based methods [1, 23]. A key challenge is to have methods that can scale to large datasets; hence, polynomial time methods are desirable. High accuracy is also needed, so methods that recover the true tree with high probability from limited numbers of sites are best. In this respect, methods that are "absolute fast converging" [8, 9, 25] (i.e., methods that recover the true tree with high probability from polynomial length sequences) are the most promising.

There are several methods that have been established to be absolute fast converging (AFC) under the CFN model, including maximum likelihood [19] (if solved exactly) and various distance-based methods [8, 9, 25, 14, 17, 18, 12, 4]. Some of these algorithms achieve a poly-logarithmic sample complexity but require a balanced model tree and an upper bound on g, the maximum edge length (defining the expected number of changes of a random site) in the CFN model. Specifically, methods based on reconstruction of ancestral sequences provide the best sample complexity bounds but cannot handle the case where g is larger than what is known as the Kesten-Stigum threshold, which is $\ln(\sqrt{2})$ [13] for the CFN model. Other methods, which are AFC in the regime where g is unbounded, are not guaranteed to recover a tree in certain situations [8, 9, 25]. Two of the fastest AFC methods are the Harmonic Greedy Triplets (HGT+FP) method by Csűrös [7] and a method developed by King et al. [11]; these methods use $O(n^2)$ time and are based on quartet trees. Another approach is $\texttt{DCM}_{NJ}$, which uses a divide-and-conquer technique [25]. In the first phase, $O(n^2)$ trees are computed, each based on dividing the sequence dataset into overlapping subsets, constructing trees on each subset using the polynomial time distance-based method neighbor joining (NJ) [20], and then combining the subset trees using a supertree method. This approach differs from other AFC methods in its use of neighbor joining to construct trees on subsets, whereas the other AFC methods in essence construct the tree by independently constructing quartet trees, and then assembling the quartet trees together using a quartet amalgamation method.

Very few AFC methods have been implemented; however, a study [14] comparing `HGT+FP` [7] and $\texttt{DCM}_{NJ}$ [25] showed that $\texttt{DCM}_{NJ}$ had better accuracy despite being slower. Since the theory does not predict this, the results on simulated data suggest that the use of `NJ` to construct trees on subsets and then combine the trees using a supertree method may be empirically advantageous compared to methods that combine quartet trees that are estimated independently. Unfortunately, the reliance on a supertree method to combine subset trees

means that $\texttt{DCM}_{NJ}$ is unlikely to scale to ultra-large datasets, because no current supertree method has shown the ability to maintain good accuracy and reasonable running times on large datasets [24].

The purpose of this paper is to describe a new polynomial time AFC phylogeny estimation method that should improve on $\texttt{DCM}_{NJ}$: it is designed to have comparable accuracy to $\texttt{DCM}_{NJ}$ but also to be able to analyze ultra-large datasets (i.e., more than 100,000 sequences). The basic approach of this new method is similar to $\texttt{DCM}_{NJ}$ in that it uses divide-and-conquer, applies neighbor joining to subsets of the sequence dataset, and then merges the subtrees together. However, it differs from $\texttt{DCM}_{NJ}$ in a few important ways, which we describe below. Most importantly, it divides the taxon set into disjoint subsets and then merges the subset trees without relying on any supertree method; thus, it avoids the challenge of relying on existing supertree methods, none of which are likely to scale to large datasets. We present the results here for the CFN model, but the extension to even complex DNA sequence evolution models is straightforward (e.g., see [9]). Our arguments for correctness are very similar to those in [7, 11]. The rest of the manuscript is organized as follows. We provide background material in Section 2. We present the basic method, INC, in Section 3, and its extension to allow for disjoint constraint trees (computed using various methods) in Section 4. We conclude with a discussion of future work in Section 5.

2 Background Material

2.1 Absolute Fast Convergence under the CFN Model.

Under the Cavender-Farris-Neyman (CFN) Model, we have a rooted binary tree T and substitution probabilities $p(e)$ on the edges e of T. The state at the root is 0 of 1 with equal probability, and the state changes on edge e with probability $p(e)$, with $0 < p(e) < 0.5$ for all edges e. This model can be used for sequence evolution by requiring that all the sites evolve *i.i.d.* down the tree. Finally, we define $w(e) = -\frac{1}{2}\log(1-2p_e)$. We also define $CFN_{f,g}$ to be the set of all CFN model trees (T, Θ) (where Θ denotes the set of numeric parameters on the edges) with $f \leq w(e) \leq g$ for all edges in T, for arbitrarily selected positive real numbers $f \leq g$.

▶ **Definition 1.** A phylogeny estimation method Φ is said to be absolute fast converging (**AFC**) under the CFN model if, for all positive values f, g, ϵ (with $f \leq g$), there is a polynomial p such that for all CFN model trees (T, Θ) in $CFN_{f,g}$ the method Φ will recover the unrooted tree topology T given sequences of length $p(n)$ with probability at least $1 - \epsilon$. Note that the polynomial will in general depend on f, g and ϵ.

2.2 DCM_{NJ}+SQS: an AFC method with good empirical performance but low scalability

Here we describe the approach used in [25] called "DCM_{NJ}+SQS", which was shown to have high accuracy in simulation studies with up to 1600 sequences [14]. (Note: "DCM" refers to "disk-covering method" and "SQS" refers to the "short quartet support" criterion; see [25, 23]). The input is a set of sequences generated by an unknown model tree and a dissimilarity matrix d (i.e., a symmetric matrix that is zero on the diagonal) where d_{ij} is the estimated distance between taxa s_i and s_j, based on the selected sequence evolution model. For example, when the model is CFN, then $d_{ij} = -\frac{1}{2}\ln(1-2H_{ij})$ will be the "empirical CFN distance", where H_{ij} is the Hamming distance between sequences i and j divided by the sequence length (i.e., the normalized Hamming distances). Note that these empirical

CFN distances converge in probability to $D_{ij} = -\frac{1}{2}\log(1-2E_{ij})$ where E_{ij} is the expected normalized Hamming distance between leaves i and j, and that D is an additive matrix for the model tree. $\texttt{DCM}_{NJ}$+SQS uses a two-phase structure, as follows:

- Phase 1: A set $\mathcal{T}$ of $O(n^2)$ trees is computed, with at most one tree t_q for each entry q in the dissimilarity matrix d.
- Phase 2: All the trees in $\mathcal{T}$ are scored using the SQS criterion (where "SQS" refers to the short quartet support, defined in [25]) and the best-scoring tree is returned.

Before we can describe these phases, we need to provide some definitions:

▶ **Definition 2.** (From [10].) For the given dissimilarity matrix d and positive real number q, we define the threshold graph $TG(d, q)$ to be the graph with the n taxa as the vertex set and edges (i, j) if and only if $d_{ij} \leq q$. We also assign weight d_{ij} to each edge (i, j) in $TG(d, q)$.

We use a standard technique, called the Four Point Method, to compute quartet trees (i.e., unrooted binary trees on four leaves) that is based on the Four Point Condition [5].

▶ **Definition 3.** (From [8]) Given a four taxa set $\{u, v, w, z\}$ and a dissimilarity matrix d, the Four Point Method (FPM) infers tree $uv|wz$ (meaning the quartet tree with an edge separating u, v from w, z) if $d(u, v) + d(w, z) \leq \min\{d(u, w) + d(v, z), d(u, z) + d(v, w)\}$. If equality holds, then the FPM infers an arbitrary topology.

Phase 1 for $\texttt{DCM}_{NJ}$+SQS is performed as follows. Given q (the selected threshold), a threshold graph $TG(d, q)$ is computed, then edges are added to the graph to make it triangulated (if necessary), where a triangulated graph is one that has no simple cycles of size four or more; furthermore, if d is additive, then $TG(d, q)$ is triangulated. Once the triangulated graph is computed, the set of all maximal cliques can be extracted in polynomial time. See [25] for additional details and proofs.

Trees are computed for each maximal clique using neighbor joining [20], and the trees on these cliques are then combined into a tree on the full set of species using a selected supertree method. Phase 2 uses the SQS criterion, but other criteria also have good theoretical properties. The SQS score of a tree T, defined by $SQS(T)$, is the maximum l such that for all quartets $\{u, v, w, x\}$ of taxa with diameter at most l, the Four Point Method on $\{u, v, w, x\}$ produces a four-leaf tree that agrees with the T.

▶ **Theorem 4** (From [25, 14])**.** *The short quartet support (SQS) criterion score can be calculated for each tree $t_q \in \mathcal{T}$ in polynomial time. Also, there is a polynomial $p(n)$ such that if $\mathcal{T}$ contains the true tree T and the sequences are of length $p(n)$ (where n is the number of leaves), then with probability at least $1 - \epsilon$, the tree in $\mathcal{T}$ with the highest SQS score is the true tree.*

The basic approach has good theoretical guarantees (i.e., it is AFC, and more generally if X is any exponentially converging base method (meaning X recovers the tree with high probability given exponentially many sites) and Y is any true tree selection criteria that has the same theoretical properties as SQS (as given in Theorem 4), then $\texttt{DCM}_X$+Y is AFC. Empirical performance of $\texttt{DCM}_{NJ}$+SQS on simulated data was excellent, substantially outperforming NJ and fast triplet/quartet-based greedy tree growing methods, such as HTP+FP, on large datasets especially when there is a high rate of evolution [14]. However, these studies were limited to datasets with at most 1600 sequences. In other words, $\texttt{DCM}_{NJ}$+SQS was not tested on very large datasets, which is to some extent the point of absolute fast converging methods.

Furthermore, the design of $\texttt{DCM}_{NJ}$ suggests some definite problems in terms of scalability to large datasets. Most importantly, supertree methods do not have good scalability, as all

current supertree methods with good accuracy are attempts to solve NP-hard optimization problems, and so become computationally intensive on large datasets [24]. Any reasonably fast method will need to completely avoid the supertree calculation step.

3 The basic technique: Incremental Tree-Building

We begin by describing the INC method in the unconstrained condition, which uses short quartet trees to build a tree, and we prove that the method is AFC. We then show how INC can be modified to take an arbitrary set of disjoint constraint trees, and we prove that when the constraint trees are constructed using an AFC method then the result is also AFC. We then present the $O(n^2)$ INC-NJ method, which is AFC and uses neighbor joining on carefully selected subsets to achieve this optimal running time. We then discuss the use of constrained-INC with maximum likelihood methods (or other computationally intensive but highly accurate methods) to construct the subset trees.

Throughout this section, we denote d_{ij} to be the empirical CFN dissimilarity between taxa i, j and D_{ij} to be the underlying CFN model distance between taxa i, j defined by the model tree (T, Θ). Given matrices d and D and positive real q, we define the $\epsilon(q) = \max\{|d_{ij} - D_{ij}| : d_{ij} \leq q \text{ or } D_{ij} \leq q\}$.

We now describe the fast incremental tree-growing method (INC) that inserts taxa in a specific order and outputs a binary tree. We note that the structure of our method is similar to that of HGT+FP. INC achieves a faster runtime when compared to DCM_{NJ} for two primary reasons. First, our method simply requires a minimum spanning tree of our dissimilarity matrix without the need to triangulate and compute maximal cliques, as well as iterate over multiple thresholding values. Second, our method uses the spanning tree as a guide for an incremental tree-growing method that runs in $O(n^2)$, as opposed to the slower runtimes required by NJ techniques. Together, this algorithm runs in the optimal $O(n^2)$ time.

The input is the dissimilarity matrix d (which is based on computing CFN distances, and so will converge to an additive matrix as the sequence length increases). We begin by constructing the minimum spanning tree S of $TG(d, \infty)$ and use S during an incremental tree growing procedure that inserts taxa one by one in an order such that the subgraph induced by the inserted vertices is connected in S. We let q_0 be the maximum distance of an edge in S. We will show that by setting $q = 8q_0$, we can develop a tree construction method based on this threshold q that runs in polynomial time and is AFC. Specifically, we order the vertices as follows.

Insertion Ordering: Choose an arbitrary leaf in S to be the starting vertex in the ordering. Then, order the vertices according to order of traversal in a BFS (or DFS) from the starting vertex.

We initialize our tree t by choosing the first three taxa according to our insertion ordering and forming the three-leaf tree with an internal node that connects to all three taxa. Now suppose we wish to add a vertex x into t. We will select the edge of t into which we add x by performing a collection of "short quartet queries" (see [8, 9] for definition and discussion of short quartets) at the different internal nodes of t, and then insert x into the edge that has the highest support (in the best case, it "agrees" with each query). Note that this node query insertion procedure was explored earlier in [3, 22] to produce a fast $O(n \log n)$ tree-growing algorithm, but analyzed under a different model of quartet tree error than we use here and a direct implementation of their model does not produce an AFC algorithm.

Given an internal node u of t, because t is binary the removal of u splits t into three non-empty vertex components t_1, t_2, t_3 (with internal nodes included). Because the leaves of t form a connected induced subtree of S, we can find some taxon u_i in t_i such that (u_i, v_i) is an edge in S and $v_i \in V(t) \setminus t_i$ is also a taxon, where $V(t)$ is the vertex set of t.

▶ **Definition 5.** Let u be an internal node of a binary tree t and let u_1, u_2, u_3 be leaves in the three components of t upon removing u such that v_i is not in the same component as u_i and there exists v_i with (u_i, v_i) an edge in S. Then, for some choice of q, a quartet query on $\{u_1, u_2, u_3, x\}$ is **q-valid** iff the d-diameter (maximum pairwise distance with respect to the input matrix d) of $\{u_1, u_2, u_3, x\}$ is less than q. We use **valid** when q is clear from context.

By looking at all the valid queries and the corresponding quartet trees calculated using the Four Point Method, we can choose an edge e in the tree t on which to add x that maximizes the support for the placement; we then subdivide e and then make x adjacent to the node created. To choose an edge, each valid short quartet query identifies some subset of edges of t into which x should be placed. Specifically, we assign votes according to a valid query as follows:

Vote Assignment: If the valid query on internal node u shows that t_i and x should be on the same side of the quartet using the Four Point Method, then all edges in the induced subtree on $t_i \cup u$ will receive a vote.

The support for an edge e is then the total number of votes it receives from all the short quartets, and the edge that receives the highest number of votes wins. If there is a tie, then one of the edges receiving the most votes is selected (randomly). However, for favorable values of d and q, there will be a unique edge on which all queries agree, and so the algorithm will correctly add x into t (in such a way that it agrees with the model tree T), and so inductively the greedy algorithm will construct the model tree. In particular, we show that for our choice of $q = 8q_0$, if $\epsilon(q) < f/2$, then there is always a unique edge chosen by this voting procedure, and furthermore that unique edge is the correct edge. Hence, this greedy algorithm will correctly reconstruct the true tree.

$t =$ `INC`(d)
1. Calculate a spanning tree S of $TG(d, \infty)$.
2. Set $q = 8q_0$, where q_0 is the maximum edge length of S.
3. Find the insertion ordering according to a `BFS` of S.
4. Initialize t as the three-leaf tree on the first three taxa in the ordering.
5. Insert each taxon according to the ordering on the edge with the most votes from valid queries.
6. Return the resulting tree t.

▶ **Theorem 6.** *Let d be a dissimilarity matrix, D an additive matrix defining a model tree T with edge weights $w : E(T) \to \mathbb{R}^+$, and $q = 8q_0$, where q_0 is the maximum distance in the minimum spanning tree of $TG(d, \infty)$. Let f be the length of the shortest internal edge in T and suppose $\epsilon(q) < f/2$ and $q_0 \geq f/2$, then `INC`(d) returns T.*

Proof. We proceed by induction on the size of T. Our claim holds when T is size 3, since there is only one such topology. Let t be the tree maintained by `INC` before the insertion of the last taxon x according to our insertion ordering. By induction, t must have the correct topology. It suffices to show that valid queries can determine the accurate placing of x. Assume that the correct location on which to place x is $e = (u, v)$ of t. Note that when

$\epsilon(q) < f/2$, all valid node queries will return the correct quartet tree. Therefore, e will receive all possible votes.

To show that all other edges will miss at least one vote, it suffices to show that node queries are valid at u, v because all other edges will miss a vote from one of the two such queries, confirming that x is inserted at e (if one of u, v is a leaf vertex, the same conclusion is achieved).

We will show that a node query at u is valid; a similar argument is done for v. Let u_1, u_2, u_3 be selected upon the deletion of u with their corresponding v_i. First, for all i, $D(u_i, v_i) \leq q_0 + f/2$ since $d(u_i, v_i) \leq q_0$ and $\epsilon(q) \leq f/2$. Then, since $v_i \in V \setminus t_i$ and t is the correct tree topology, $D(u_i, u) \leq D(u_i, v_i) \leq q_0 + f/2$, where D is the true distance matrix and u is the corresponding internal node in T.

Next, we claim that $D(x, u) \leq 2q_0 + f$ and together we will have concluded that $\{u_1, u_2, u_3, x\}$ is a quartet of d-diameter at most $3q_0 + 2f$ since by triangle inequality, we deduce $\{u_1, u_2, u_3, x\}$ is of D-diameter $3q_0 + 3f/2$ and so the d-diameter is bounded at $3q_0 + 2f \leq 7q_0 \leq q$, proving validity.

For the last claim, since x is within q_0 of some leaf x' in t, $D(x, x') \leq d(x, x') + f/2 \leq q_0 + f/2$. Since the path x to x' must pass through u or v in T and $e = (u, v)$ is the correct edge insertion location, we conclude that $\min(D(x, u), D(x, v)) \leq q_0 + f/2$.

If $D(x, u) \leq q_0 + f/2$, then our claim follows (we automatically get $D(x, u) \leq 2q_0 + f$). Else, $D(x, v) \leq q_0 + f/2$. Since $v \in V(t_3)$, $D(x, u) \leq D(x, v) + D(u, v) \leq q_0 + f/2 + D(u, v) \leq q_0 + f/2 + D(u, u_3) \leq q_0 + f/2 + q_0 + f/2 \leq 2q_0 + f$.

Therefore, if the correct edge is u, v, then the query at u votes against all edge placements for x in $t_1 \cup t_2$, and symmetrically for v. Hence all other edges will miss at least one vote. ◀

▶ **Theorem 7.** *Given the $n \times n$ dissimilarity matrix d and a given q, `INC`(d) can be implemented in $O(n^2)$.*

Proof. It suffices to show that as we grow our tree t, each insertion step can be implemented to take $O(n)$ time. First, each internal node, when initialized due to an inserted taxon, can store the necessary vertices u_1, u_2, u_3 and that can be done in $O(n)$ time. Each node query is $O(1)$, so in total the $|V(t)|$ node queries take $O(n)$ time.

The only possible difficulty is efficiently finding the edge with the most votes. A naive implementation will lead to an $O(n^3)$ runtime. For any edge $e = (u, v)$ on the tree, let $n(e)$ denote the total number of votes that e has. Note that if e, e' are adjacent edges in t with common vertex u , then $n(e) - n(e')$ can be determined by simply looking at the short quartet query at u. Specifically, let $e = (u, v)$ and $e' = (u, v')$ be adjacent edges at vertex u. If the query at u was invalid or it showed that x should be in t_j with $v, v' \notin t_j$, then $n(e) - n(e') = 0$. Otherwise, if the query returned t_j with $v \in t_j$, then $n(e) - n(e') = 1$; a similar argument shows that if $v' \in t_j$ then $n(e) - n(e') = -1$. Therefore, by using this local property, we can calculate $n(e) + C$ for some constant C in $O(n)$ time by performing `BFS` starting from any leaf of the tree. The `BFS` then simply returns the edge with the highest score. ◀

▶ **Theorem 8.** *[2](Azuma's Inequality) Suppose $X = (X_1, X_2, ..., X_k)$ are independent random variables taking values in any set S, and let $L : S^k \to \mathbb{R}$ be any function that satisfies the condition: $|L(u) - L(v)| \leq t$ whenever u and v differ in just one coordinate. Then,*

$$P(|L(X) - E[L(X)]| \geq \lambda) \leq 2\exp(-\frac{\lambda^2}{2t^2k}).$$

▶ **Theorem 9.** *For $k = \Omega(\frac{\ln(n)e^{4q}}{f^2})$, with high probability, we have $\epsilon(q) < f/2$ for all $q = O(g \log n)$. Furthermore, if q_0 is the minimum value of q such that $TG(d, q)$ is connected, then $q_0 = O(g \log n)$. and $q_0 \geq f/2$.*

Proof. To show that $\epsilon(q) < f/2$, we must show that $|D_{ij} - d_{ij}| < f/2$ when $D_{ij} < q$ or $d_{ij} < q$. First, if $D_{ij} < q$ we show that $k = \Omega(\frac{\ln(n)e^{2q}}{f^2})$ suffices to show $|D_{ij} - d_{ij}| < f/2$ with high probability. We express our probability of failure as:

$$\begin{aligned} P(|D_{ij} - d_{ij}| \geq f/2) &= P(|\log(\frac{1-2H_{ij}}{1-2E_{ij}})| \geq f) \\ &\leq P((1-2E_{ij})e^{-f} \geq 1-2H_{ij}) + P(1-2H_{ij} \geq (1-2E_{ij})e^{f}) \end{aligned}$$

The first expression can be written as:

$$\begin{aligned} P(H_{ij} - E_{ij} \geq \frac{1}{2}(1-e^{-f})(1-2E_{ij})) &\leq P(H_{ij} - E_{ij} \geq \frac{1}{2}(1-e^{-f})e^{-2D_{ij}}) \\ &\leq 2\exp(-\Omega(k(1-e^{-f})^2 e^{-2q})) \\ &\leq 2\exp(-\Omega(\ln n)) \end{aligned}$$

The second line follows from Azuma's with $t = 1/k$. Similarly, the second expression is equivalent to:

$$\begin{aligned} P(E_{ij} - H_{ij} \geq \frac{1}{2}(e^{f}-1)(1-2E_{ij})) &\leq P(E_{ij} - H_{ij} \geq \frac{1}{2}(e^{f}-1)e^{-2D_{ij}}) \\ &\leq 2\exp(-\Omega(k(e^{f}-1)^2 e^{-2q})) \\ &\leq 2\exp(-\Omega(\ln n)) \end{aligned}$$

Next, if $d_{ij} < q$, then we show that $D_{ij} < 2q + 1$ with high probability and then apply the previous result with $q' = 2q + 1$. First, if we let $r_{ij} = D_{ij} - q$, then by simple algebra our probability that $d_{ij} < q$ when $r_{ij} > q + 1$ is bounded by

$$\begin{aligned} P(d_{ij} < q) &= P(D_{ij} - d_{ij} > D_{ij} - q) \\ &= P(-\frac{1}{2}\log(\frac{1-2E_{ij}}{1-2H_{ij}}) > D_{ij} - q) \\ &= P(1-2H_{ij} > e^{2r_{ij}}(1-2E_{ij})) \\ &= P(E_{ij} - H_{ij} > \frac{1}{2}(e^{2r_{ij}}-1)e^{-2D_{ij}}) \\ &= P(E_{ij} - H_{ij} > \frac{1}{4}e^{2r_{ij}-2D_{ij}}) \\ &\leq 2\exp(-\Omega(ke^{-2q})) \\ &\leq 2\exp(-\Omega(\ln n)) \end{aligned}$$

The fourth to fifth line follows since $e^{2r_{ij}} - 1 > \frac{1}{2}e^{2r_{ij}}$ whenever $r_{ij} > 1$. Therefore, we conclude our claim that $\epsilon(q) < f/2$ with high probability.

We now show that $q_0 = O(g \log n)$. Note that in our model tree, since g is the maximum length of an edge in a binary tree with n leaves, it follows that $TG(D, O(g \log n))$ is connected. By our previous part, we know that $|d_{ij} - D_{ij}| < f/2$ for all edges in $TG(D, O(g \log n))$. Therefore, we conclude that $TG(d, O(g \log n))$ is also connected, and so $q_0 = O(g \log n)$.

Lastly, we show $q_0 \geq f/2$. Consider all edges in the minimum spanning tree of $TG(d, \infty)$. These edges have true length (in D) at least f and by our previous part the length of the edges in the minimum spanning tree deviate from the true length (as defined by the model tree) by at most $f/2$ with high probability. Thus, we conclude that $q_0 \geq f/2$. ◀

▶ **Theorem 10.** *INC is absolute fast converging for the CFN model and takes $O(n^2)$ time (assuming distances are precomputed).*

Proof. All we need to establish is that for every triplet ϵ, f, g with $0 < f < g < \infty$ and $\epsilon > 0$, there is a polynomial $p(n)$ such that for all model trees (T, Θ) in $CFN_{f,g}$, given sequences of length at least $p(n)$ generated by (T, Θ) and empirical CFN dissimilarity matrix d computed on these sequences, the tree returned by $\texttt{INC}(d)$ is the model tree T with probability at least $1 - \epsilon$.

Let q_0 be the maximum edge length on the minimum spanning tree. By Theorem 9, with high probability, we know $q_0 = O(g \log n)$ and thus $q = 8q_0 = O(g \log n)$. Thus, when $k = \Omega(\ln(n)e^{4q}/f^2) = poly(n)$, we have $\epsilon(q) < f/2$. Furthermore, $q_0 \geq f/2$. Therefore, by Theorem 12, the insertion algorithm will return the model tree T. Hence, INC is absolute fast converging for the CFN model.

Finally, given the dissimilarity matrix, the running time to compute the constraint trees andthe minimum spanning tree and to run the insertion algorithm are all $O(n^2)$ by Theorem 7. ◀

The method we described runs in $O(n^2)$ time and is AFC. As shown by [11], any algorithm that reconstructs the true tree with high probability and uses distance calculations as its only source of information about the phylogeny on sequences of length $O(poly \log n)$ will have $\Omega(n^2)$ runtime, up to logarithmic factors. Hence, our runtime is optimal.

4 Boosting INC using constraint trees

We show how INC can be modified to take an arbitrary set of disjoint constraint trees, and we prove that when the constraint trees are constructed using an AFC method then the result is also AFC. We present several variants of this method: one that optimizes speed while still guaranteeing that the resultant algorithm is AFC, and other AFC variants that are designed for improved accuracy but with an increase in running time.

4.1 Constrained INC

The input to the constrained version of INC includes a set of leaf-disjoint trees. Therefore, the constraint set is compatible (i.e., a compatibility tree exists). We will describe a straightforward modification to INC so that it never violates the topological information in the constraint trees, by which we mean only that the final output tree is required to induce each constraint tree, when restricted to that specific set of leaves. Thus, the constraint trees are not required to be clades in the output tree.

Before proceeding, we define the **induced tree topology**, t^A, on a tree t and a subset of leaves A as follows. Consider the minimal subtree of t that contains the leaves A, and then suppress all nodes of degree two (i.e., replace all maximal paths of degree two nodes by a single edge). For an edge $e \in t^A$, note that its endpoints correspond to vertices in t, and that e corresponds to a (possibly length 1) path in t; we denote the corresponding endpoints of e as $e_t(u)$ and $e_t(v)$. For an induced tree t^A, the **component in t corresponding to an edge** $e \in t^A$ is the set of edges and vertices that can be reached by a walk starting from $e_t(u)$ and ending at $e_t(v)$ without using $e_t(u)$ or $e_t(v)$ multiple times. Note this will be a subgraph of t that includes $e_t(u)$ and $e_t(v)$ as leaves.

When working with constraint trees, we alter the insertion algorithm to account for the constraints. Recall that the constraint trees are on disjoint sets of taxa. Thus, when inserting a taxon, we identify the corresponding constraint tree t_c that includes that new taxon. Let

A be the taxa shared between t_c and t and note that $t_c^A = t^A$ since the trees are consistent. Thus, in $t_c^{A\cup\{x\}}$, the taxon x can be viewed as being attached to some edge e in t^A. The eligible edges are in the component in t corresponding to e. We then simply run `INC` on the component to find the desired edge of insertion.

4.2 INC-NJ

We continue by presenting the $O(n^2)$ `INC-NJ` method, which is AFC and uses neighbor joining on carefully selected subsets to achieve this running time. Given the selected threshold q, we take our threshold graph $TG(d, q)$ and compute a greedy disjoint clique decomposition, such that each clique is of size $O(\sqrt{n})$. This is done by a simple ball-growing procedure that is interrupted when the ball includes more than $O(\sqrt{n})$ vertices; it is easy to see this takes $O(n^2)$ time. Next, we run `NJ` on each of these clique; this produces a set of neighbor joining subset trees that will serve as our constraint trees. Note that since the subsets are disjoint, the subset trees are compatible (in that a compatibility supertree exists). Finally, if $\epsilon(q) < f/2$, then the following theorem guarantees that all constraint trees produced by our technique are correct. Furthermore, since `NJ` applied to each clique of size $O(\sqrt{n})$ takes at most $O(n^{1.5})$ time, our total runtime over all cliques is $O(n^2)$.

▶ **Theorem 11.** *[From [1]] Neighbor Joining (`NJ`) will recover the true tree T whenever the input matrix $\mathbf{d}$ satisfies $L_\infty(d, D) < f/2$, where D is an additive matrix defining an edge-weighted version of the true tree T and f is the shortest internal branch length in T. Furthermore, the runtime is $O(n^3)$.*

▶ **Theorem 12.** *`INC-NJ` is AFC and takes $O(n^2)$ (assuming distances are precomputed).*

Proof. Follows directly from combining Theorem 10 and 11. ◀

4.3 Boosting INC using other constraint trees

The version of constrained-`INC` we presented in the previous section achieves an optimal running time, but is based on obtaining constraint trees using neighbor joining on small subsets. However, we could modify the decomposition to produce larger subsets and thus take advantage of the likely improvement in accuracy obtained by using neighbor joining on these larger subsets to produce the constraint trees. This would still produce a polynomial time algorithm, but one with a higher (and hence suboptimal) running time.

Another variation would be to use other phylogeny estimation methods besides neighbor joining. In particular, maximum likelihood could be used to construct the constraint trees. As shown in [19], maximum likelihood under the CFN model is AFC. Hence, running maximum likelihood on any subsets of the taxon set will provide correct subset trees from polynomial length sequences with high probability. Hence, the same approach we describe for use with neighbor joining (where that subset trees are computed using neighbor joining) can be modified to be used with maximum likelihood, and will be an AFC method. Furthermore, because maximum likelihood is AFC, we can afford to use maximum likelihood on arbitrarily large subsets, without losing the overall AFC property (as we show below); the only negative impact of doing this would be running time, since maximum likelihood is NP-hard [16] and heuristics for maximum likelihood are computationally more intensive than neighbor joining.

▶ **Theorem 13.** *If the disjoint constraint trees are estimated by a method that is AFC under the CFN model, then constrained `INC` with the estimated constraint trees is AFC under the CFN model. Hence, using maximum likelihood to produce constraint trees and combining the constraint trees using `INC` is AFC under the CFN model.*

Proof. Let Φ be an AFC method, and fix ϵ, f, g and some CFN model tree (T, Θ). Hence, we can find a polynomial $p_1(n)$ (where the degree may depend on f, g) such that given sequences of length $p_1(n)$ guarantees that with probability at least $1 - \epsilon$, all the constraint trees computed using Φ are correct. Then, by Theorem 10, we can find $p_2(n)$ (where the degree may depend on f, g) such that `INC` returns the model tree with probability at least $1 - \epsilon$ given sequences of length $p_2(n)$. Hence, given sequences of length $\max\{p_1(n), p_2(n)\}$, constrained `INC` returns the model tree with probability at least $1 - 2\epsilon$. Thus, constrained `INC` is AFC. By [19], maximum likelihood is AFC. Hence, using `INC` with disjoint maximum likelihood constraint trees is AFC, for all ways of decomposing the dataset into disjoint subsets. ◀

5 Discussion

This paper presents a novel algorithmic technique for constructing phylogenetic trees, which allows statistically consistent and highly accurate methods to be used on subsets of the taxa in a divide-and-conquer framework. We proved that several of these variants are absolute fast converging (AFC) under the CFN sequence evolution model, and that some of these achieve $O(n^2)$ time (where n is the number of sequences) once the dissimilarity matrix relating the sequences is computed. Although this theory was presented for the CFN model, the extension to the Jukes-Cantor DNA sequence evolution models is trivial (i.e., just substitute the CFN distances by Jukes-Cantor distances), and equally simple for more complex models, provided that statistically consistent distance estimators exist (see discussion in [23]). Fortunately, all the standard sequence evolution models, including the General Markov Model [21], have such distance estimators. More generally, tree estimation methods could be scaled to large datasets using this approach, provided that it is possible to compute a matrix of pairwise distances that is guaranteed to converge to an additive matrix as the amount of data increases. The goal of this work was improved empirical performance, compared to prior AFC methods. Thus, additional work is needed to evaluate the empirical benefits of this approach in these various applications.

References

1. Kevin Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25(2-3):251–278, 1999.
2. Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.
3. Daniel G Brown and Jakub Truszkowski. Fast error-tolerant quartet phylogeny algorithms. In *Annual Symposium on Combinatorial Pattern Matching*, pages 147–161. Springer, 2011.
4. Daniel G Brown and Jakub Truszkowski. Fast phylogenetic tree reconstruction using locality-sensitive hashing. In *Workshop on Algorithms for Bioinformatics (WABI)*, pages 14–29. Springer, 2012.
5. Peter Buneman. A note on the metric properties of trees. *Journal of Combinatorial Theory (B)*, 17:48–50, 1974.
6. James A Cavender. Taxonomy with confidence. *Mathematical biosciences*, 40(3-4):271–280, 1978.
7. Miklós Csűrös. Fast recovery of evolutionary trees with thousands of nodes. *Journal of Computational Biology*, 9(2):277–297, 2002.
8. Peter L. Erdös, Michael A. Steel, Laszlo Székely, and Tandy Warnow. A few logs suffice to build (almost) all trees (i). *Random Structures and Algorithms*, 14:153–184, 1999.

9 Peter L. Erdös, Michael A. Steel, Laszlo Székely, and Tandy Warnow. A few logs suffice to build (almost) all trees (ii). *Theoretical Computer Science*, 221:77–118, 1999.

10 Daniel Huson, Scott Nettles, Laxmi Parida, Tandy Warnow, and Shibu Yooseph. The disk-covering method for tree reconstruction. In *Algorithms and Experiments (ALEX)*, pages 62–75, 1998.

11 Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 444–453. Society for Industrial and Applied Mathematics, 2003.

12 Radu Mihaescu, Cameron Hill, and Satish Rao. Fast phylogeny reconstruction through learning of ancestral sequences. *Algorithmica*, 66(2):419–449, 2013.

13 Elchanan Mossel. Phase transitions in phylogeny. *Transactions of the American Mathematical Society*, 356(6):2379–2404, 2004.

14 Luay Nakhleh, Usman Roshan, Katherine St. John, Jerry Sun, and Tandy Warnow. Designing fast converging phylogenetic methods. *Bioinformatics*, 17(suppl_1):S190–S198, 2001.

15 Jerzy Neyman. Molecular studies of evolution: a source of novel statistical problems. In *Statistical decision theory and related topics*, pages 1–27. Elsevier, 1971.

16 Sébastien Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(1):92–94, 2006.

17 Sébastien Roch. Sequence-length requirement for distance-based phylogeny reconstruction: Breaking the polynomial barrier. In *Proc. of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 729—-738, 2008.

18 Sébastien Roch. Towards extracting all phylogenetic information from matrices of evolutionary distances. *Science*, 327(5971):1376–1379, 2010.

19 Sébastien Roch and Allan Sly. Phase transition in the sample complexity of likelihood-based phylogeny inference. *Probability Theory and Related Fields*, 169(1):3–62, Oct 2017.

20 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.

21 Michael A. Steel. Recovering a tree from the leaf colourations it generates under a Markov model. *Applied Mathematics Letters*, 7(2):19 – 23, 1994.

22 Jakub Truszkowski, Yanqi Hao, and Daniel G Brown. Towards a practical O(n log n) phylogeny algorithm. *Algorithms for Molecular Biology*, 7(1):32, 2012.

23 Tandy Warnow. *Computational Phylogenetics: An introduction to designing methods for phylogeny estimation.* Cambridge University Press, 2018.

24 Tandy Warnow. Supertree construction: Opportunities and challenges. *arXiv:1805.03530 [q-bio.PE]*, 2018.

25 Tandy Warnow, Bernard M.E. Moret, and Katherine St. John. Absolute convergence: true trees from short sequences. In *Proceedings of SODA*, pages 186–195. ACM, 2001.

An Average-Case Sublinear Exact Li and Stephens Forward Algorithm

Yohei M. Rosen[1]
University of California, Santa Cruz, California
New York University School of Medicine, New York, New York
yohei@ucsc.edu
https://orcid.org/0000-0002-3870-2169

Benedict J. Paten
University of California, Santa Cruz, California
bpaten@ucsc.edu

Abstract

Hidden Markov models of haplotype inheritance such as the Li and Stephens model allow for computationally tractable probability calculations using the forward algorithms as long as the representative reference panel used in the model is sufficiently small. Specifically, the monoploid Li and Stephens model and its variants are linear in reference panel size unless heuristic approximations are used. However, sequencing projects numbering in the thousands to hundreds of thousands of individuals are underway, and others numbering in the millions are anticipated.

To make the Li and Stephens forward algorithm for these datasets computationally tractable, we have created a numerically exact version of the algorithm with observed average case $\mathcal{O}(nk^{0.35})$ runtime in number of genetic sites n and reference panel size k. This avoids any tradeoff between runtime and model complexity. We demonstrate that our approach also provides a succinct data structure for general purpose haplotype data storage. We discuss generalizations of our algorithmic techniques to other hidden Markov models.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms, Applied computing → Bioinformatics

Keywords and phrases Haplotype, Hidden Markov Model, Forward Algorithm, Lazy Evaluation

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.9

Related Version `https://doi.org/10.1101/322396`

Supplement Material https://github.com/yoheirosen/sublinear-Li-Stephens/

Funding This work was supported by the National Human Genome Research Institute of the National Institutes of Health under Award Number 5U54HG007990, the National Heart, Lung, and Blood Institute of the National Institutes of Health under Award Number 1U01HL137183-01, and grants from the W.M. Keck foundation and the Simons Foundation.

Acknowledgements We would like to thank Jordan Eizenga for his helpful discussions throughout the development of this work.

[1] Yohei Rosen was supported in part by a Howard Hughes Medical Institute Medical Research Fellowship.

1 Introduction

Probabilistic models of haplotypes describe how variation is shared in a population. One application of these models is to calculate the probability $P(o|H)$ of a haplotype o given the assumption that it is a member of a population represented by a *reference panel* of haplotypes H. This computation has been used in estimating recombination rates [8], a problem of interest in genetics and in medicine. It may also be used to detect errors in genotype calls.

Early approaches to haplotype modeling used coalescent [7] models which were accurate but computationally complex, especially when including recombination. Li and Stephens wrote the foundational computationally tractable haplotype model [8] with recombination. Under their model, the probability $P(o|H)$ can be calculated using the forward algorithm for hidden Markov models. Generalizations of their model have been used for haplotype phasing and genotype imputation. Most of these algorithms [10, 1, 14, 3, 12] use the *forward probabilities* calculated as intermediate values in the forward algorithm.

1.1 The Li and Stephens model

Consider a *reference panel* H of k haplotypes sampled from some population. Each haplotype $h_j \in H$ is a sequence $(h_{j,1}, \ldots, h_{j,n})$ of alleles at a contiguous sequence $1, \ldots, n$ of genetic sites. Classically [8], the sites are biallelic, but the model extends to multiallelic sites. [11]

Consider an observed sequence of alleles $o = (o_1, \ldots, o_n)$ representing another haplotype. The monoploid Li and Stephens model (LS) [8] specifies a probability that o is descended from the population represented by H. LS can be written as a hidden Markov model wherein the haplotype o is assembled by copying (with possible error) consecutive contiguous subsequences of haplotypes $h_j \in H$.

▶ **Definition 1** (Li and Stephens HMM). Define $x_{j,i}$ as the event that the allele o_i at site i of the haplotype o was copied from the allele $h_{j,i}$ of haplotype $h_j \in H$. Take parameters

$$\rho^*_{i-1\to i} \quad \text{the probability of any recombination between sites } i-1 \text{ and } i \tag{1}$$

$$\mu_i \quad \text{the probability of a mutation from one allele to another at site } i \tag{2}$$

and from them define the transition and recombination probabilities

$$p(x_{j,i}|x_{j',i-1}) = \begin{cases} 1-(k-1)\rho_i & \text{if } j = j' \\ \rho_i & \text{if } j \neq j' \end{cases} \qquad \text{where } \rho_i = \frac{\rho^*_{i-1\to i}}{k-1} \tag{3}$$

$$p(o_i|x_{j,i}) = \begin{cases} 1-(A-1)\mu_i & \text{if } o_i = h_{j,i} \\ \mu_i & \text{if } o_i \neq h_{j,i} \end{cases} \qquad \text{where } A = \text{number of alleles} \tag{4}$$

We will write $\mu_i(j)$ as shorthand for $p(o_i|x_{j,i})$. We will also define the values of the initial probabilities $p(x_{j,1}, o_1|H) = \frac{\mu_1(j)}{k}$.

Let $P(o|H)$ be the probability that haplotype o was produced from population H. The forward algorithm for hidden Markov models allows calculation of this probability in $\mathcal{O}(nk^2)$ time using an $n \times k$ dynamic programming matrix of *forward states*

$$p_i[j] = P(x_{j,i}, o_1, \ldots, o_i|H) \tag{5}$$

In practice, the Li and Stephens forward algorithm is $\mathcal{O}(nk)$. (See §3)

1.1.1 Li and Stephens like algorithms for large populations

The $\mathcal{O}(nk)$ time complexity of the forward algorithm is intractable for reference panels with large size k. The UK Biobank has amassed $k = 500,000$ array samples. Whole genome sequencing projects, with a denser distribution of sites, are catching up. Major sequencing projects with $k = 100,000$ or more samples are nearing completion. Others numbering k in the millions have been announced. These large population datasets have significant potential benefits: They are statistically likely to more accurately represent population frequencies and those employing genome sequencing can provide phasing information for rare variants.

In order to handle datasets with size k even fractions of these sizes, modern haplotype inference algorithms depend on models which are simpler than the Li and Stephens model or which sample subsets of the data. For example, the common tools Eagle-2, Beagle, HAPI-UR and Shapeit-2 and -3 [10, 1, 14, 3, 12] either restrict where recombination can occur, fail to model mutation, model long-range phasing approximately or sample subsets of the reference panel.

Lunter's "fastLS" algorithm [11] demonstrated that haplotypes models which include all k reference panel haplotype could find the Viterbi maximum likelihood path in time sublinear in k, using preprocessing to reduce redundant information in the algorithm's input. However, his techniques do not extend to the forward and forward-backward algorithms.

1.2 Our contributions

We have developed an arithmetically exact forward algorithm whose expected time complexity is a function of the expected allele distribution of the reference panel. This expected time complexity proves to be $\mathcal{O}(k^{0.35})$ in reference panel size. We have also developed a technique for succinctly representing large panels of haplotypes whose size also scales as a sublinear function of the expected allele distribution.

Our forward algorithm contains three optimizations, all of which might be generalized to other bioinformatics algorithms. In (§2), we rewrite the reference panel as a sparse matrix containing the minimum information necessary to directly infer all allele values. In (§3), we define recurrence relations which are numerically equivalent to the forward algorithm but use minimal arithmetic operations. In (§4), we delay computation of forward states using a lazy evaluation algorithm which benefits from blocks of common sequence. Our methods apply to other models which share certain properties with the monoploid Li and Stephens model.

2 Sparse representation of haplotypes

The forward algorithm to calculate the probability $P(o|H)$ takes as input a length n vector o and a $k \times n$ matrix of haplotypes H. Therefore time complexity better than $\mathcal{O}(nk)$ is impossible unless there is preprocessing of its input. However, such preprocessing can be amortized over many queries o.

2.1 Information content of a reference panel

Recall that $(o_i)_{i=1}^n$ is the allele sequence of the emitted haplotype o. (§3) will show that $\phi_i(o_i), 1 \leq i \leq n$ defined below are sufficient data to calculate $P(o|H)$.

▶ **Definition 2.** The **information content** ϕ of H for allele a at site i is defined as

$$\phi_i(a) = \begin{cases} Match_i(a) & := \{h_j \mid h_{j,i} = a\} \quad \text{if } |Match_i(a)| \leq |NonMatch_i(a)| \\ NonMatch_i(a) & := \{h_j \mid h_{j,i} \neq a\} \quad \text{if } |NonMatch_i(a)| < |Match_i(a)| \end{cases} \tag{6}$$

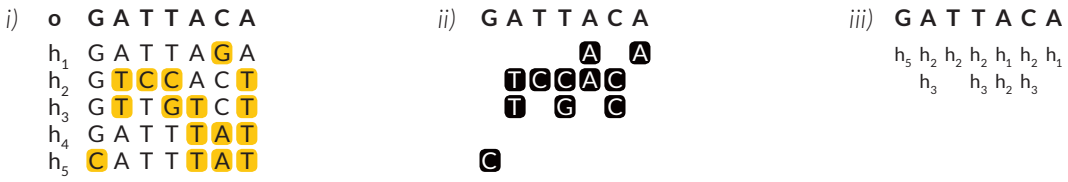


Figure 1 i) Reference panel $\{h_1, \ldots, h_5\}$ with mismatches to haplotype o shown in yellow. **ii)** Alleles at site i of elements of $\phi_i(o_i)$ in black. **iii)** Vectors to encode $\phi_i(o_i)$ at each site.

We will often abuse notation and refer to the $h_j \in \phi_i(a)$ by their indices j alone.

2.2 Relation of information content to allele frequency spectrum

Our sparse representation of the haplotype reference panel benefits from the recent finding [6] that the distribution over sites of minor allele frequencies is biased towards low frequencies[2].

We will compute the expected time sum of the information content over all sites assuming first that all sites are biallelic[3]. In the biallelic case $\phi_i(\cdot)$ is always the set of haplotypes displaying the minor allele at site i and the distribution of $\phi_i(a)$ is the allele frequency spectrum.

▶ **Lemma 3.** *Let $\mathbb{E}[\overline{f}](k)$ be the expected mean minor allele frequency for k genotypes. Then*

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left|\phi_i(a)\right|\right] = \mathbb{E}[\overline{f}](k) \tag{7}$$

▶ **Corollary 4.** *If $\mathcal{O}(\mathbb{E}[\overline{f}]) < \mathcal{O}(k)$, then $\mathcal{O}(\sum_i \left|\phi_i(a)\right|) < \mathcal{O}(nk)$ in expected value.*

2.3 Implementation

For biallelic sites, we store our ϕ_i's using a length-n vector of length $|\phi_i|$ vectors containing the indices j of the haplotypes $h_j \in \phi_i$ and a length-n vector listing the major allele at each site. (See Figure 1 panel iii) Random access by key i to iterators to the first elements of sets $\phi_i(a)$ is $\mathcal{O}(1)$ and iteration across these $\phi_i(a)$ is linear in the size of $\phi_i(a)$. For multiallelic sites, the data structure uses slightly more space but has the same speed guarantees.

Generating these data structures takes $\mathcal{O}(nk)$ time but is embarrassingly parallel in n. Our "*.slls" data structure doubles as a succinct haplotype index which could be distributed instead of a large vcf record. A vcf $\rightarrow$ slls conversion tool is found in our github repository.

Adding or rewriting a haplotype is constant time per site per haplotype unless this edit changes which allele is the most frequent. This allows our algorithm to extend to uses of the Li and Stephens model where one might want to dynamically edit the reference panel.

3 Efficient dynamic programming

We begin with the recurrence relation of the $\mathcal{O}(nk)$ Li and Stephens forward algorithm [8]. To establish our notation, recall that $p_i[j] = P(x_{j,i}, o_1, \ldots, o_i | H)$, that we write $\mu_i(j)$ as

[2] We confirm these results in section 5.2
[3] The generalization is trivial

shorthand for $p(o_i|x_{j,i})$ and that we have initialized $p_1[j] = p(x_{j,1}, o_1|H) = \frac{\mu_1(j)}{k}$. For $i > 1$, we may then write:

$$p_i[j] = \mu_i(j)\big((1 - k\rho_i)p_{i-1}[j] + \rho_i S_{i-1}\big) \tag{8}$$

$$S_i = \sum_{j=1}^{k} p_i[j] \tag{9}$$

We will reduce the number of summands in (9) and reduce the number indices j for which (8) is evaluated. This will use the **information content** defined in (§2.1).

▶ **Lemma 5.** *The summation (9) is calculable using strictly fewer than k summands.*

Proof. Suppose first that $\mu_i(j) = \mu_i$ for all j. Then

$$S_i = \sum_{j=1}^{k} p_i[j] = \mu_i \sum_{j=1}^{k} \big((1 - k\rho_i)p_{i-1}[j] + \rho_i S_{i-1}\big) \tag{10}$$

$$= \mu_i\big((1 - k\rho_i)S_{i-1} + k\rho_i S_{i-1}\big) = \mu_i S_{i-1} \tag{11}$$

Now suppose that $\mu_i(j) = 1 - \mu_i$ for some set of j. We must then correct for these j. This gives us

$$S_i = \mu_i S_{i-1} + \frac{1 - \mu_i - \mu_i}{1 - \mu_i} \sum_{j \text{ where } \mu_i(j) \neq \mu_i} p_i[j] \tag{12}$$

The same argument holds when we reverse the roles of μ_i and $1 - \mu_i$. Therefore we can choose which calculation to perform according to which involves a sum with fewer summands. This gives us the following formula:

$$S_i = \alpha S_{i-1} + \beta \sum_{j \in \phi_i(o_i)} p_i[j] \tag{13}$$

where

$$\alpha = \mu_i \qquad \beta = \frac{1 - 2\mu_i}{1 - \mu_i} \qquad \text{if } \phi_i(a) = Match_i(a) \tag{14}$$

$$\alpha = 1 - \mu_i \qquad \beta = \frac{2\mu_i - 1}{\mu_i} \qquad \text{if } \phi_i(a) = NonMatch_i(a) \tag{15}$$

◀

▶ **Lemma 6.** *If $j \notin \phi_i(o_i)$ and $j \notin \phi_{i-1}(o_{i-1})$, then S_i can be calculated without knowing $p_{i-1}[j]$ and $p_i[j]$, as can $p_i[j']$ for $j' \neq j$.*

Proof. By inspection of equation (13). ◀

▶ **Corollary 7.** *The recurrences (9) and the minimum set of recurrences (8) needed to compute (9) can be evaluated in $\mathcal{O}(|\phi|)$ time, assuming that $p_{i-1}[j]$ have been computed $\forall j \in \phi_i(o_i)$.*

We address the assumption on prior calculation of the necessary $p_{i-1}[j]$'s in section 4.

3.1 Time complexity

Recall that we defined $\mathbb{E}[\overline{f}](k)$ as the expected mean minor allele frequency in a sample of size k. By Corollary 7 the procedure in eq. (13) has expected time complexity $\mathcal{O}\left(n\mathbb{E}[\overline{f}](k)\right)$.

Figure 2 Using the example that at site i, $\phi_i(o_i) = \{h_3\}$, we illustrate the number of arithmetic operations used in **i)** the conventional $\mathcal{O}(nk)$ Li and Stephens HMM recurrence relations **ii)** Our procedure specified in equation (13). Black lines correspond to arithmetic operations; operations which cannot be parallelized over j are colored yellow.

4 Lazy evaluation of dynamic programming rows

Corollary 7 was conditioned on the assumption that specific forward probabilities had already been evaluated. We will describe a second algorithm which performs this task efficiently by avoiding arithmetic which will prove unnecessary at future steps.[4]

4.1 Eliminating redundant recurrence evaluations

The recurrence relations (8) are linear maps $r_i[j] : \mathbb{R} \to \mathbb{R}$ of the form

$$r_i[j] : x_j \longmapsto \alpha_i x_j + \beta_i \tag{16}$$

Let us formalize this notion of recurrence relation as linear map:

▶ **Definition 8.** For any $i_1 < i_2$, define the *update map* $r_{i_1 \to i_2}[j] = r_{i_2}[j] \circ r_{i_2-1}[j] \circ \cdots \circ r_{i_1+1}[j]$

This update map is defined such that $r_{i_1 \to i_2}[j](p_{i_1}[j]) = p_{i_2}[j]$.

▶ **Lemma 9.** *At each i there exist only two unique maps among the $r_i[j]$.*

Proof. Assume, without loss of generality, that $\phi_i(a) = Match_i(a)$. Then define $\mu_i^\circ = \mu_i$ and $\mu_i^\bullet = 1 - \mu_i$. Then

$$r_i^\circ(x) = \mu_i^\circ\big((1 - k\rho)x + \rho S_i\big) \tag{17}$$

$$r_i^\bullet(x) = \mu_i^\bullet\big((1 - k\rho)x + \rho S_i\big) = \frac{\mu_i^\bullet}{\mu_i^\circ} r_i^\circ \tag{18}$$

◀

If $\phi_i(a) \neq Match_i(a)$ then the same is true with the definitions of μ_i° and $\mu_i^\bullet$ switched. This lemma allows us to rewrite each $r_{i_1 \to i_2}[j]$ as a binary vector of the form $(\circ, \circ, \bullet, \ldots, \circ, \bullet, \circ)$.

We start with a sketch of the general concept of our algorithm

When Algorithm 1 is applied independently to all h_j, the aggregate algorithm has $\mathcal{O}(nk)$ time complexity, so we will share work between haplotypes j using equivalence classes segregated by runs of homology.

[4] This approach is known as *lazy evaluation*.

Algorithm 1: General approach to evaluating $p_{(\cdot)}[j]$ for a given haplotype j.

Calculate $p_1[j]$ **and** set $\ell \leftarrow 1$
for *all sites* $i > 1$ **do**
  if $r_{\ell \to i}[j] = (\circ, \circ, \ldots, \circ)$ **then** Do nothing
  if $r_{\ell \to i}[j] = (\circ, \circ, \ldots, \bullet)$ **then** Evaluate $p_i[j] = r_{\ell \to i}(p_\ell[j])$ **and** set $\ell \leftarrow i$

4.2 Equivalence classes of update map prefixes

Consider two different instances of the **for** loop in Algorithm 1, where the first is to evaluate $p_{(\cdot)}[j_1]$ and the second to evaluate $p_{(\cdot)}[j_2]$. Suppose that both are halted at the same step i, and suppose that at this step, the marker variable ℓ is the same for both of them. Then the sequences of h_{j_1} and h_{j_2} are identical between ℓ and i, and therefore

$$r_{\ell \to i}[j_1] = r_{\ell \to i}[j_2] = \underbrace{(\circ, \circ, \ldots, \circ)}_{i-\ell} \tag{19}$$

Therefore at each step i, we can divide the haplotype indices j into equivalence classes $J[\ell]$ for which

1. The current marker variable ℓ is the same for all j in this equivalence class
2. The map $r_{\ell \to i}[j]$ is the same for all j in this equivalence class

And so we only need to calculate $r_{\ell \to i}[j]$ at most once per nonempty equivalence class $J[\ell]$, and for this map we write $r_{\ell \to i}$. The following lemma allows us to be even more efficient.

▶ **Lemma 10.** *If* $i_1 < i_2 < i$, *then* $r_{i_1 \to i} = r_{i_2 \to i} \circ r_{i_1 \to i_2}$

Lemma 10 allows us to calculate intermediate prefixes of the maps $r_{\ell \to i}$ and extend them at a later time. To make this concrete, suppose that we have an index π_ℓ ("prefix") where $\ell < \pi_\ell < i$. Then we can evaluate the prefix $r_{\ell \to \pi_\ell}$ of $r_{\ell \to i}$ knowing that $r_{\ell \to i} = r_{\pi_\ell \to i} \circ r_{\ell \to \pi_\ell}$ can be evaluated at a later time.

4.3 The lazy evaluation algorithm

Our full lazy evaluation algorithm stores the following state data at each step. The algorithm initialization is described in Algorithm 2 and the recurrence in Algorithm 3.

- The maps $j \mapsto J[\ell]$ from haplotype to its equivalence class; ℓ defined as the index at which $p_{(\cdot)}[j]$ was most recently calculated
- The maps $\ell \mapsto \pi_\ell$ mapping ℓ to the index π_ℓ for which a prefix $r_{\ell \to \pi_\ell}$ of $r_{\ell \to i}$ which was most recently calculated
- The maps $\ell \mapsto r_{\ell \to \pi_\ell}$ mapping ℓ to the the prefix $r_{\ell \to \pi_\ell}$ of $r_{\ell \to i}$ which was most recently calculated
- The maps $r^\circ_1, r^\circ_2, \ldots, r^\circ_i$ from which all maps $r_{(\cdot) \to (\cdot)}$ are formed

Calculating a closed form expression for the time complexity of the lazy evaluation algorithm 3 is not straightforward. It is easy to show that it bounded by $\mathcal{O}(nk)$, since the first loop is worst-case $\mathcal{O}(k)$. However, we find experimentally, this lazy evaluation component does not contribute to overall computational complexity. (See Fig. 6)

Algorithm 2: Lazy evaluation initialization.

Input: reference panel size k **and** active rows $\phi_1(o_1)$
All $j \notin \phi_1(o_1)$ are assigned to equivalence class $J[0]$
$r_{0\to\pi_0} \leftarrow r_1^\circ$ **and** $\pi_0 \leftarrow 1$
All $j \in \phi_1(o_1)$ are assigned to equivalence class $J[1]$
$r_{1\to\pi_1} \leftarrow identity$ **and** $\pi_1 \leftarrow 1$

Algorithm 3: Lazy evaluation recurrence.

Input: active rows $\phi_i(o_i)$ **and** previous lazy evaluation state
$J_{active} \leftarrow$ subset of $\left\{J[\ell]\right\}_{\ell\le i}$ consisting of equivalence classes containing at least one index from the set of active indices $\phi_i(o_i)$
$\ell_{min} \leftarrow$ smallest ℓ such that $J[\ell] \in J_{active}$
$\sigma_{i\to i} \leftarrow r_i^\circ$
for $\lambda = i-1$ ***to*** ℓ_{min} **do**
  $\sigma_{\lambda\to i} \leftarrow \sigma_{\lambda+1\to i} \circ r_\lambda^\circ$
  For one ℓ with $\pi_\ell = \lambda$, $J_{active} \leftarrow J_{active} \cup J[\ell]$
for $J[\ell] \in J_{active}$ *with* $\pi_\ell \neq i$ **do**
  $r_{\ell\to\pi_\ell} \leftarrow \sigma_{\pi_\ell+1\to i} \circ r_{\ell\to\pi_\ell}$ **and** $\pi_\ell \leftarrow i$
for $j \in \phi_i$ **do**
  $\ell \leftarrow$ index for which $j \in J[\ell]$
  Evaluate $p_i[j] = r_{\ell\to\pi_\ell}(p_\ell[j])$ **and** reassign j to the new class $J[i]$
$r_{i\to\pi_i} \leftarrow identity$ **and** $\pi_i \leftarrow i$

5 Results

5.1 Implementation

Our algorithm was implemented as a C++ library located at `https://github.com/yoheirosen/sublinear-Li-Stephens` Details of Algorithm 3 will be found there.

We also implemented the linear time monoploid Li and Stephens forward algorithm in C++ as to evaluate it on identical footing. Profiling was performed using a single Intel Xeon X7560 core running at 2.3 GHz on a shared memory machine. Our reference panels H were the phased haplotypes from the 1000 Genomes [2] phase 3 vcf records for chromosome 22 and subsamples thereof. Haplotypes o were randomly generated simulated descendants.

5.2 Minor allele frequency distribution for the 1000 Genomes dataset

We simulated haplotypes o of $1,000,000$ bp length on chromosome 22 and recorded the sizes of the sets $\phi_i(o_i)$ for $k = 5008$. These data produced a mean $|\phi_i(o_i)|$ of 59.9, which is 1.2% of the size of k. We have plotted the distribution of $|\phi_i(o_i)|$ which we observed from this experiment in (Fig. 4). It is skewed toward low frequencies; the minor allele is unique at 71% of sites, and it is below 1% frequency at 92% of sites.

5.3 Comparison of our algorithm with the linear time forward algorithm

In order to compare the dependence of our algorithm's runtime on haplotype panel size k against that of the standard linear LS forward algorithm, we measured the CPU time per

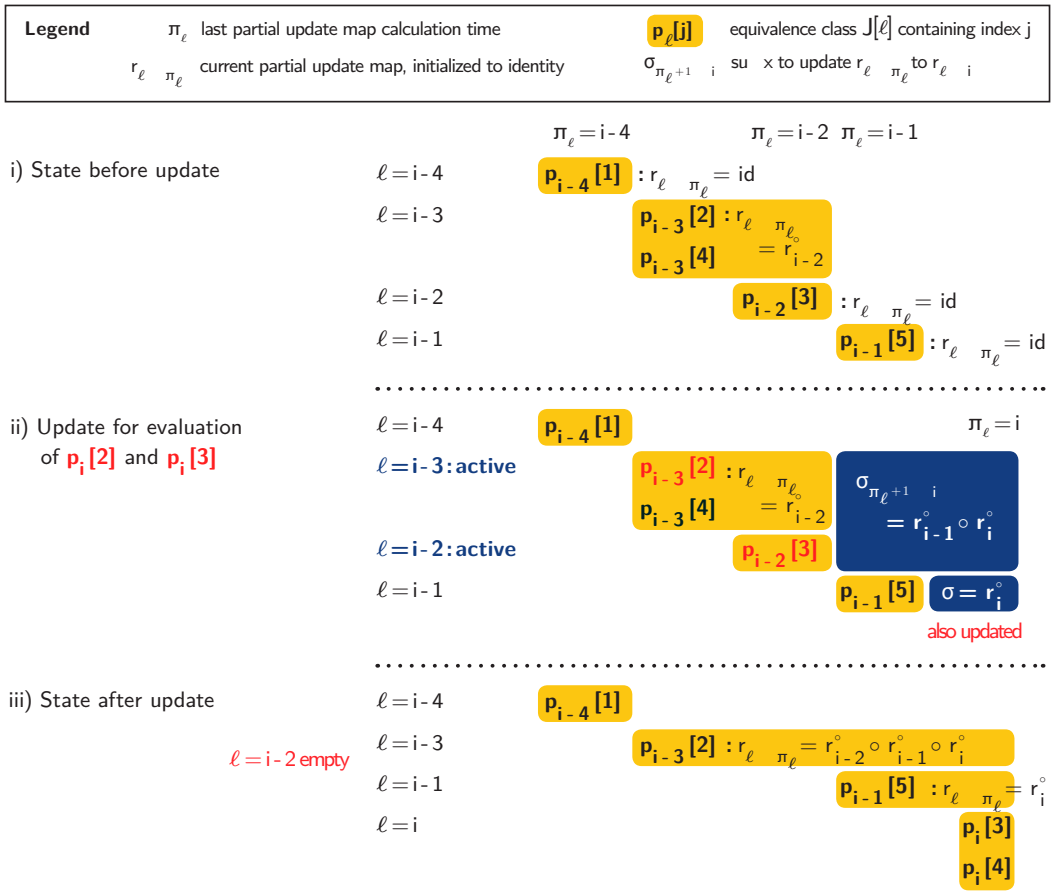


Figure 3 An illustration of the lazy evaluation states defined above as well as the steps of Algorithm 3. In this case, we have a lazy evaluation state at i = 5, which is updated with $\phi_i(o_i) = \{2, 4\}$. $j = 3$ is updated as well as specified by the first loop in the algorithm.

genetic site of both across a range of haplotype panel sizes from 30 to 5008. Figure 5 shows this comparison. Observed time complexity of our algorithm was $\mathcal{O}(k^{0.35})$ as calculated from the slope of the line of best fit to a log-log plot of time per site versus haplotype panel size.

For data points where we used all 1000 Genomes project haplotypes ($k = 5008$), on average, time per site is 37 μs for our algorithm and 1308 μs for the linear LS algorithm. For the forthcoming 100,000 Genomes Project, these numbers can be extrapolated to 251 μs for our algorithm and 260,760 μs for the linear LS algorithm.

5.3.1 Lazy evaluation of dynamic programming rows

We also measured the time which our algorithm spent within its lazy evaluation subalgorithm. In the average case, the time complexity of our lazy evaluation subalgorithm does not contribute to the overall algebraic time complexity of the algorithm. (Fig. 6, right) The lazy evaluation runtime also contributes minimally to the total actual runtime of our algorithm. (Fig. 6, left)

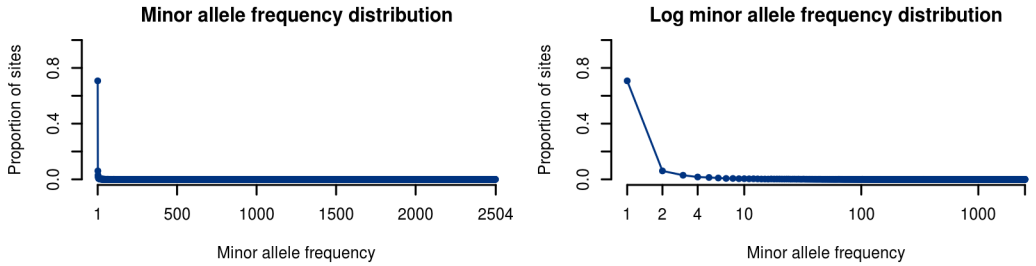


Figure 4 Biallelic site minor allele frequency distribution from 1000 Genomes chromosome 22.

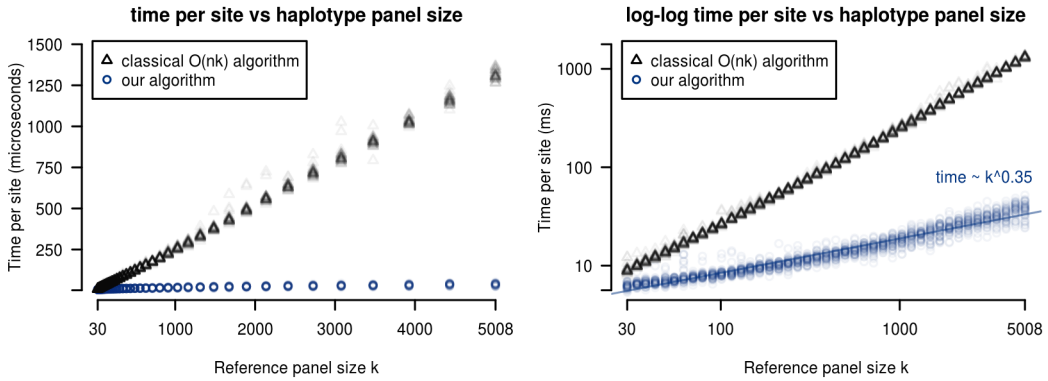


Figure 5 Runtime per site as a function of haplotype reference panel size k for our algorithm (blue) as compared to the classical linear time algorithm (black).

5.4 Sparse haplotype encoding

5.4.1 Generating our sparse vectors

We generated the haplotype panel data structures from (§2) using the vcf-encoding tool `vcf2slls` which we provide. We built indices with multiallelic sites, which increases their time and memory profile relative to the results in (§5.2) but allows direct comparison to vcf records. Encoding of chromosome 22 was completed in 38 minutes on a single CPU core. Use of M CPU cores will reduce runtime proportional to M.

5.4.2 Size of sparse haplotype index

In uncompressed form, our whole genome `*.slls` index for chromosome 22 of the 1000 genomes dataset was 285 MB in size versus 11 GB for the vcf record using `uint16_t`'s to encode haplotype ranks. When compressed with gzip, the same index was 67 MB in size versus 205 MB for the vcf record.

In the interest of speed (both for our algorithm and the $\mathcal{O}(nk)$ algorithm) our experiments loaded entire chromosome sparse matrices into memory and stored haplotype indices as `uint64_t`'s. This requires on the order of 1 GB memory for chromosome 22. For long chromosomes or larger reference panels on low memory machines, algorithm can operate on sequential chunks of the reference panel.

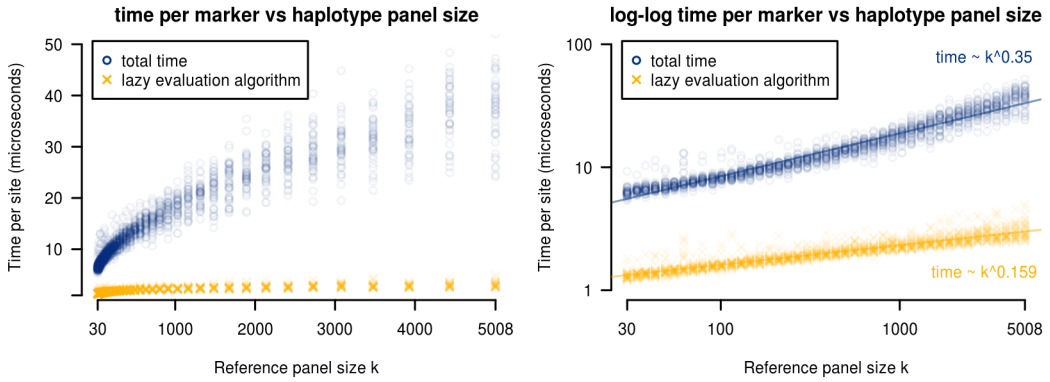


Figure 6 Time per site for the lazy evaluation subalgorithm (yellow) vs. the full algorithm (blue).

6 Discussion and significance

To the best of our knowledge, ours is the first forward algorithm for any haplotype model to attain sublinear time complexity with respect to reference panel size. Our algorithms could be incorporated into haplotype inference strategies by interfacing with our C++ library. This opens the potential for tools which are tractable on haplotype reference panels at the scale of current 100,000 to 1,000,000+ sample sequencing projects.

6.1 Applications which use individual forward probabilities

Our algorithm attains its runtime specifically for the problem of calculating the single overall probability $P(o|H, \rho, \mu)$ and does not compute all nk forward probabilities. We can prove that if m many specific forward probabilities are also required as output, and if the time complexity of our algorithm is $\mathcal{O}(\sum_i |\phi_i|)$, then the time complexity of the algorithm which also returns the m forward probabilities is $\mathcal{O}(\sum_i |\phi_i| + m)$.

In general, haplotype phasing or genotype imputation tools use stochastic traceback or other similar sampling algorithms. The standard algorithm for stochastic traceback samples states from the full posterior distribution and therefore requires all forward probabilities. The algorithm output and lower bound of its speed is therefore $\mathcal{O}(nk)$. The same is true for many applications of the forward-backward algorithm.

There are two possible approaches which might allow runtime sublinear in k for these applications. Using stochastic traceback as an example, first is to devise an $\mathcal{O}(f(m))$ sampling algorithm which uses $m = g(k)$ forward probabilities such that $\mathcal{O}(f \circ g(k)) < \mathcal{O}(k)$. The second is to succinctly represent forward probabilities such that nested sums of the nk forward probabilities can be queried from $\mathcal{O}(\phi) < \mathcal{O}(nk)$ data. This should be possible, perhaps using the positional Burrows-Wheeler transform [5] as in [11], since we have already devised a forward algorithm with this property for a different model in [13].

6.2 Generalizability of algorithm

The optimizations which we have made are not strictly specific to the monoploid Li and Stephens algorithm. Necessary conditions for our reduction in the time complexity of the recurrence relations are

▶ **Condition 1.** *The number of distinct transition probabilities is bounded.*

▶ **Condition 2.** *The number of distinct emission probabilities is bounded.*

Favourable conditions for efficient time complexity of the lazy evaluation algorithm are

▶ **Condition 1.** *The number of unique update maps added per step is bounded.*

▶ **Condition 2.** *The update map extension operation is composition of matrices of bounded size. This can be generalized to a broad algebraic class*[5] *of update operations provided that they have bounded runtime.*

The reduction in time complexity of the recurrence relations depends on the Markov property, however we hypothesize that the delayed evaluation needs only the semi-Markov property.

6.2.1 Other haplotype forward algorithms

Our optimizations are of immediate interest for other haplotype copying models. The following related algorithms have been explored without implementation.

▶ **Example** (Diploid Li and Stephens). *We have yet to implement this model but expect average runtime at least subquadratic in reference panel size k. We build on the statement of the model and its optimizations in [9]. We have found the following recurrences which we believe will work when combined with a system of lazy evaluation algorithms:*

▶ **Lemma 11.** *The diploid Li and Stephens HMM may be expressed using recurrences of the form*

$$p_i[j_1, j_2] = \alpha_p p_{i-1}[j_1, j_2] + \beta_p(S_{i-1}(j_1) + S_{i-1}(j_2)) + \gamma_p S \tag{20}$$

which use on the intermediate sums

$$S_i := \alpha_c S_{i-1} + \beta_c \sum_{j \in \phi_i} S_{i-1}(j) + \gamma_c \sum_{(j_1, j_2) \in \phi_i^2} p_{i-1}[j_1, j_2] \qquad \mathcal{O}(|\phi_i|^2) \tag{21}$$

$$S_i(j) := \alpha_c S_{i-1} + \beta_c S_{i-1}(j) + \gamma_c \sum_{j_2 \in \phi_i} p_{i-1}[j, j_2] \qquad \mathcal{O}(|\phi_i|^2) \tag{22}$$

where $\alpha_{(\cdot)}, \beta_{(\cdot)}, \gamma_{(\cdot)}$ depend only on the diploid genotype o_i.

Implementing and verifying this extension of our algorithm will be among our next steps.

▶ **Example** (Multipopulation Li and Stephens). *[4] We maintain separate sparse haplotype panel representations $\phi_i^A(o_i)$ and $\phi_i^B(o_i)$ and separate lazy evaluation mechanisms for the two populations A and B. Expected runtime guarantees are similar.*

This model, and versions for > 2 populations, will be important in large sequencing cohorts (such as NHLBI TOPMed) where assuming a single related population is unrealistic.

▶ **Example** (More detailed mutation model). *It may also be desirable to model distinct mutation probabilities for different pairs of alleles at multiallelic sites. Runtime is worse than the biallelic model but remains average case sublinear.*

▶ **Example** (Sequence graph Li and Stephens analogue). *In [13] we described a hidden Markov model for a haplotype-copying with recombination but not mutation in the context of sequence graphs. Assuming we can decompose our graph into nested sites then we can achieve a fast forward algorithm with mutation.*

▶ **Example** (Semi-Markovian recombination model). *The lazy evaluation algorithm 3 may efficiently allow time-since-recombination dependent transition probabilities.*

[5] Specifically, any collection of operations forming a *category* in the sense of category theory

References

1. Brian L Browning and Sharon R Browning. A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, 84(2):210–223, 2009.
2. 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.
3. Olivier Delaneau, Jean-Francois Zagury, and Jonathan Marchini. Improved whole-chromosome phasing for disease and population genetic studies. *Nature methods*, 10(1):5, 2013.
4. Peter Donnelly and Stephen Leslie. The coalescent and its descendants. *arXiv preprint arXiv:1006.1514*, 2010.
5. Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.
6. Alon Keinan and Andrew G Clark. Recent explosive human population growth has resulted in an excess of rare genetic variants. *science*, 336(6082):740–743, 2012.
7. John Frank Charles Kingman. The coalescent. *Stochastic processes and their applications*, 13(3):235–248, 1982.
8. Na Li and Matthew Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4):2213–2233, 2003.
9. Yun Li, Cristen J Willer, Jun Ding, Paul Scheet, and Gonçalo R Abecasis. Mach: using sequence and genotype data to estimate haplotypes and unobserved genotypes. *Genetic epidemiology*, 34(8):816–834, 2010.
10. Po-Ru Loh, Petr Danecek, Pier Francesco Palamara, Christian Fuchsberger, Yakir A Reshef, Hilary K Finucane, Sebastian Schoenherr, Lukas Forer, Shane McCarthy, Goncalo R Abecasis, et al. Reference-based phasing using the haplotype reference consortium panel. *Nature genetics*, 48(11):1443, 2016.
11. Gerton Lunter. Fast haplotype matching in very large cohorts using the li and stephens model. *bioRxiv*, 2016. doi:10.1101/048280.
12. Jared O'Connell, Kevin Sharp, Nick Shrine, Louise Wain, Ian Hall, Martin Tobin, Jean-Francois Zagury, Olivier Delaneau, and Jonathan Marchini. Haplotype estimation for biobank-scale data sets. *Nature genetics*, 48(7):817, 2016.
13. Yohei Rosen, Jordan Eizenga, and Benedict Paten. Modelling haplotypes with respect to reference cohort variation graphs. *Bioinformatics*, 33(14):i118–i123, 2017.
14. Amy L Williams, Nick Patterson, Joseph Glessner, Hakon Hakonarson, and David Reich. Phasing of many thousands of genotyped samples. *The American Journal of Human Genetics*, 91(2):238–251, 2012.

External memory BWT and LCP computation for sequence collections with applications

Lavinia Egidi[1]
University of Eastern Piedmont, Alessandria, Italy
lavinia.egidi@uniupo.it
https://orcid.org/0000-0002-9745-0942

Felipe A. Louza[2]
Department of Computing and Mathematics, University of São Paulo, Ribeirão Preto, Brazil
louza@usp.br
https://orcid.org/0000-0003-2931-1470

Giovanni Manzini[3]
University of Eastern Piedmont, Alessandria, Italy
IIT CNR, Pisa, Italy
giovanni.manzini@uniupo.it
https://orcid.org/0000-0002-5047-0196

Guilherme P. Telles[4]
Institute of Computing, University of Campinas, Campinas, Brazil
gpt@ic.unicamp.br

Abstract

We propose an external memory algorithm for the computation of the BWT and LCP array for a collection of sequences. Our algorithm takes the amount of available memory as an input parameter, and tries to make the best use of it by splitting the input collection into subcollections sufficiently small that it can compute their BWT in RAM using an optimal linear time algorithm. Next, it merges the partial BWTs in external memory and in the process it also computes the LCP values. We show that our algorithm performs $\mathcal{O}(n \,\mathsf{maxlcp})$ sequential I/Os, where n is the total length of the collection and maxlcp is the maximum LCP value. The experimental results show that our algorithm outperforms the current best algorithm for collections of sequences with different lengths and when the *average* LCP of the collection is relatively small compared to the length of the sequences.

In the second part of the paper, we show that our algorithm can be modified to output two additional arrays that, combined with the BWT and LCP arrays, provide simple, scan based, external memory algorithms for three well known problems in bioinformatics: the computation of the all pairs suffix-prefix overlaps, the computation of maximal repeats, and the construction of succinct de Bruijn graphs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Burrows-Wheeler Transform, Longest Common Prefix Array, All pairs suffix-prefix overlaps, Succinct de Bruijn graph, Maximal repeats

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.10

[1] L.E. was partially supported by the University of Eastern Piedmont project *Behavioural Types for Dependability Analysis with Bayesian Networks*
[2] F.A.L. was supported by the grant #2017/09105-0 from the São Paulo Research Foundation (FAPESP)
[3] G.M. was partially supported by PRIN grant 201534HNXC
[4] G.P.T. acknowledges the support of CNPq

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen; Article No. 10; pp. 10:1–10:14

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

Supplement Material Source code: `https://github.com/felipelouza/egap/`

Acknowledgements We wish to thank the anonymous referees for useful observations; we appreciated in particular an extensive comment on external memory models.

1 Introduction

A fundamental problem in bioinformatics is the ability to efficiently search into the billions of DNA sequences produced by NGS studies. The Burrows Wheeler transform (BWT) is a well known structure which is the starting point for the construction of compressed indices for collection of sequences [24]. The BWT is often complemented with the Longest Common Prefix (LCP) array since the latter makes it possible to efficiently emulate Suffix Tree algorithms [14, 31]. The construction of such data structures is a challenging problem. Although the final outcome is a *compressed* index, construction algorithms can be memory hungry and the necessity of developing *lightweight*, i.e. space economical, algorithms was recognized since the very beginning of the field [9, 26, 27]. When even lightweight algorithms do not fit in RAM, one has to resort to external memory construction algorithms (see [13, 18, 19, 23] and references therein).

Although the space efficient computation of the BWT in RAM is well studied, and remarkable advances have been recently obtained [2, 30], for external memory computation the situation is less satisfactory. For collections of sequences, the first external memory algorithm is the BCR algorithm described in [1] that computes the multi-string BWT for a collection of total size n, performing a number of sequential I/Os proportional to nK, where K is the length of the longest sequence in the collection. This approach is clearly not competitive when the sequences have non homogeneous lengths, and it is far from the theoretical optimal even for sequences of equal length. Nevertheless, the simplicity of the algorithm makes it very effective for collections of relatively short sequences, and this has become the reference tool for this problem. This approach was later extended [11] to compute also the LCP values with the same asymptotic number of I/Os. When computing also the LCP values, or when the input strings have different lengths, the algorithm uses $\mathcal{O}(m)$ words of RAM, where m is the number of input sequences.

In this paper, we present a new external memory algorithm for the computation of the BWT and LCP array for a collection of sequences. Our algorithm takes the amount of available RAM as an input parameter, and tries to make the best use of it by splitting the input into subcollections sufficiently small so that it can compute their BWT in internal memory using an optimal linear time algorithm. Next, it merges the partial BWTs in external memory and in the process it also computes the LCP values. Since the LCP values are computed in a non-standard order, the algorithm is completed by an external memory merge sort procedure that computes the final LCP array. We show that our algorithm performs a number of sequential I/Os between $\mathcal{O}(n\,\mathsf{avelcp})$ and $\mathcal{O}(n\,\mathsf{maxlcp})$, where avelcp and maxlcp are respectively the average and the maximum Longest Common Prefix of the input sequences. The experimental results show that our algorithm is indeed much faster than BCR for collections of sequences when the average LCP is relatively small compared to the length of the sequences.

To our knowledge, the only other known external memory algorithm for computing the BWT and LCP arrays of a collection of sequences is the one recently proposed in [4] that performs $\mathcal{O}(n\,\mathsf{maxlcp})$ sequential I/Os and uses $\mathcal{O}(m+k)$ words of RAM. We plan to experimentally compare this algorithm to ours in the near future.

Another contribution of the paper, which follows from our main result, is the design of simple external memory algorithms for three well known problems, namely: the computation of maximal repeats [21, 34], the computation of the all pairs suffix-prefix overlaps [16, 33, 35], and the construction of succinct de Bruijn graphs [3, 7, 8]. This is achieved using the BWT and LCP arrays, together with two additional arrays that our algorithm can compute without any asymptotic slowdown. The first one is the so called Document Array providing for each suffix the ID of the sequence it belongs to; the second one is a bit array indicating whether each suffix is a substring of the one immediately following it in lexicographic order. Our external memory algorithms for these problems are derived from known internal memory algorithms, but they process the input data in a single sequential scan. In addition, for the problem of the all pairs suffix-prefix, we go beyond the recent solutions [5, 33, 35] that compute *all* the overlaps, by computing only the overlaps above a certain length, still spending constant time per reported overlap. Since the above problems often involve huge datasets we believe it is important to provide external memory algorithms. To our knowledge, only for the all pair suffix-prefix problem there exists an external memory algorithm that computes all the overlaps given the BWT, LCP and Generalized Suffix Array of the input collection [5, Algorithm 2].

2 Background

Let $\mathsf{s}[1,n]$ denote a string of length n over an alphabet Σ of size σ. As usual, we assume $\mathsf{s}[n]$ is a special symbol (end-marker) not appearing elsewhere in s and lexicographically smaller than any other symbol. We write $\mathsf{s}[i,j]$ to denote the substring $\mathsf{s}[i]\mathsf{s}[i+1]\cdots\mathsf{s}[j]$. If $j \geq n$ we assume $\mathsf{s}[i,j] = \mathsf{s}[i,n]$. If $i > j$ or $i > n$ then $\mathsf{s}[i,j]$ is the empty string. Given two strings s_1 and s_2 we write $\mathsf{s}_1 \preceq \mathsf{s}_2$ ($\mathsf{s}_1 \prec \mathsf{s}_2$) to denote that s_1 is lexicographically (strictly) smaller than s_2. We denote by $\mathsf{LCP}(\mathsf{s}_1,\mathsf{s}_2)$ the length of the longest common prefix between s_1 and s_2.

The *suffix array* $\mathsf{sa}[1,n]$ associated to s is the permutation of $[1,n]$ giving the lexicographic order of s's suffixes, that is, for $i = 1,\ldots,n-1$, $\mathsf{s}[\mathsf{sa}[i],n] \prec \mathsf{s}[\mathsf{sa}[i+1],n]$.

The *longest common prefix* array $\mathsf{lcp}[1,n+1]$ is defined for $i = 2,\ldots,n$ by

$$\mathsf{lcp}[i] = \mathsf{LCP}(\mathsf{s}[\mathsf{sa}[i-1],n],\mathsf{s}[\mathsf{sa}[i],n]); \tag{1}$$

the lcp array stores the length of the longest common prefix (LCP) between lexicographically consecutive suffixes. For convenience we define $\mathsf{lcp}[1] = \mathsf{lcp}[n+1] = -1$.

Let $\mathsf{s}_1[1,n_1],\ldots,\mathsf{s}_k[1,n_k]$ be such that $\mathsf{s}_1[n_1] = \$_1,\ldots,\mathsf{s}_k[n_k] = \$_k$, where where $\$_1 < \ldots < \$_k$ are k symbols not appearing elsewhere in $\mathsf{s}_1,\ldots,\mathsf{s}_k$ and smaller than any other symbol. Let $\mathsf{sa}_{1\cdots k}[1,n]$ denote the suffix array of the concatenation $\mathsf{s}_1\cdots\mathsf{s}_k$ of total length $n = \Sigma_{h=1}^{k} n_h$. The *multi-string* BWT [11, 25] of $\mathsf{s}_1,\ldots,\mathsf{s}_k$, denoted by $\mathsf{bwt}_{1\cdots k}[1,n]$, is defined as

$$\mathsf{bwt}_{1\cdots k}[i] = \begin{cases} \mathsf{s}_j[\mathsf{n}_j] & \text{if } \mathsf{sa}_{1\cdots k}[i] = \Sigma_{h=1}^{j-1} n_h + 1 \\ \mathsf{s}_j[\mathsf{sa}_{1\cdots k}[i] - \Sigma_{h=1}^{j-1} n_h - 1] & \text{if } \Sigma_{h=1}^{j-1} n_h + 1 < \mathsf{sa}_{1\cdots k}[i] \leq \Sigma_{h=1}^{j} n_h. \end{cases} \tag{2}$$

Essentially $\mathsf{bwt}_{1\cdots k}$ is a permutation of the symbols in $\mathsf{s}_1,\ldots,\mathsf{s}_k$ such that the position in $\mathsf{bwt}_{1\cdots k}$ of $\mathsf{s}_i[j]$ is given by the lexicographic rank of its context $\mathsf{s}_i[j+1,n_i]$ (or $\mathsf{s}_i[1,n_i]$ if $j = n_i$). Fig. 1 shows an example with $k = 2$. Notice that for $k = 1$, this is the usual Burrows-Wheeler transform [10].

Given the suffix array $\mathsf{sa}_{1\cdots k}[1,n]$ of the concatenation $\mathsf{s}_1\cdots\mathsf{s}_k$, we consider the corresponding LCP array $\mathsf{lcp}_{1\cdots k}[1,n]$ defined as in (1) (see again Fig. 1). Note that, for $i = 2,\ldots,n$, $\mathsf{lcp}_{1\cdots k}[i]$ gives the length of the longest common prefix between the contexts of $\mathsf{bwt}_{1\cdots k}[i]$

lcp	bwt	*context*
-1	b	$\$_1$
0	c	ab$\$_1$
2	$\$_1$	abcab$\$_1$
0	a	b$\$_1$
1	a	bcab$\$_1$
0	b	cab$\$_1$
-1		

lcp	bwt	*context*
-1	c	$\$_2$
0	$\$_2$	aabcabc$\$_2$
1	c	abc$\$_2$
3	a	abcabc$\$_2$
0	a	bc$\$_2$
2	a	bcabc$\$_2$
0	b	c$\$_2$
1	b	cabc$\$_2$
-1		

id	lcp_{12}	bwt_{12}	*context*
1	-1	b	$\$_1$
2	0	c	$\$_2$
2	0	$\$_2$	aabcabc$\$_2$
1	1	c	ab$\$_1$
2	2	c	abc$\$_2$
1	3	$\$_1$	abcab$\$_1$
2	5	a	abcabc$\$_2$
1	0	a	b$\$_1$
2	1	a	bc$\$_2$
1	2	a	bcab$\$_1$
2	4	a	bcabc$\$_2$
2	0	b	c$\$_2$
1	1	b	cab$\$_1$
2	3	b	cabc$\$_2$
	-1		

Figure 1 LCP array and BWT for $\mathsf{s}_1 = \mathsf{abcab}\$_1$ and $\mathsf{s}_2 = \mathsf{aabcabc}\$_2$, and multi-string BWT and corresponding LCP array for the same strings. Column id shows, for each entry of $\mathsf{bwt}_{12} = \mathsf{bc}\$_2\mathsf{cc}\$_1\mathsf{aaaabbb}$ whether it comes from s_1 or s_2.

and $\mathsf{bwt}_{1\cdots k}[i-1]$. We stress that all practical implementations use a single \$ symbol as end-marker for all strings to avoid alphabet explosion, but end-markers from different strings are then sorted as described, i.e., on the basis of the index of the strings they belong to.

2.1 Computing multi-string BWTs

The gSACA-K algorithm [22], based on algorithm SACA-K [32], computes the suffix array for a string collection. Given a collection of strings of total length n, gSACA-K computes the suffix array for their concatenation in $O(n)$ time using $(\sigma+1)\log n$ additional bits (in practice, only 2KB are used for ASCII alphabets). It is optimal for alphabets of constant size $\sigma = O(1)$. The *multi-string* $\mathsf{bwt}_{1\cdots k}$ of $\mathsf{s}_1, \ldots, \mathsf{s}_k$ can be easily obtained from the suffix array as in (2). gSACA-K can compute also the lcp array $\mathsf{lcp}_{1\cdots k}$ still in linear time using only the additional space for the lcp values.

2.2 Merging multi-string BWTs

The Gap algorithm [12], based on an earlier algorithm by Holt and McMillan [17], is a simple procedure to merge multi-string BWTs. In its original formulation the Gap algorithm can also merge LCP arrays, but in this paper we compute LCP values using a different approach more suitable for external memory execution. We describe here only the main idea behind Gap and refer the reader to [12] for further details.

Given k multi-string BWTs for disjoint subcollections, the Gap algorithm computes a multi-string BWT for the whole collection. The computation does not explicitly need the collection but only the multi-string BWTs to be merged. For simplicity in the following we assume we are merging k single-string BWTs $\mathsf{bwt}_1 = \mathsf{bwt}(\mathsf{s}_1), \ldots, \mathsf{bwt}_k = \mathsf{bwt}(\mathsf{s}_k)$; the algorithm does not change in the general case where the inputs are multi-string BWTs. Recall that computing $\mathsf{bwt}_{1\cdots k}$ amounts to sorting the symbols of $\mathsf{bwt}_1, \ldots, \mathsf{bwt}_k$ according to the lexicographic order of their contexts, where the context of symbol $\mathsf{bwt}_j[i]$ is $\mathsf{s}_j[\mathsf{sa}_j[i], n_j]$, for $j = 1, \ldots, k$.

The Gap algorithm works in successive iterations. After the h-th iteration the entries of each bwt_λ are sorted on the basis of the first h symbols of their context. More formally, the output of the h-th iteration is a k-valued vector $Z^{(h)}$ containing $n_\lambda = |\mathsf{s}_\lambda|$ entries λ for each $\lambda = 1, \ldots, k$, such that the following property holds.

▶ **Property 1.** *For $\lambda_1, \lambda_2 \in \{1, \ldots, k\}$, $\lambda_1 < \lambda_2$, and $i = 1, \ldots, n_{\lambda_1}$ and $j = 1, \ldots, n_{\lambda_2}$ the i-th λ_1 precedes the j-th λ_2 in $Z^{(h)}$ iff $\mathsf{s}_{\lambda_1}[\mathsf{sa}_{\lambda_1}[i], \mathsf{sa}_{\lambda_1}[i]+h-1] \preceq \mathsf{s}_{\lambda_2}[\mathsf{sa}_{\lambda_2}[j], \mathsf{sa}_{\lambda_2}[j]+h-1]$.* ◀

Following Property 1 we identify the i-th λ in $Z^{(h)}$ with $\mathsf{bwt}_\lambda[i]$ so that $Z^{(h)}$ corresponds to a permutation of $\mathsf{bwt}_{1\ldots k}$. Property 1 is equivalent to state that we can logically partition $Z^{(h)}$ into $b(h)+1$ blocks

$$Z^{(h)}[1, \ell_1],\ Z^{(h)}[\ell_1+1, \ell_2],\ \ldots,\ Z^{(h)}[\ell_{b(h)}+1, n] \tag{3}$$

such that each block is either a singleton or corresponds to the set of $\mathsf{bwt}_{1\ldots k}$ symbols whose contexts are prefixed by the same length-h string. Within each block, for $\lambda_1 < \lambda_2$, the symbols of bwt_{λ_1} precede those of bwt_{λ_2} and the context of any symbol in block $Z^{(h)}[\ell_j+1, \ell_{j+1}]$ is lexicographically smaller than the context of any symbol in block $Z^{(h)}[\ell_k+1, \ell_{k+1}]$ with $k > j$. We keep explicit track of such blocks using a bit array $B[1, n+1]$ such that at the end of iteration h it is $B[i] \neq 0$ if and only if a block of $Z^{(h)}$ starts at position i, i.e. $\mathsf{lcp}_{1\ldots k}[i] \leq h-1$. By Property 1, when all entries in B are nonzero, $Z^{(h)}$ describes how the bwt_j $(j = 1, \ldots, k)$ should be merged to get $\mathsf{bwt}_{1\ldots k}$.

3 The eGap algorithm

At a glance, the eGap algorithm for computing the multi-string BWT and LCP array in external memory works in three phases. First it builds multi-string BWTs for sub-collections in internal memory, then it merges these BWTs in external memory and generates the LCP values. Finally, it merges the LCP values in external memory.

3.1 Phase 1: BWT computation

Given a collection of sequences $\mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_k$, we split it into sub-collections sufficiently small that we can compute the multi-string SA for each one of them using the linear time internal memory gSACA-K algorithm (Section 2). After computing each SA, Phase 1 writes each multi-string BWT to disk in uncompressed form using one byte per character.

3.2 Phase 2: BWT merging and LCP computation

This part of the algorithm is based on the Gap algorithm described in Section 2 but it is designed to work efficiently in external memory and it computes LCP values in addition to merging the input (multi-string) BWTs. In the following we assume that the input consists of k BWTs $\mathsf{bwt}_1, \ldots, \mathsf{bwt}_k$ of total length n over an alphabet of size σ. The input BWTs are read from disk and never moved to internal memory. We denote by $\mathsf{bwt}_{1\ldots k}$ and $\mathsf{lcp}_{1\ldots k}$ the output BWT and LCP arrays.

The algorithm initially sets $Z^{(0)} = \mathbf{1}^{n_1}\mathbf{2}^{n_2}\ldots\mathbf{k}^{n_k}$ and $B = \mathbf{10}^{n-1}\mathbf{1}$. Since the context of every symbol is prefixed by the same length-0 string (the empty string), initially there is a single block containing all symbols. At iteration h the algorithm computes $Z^{(h)}$ from $Z^{(h-1)}$ as follows. We define an array $F[1, \sigma]$ such that $F[c]$ contains the number of occurrences of characters smaller than c in $\mathsf{bwt}_{1\ldots k}$. F partitions $Z^{(h)}$ into σ buckets, one for each symbol. Using $Z^{(h-1)}$ we scan the partially merged BWT, and whenever we encounter the BWT

character c coming from bwt_i, with $i \in \{1, \ldots, k\}$, we store it in the next free position of bucket c in $Z^{(h)}$; note that c is not actually moved, instead we write i in its corresponding position in $Z^{(h)}$. Instead of using distinct arrays $Z^{(0)}, Z^{(1)}, \ldots$ we only use two arrays Z^{old} and Z^{new}, that are kept on disk. At the beginning of iteration h it is $Z^{\mathsf{old}} = Z^{(h-1)}$ and $Z^{\mathsf{new}} = Z^{(h-2)}$; at the end $Z^{\mathsf{new}} = Z^{(h)}$ and the roles of the two files are swapped. While Z^{old} is accessed sequentially, Z^{new} is updated sequentially within each bucket, that is within each set of positions corresponding to a given character. Since the boundary of each bucket is known in advance we logically split the Z^{new} file in buckets and write to each one sequentially.

The key to the computation of the LCP array by eGap is to exploit the bitvector B used by Gap to mark the beginning of blocks. We observe that entry $B[i]$ is set to $\mathbf{1}$ during iteration $h = \mathsf{lcp}_{1\cdots k}[i] + 1$, when it is determined that the contexts of $\mathsf{bwt}_{1\cdots k}[i]$ and $\mathsf{bwt}_{1\cdots k}[i-1]$ have a common prefix of length exactly $h-1$ (and a new block is created). We introduce an additional bit array B_x such that, at the beginning of iteration h, $B_x[i] = \mathbf{1}$ iff $B[i]$ has been set to $\mathbf{1}$ at iteration $h-2$ or earlier. During iteration h, if $B[i] = \mathbf{1}$ we look at $B_x[i]$. If $B_x[i] = \mathbf{0}$ then $B[i]$ has been set at iteration $h-1$: thus we output to a temporary file F_{h-2} the pair $\langle i, h-2\rangle$ to record that $\mathsf{lcp}_{1\cdots k}[i] = h-2$, then we set $B_x[i] = \mathbf{1}$ so no pair for position i will be produced in the following iterations. At the end of iteration h, file F_{h-2} contains all pairs $\langle i, \mathsf{lcp}_{1\cdots k}[i]\rangle$ with $\mathsf{lcp}[i] = h-2$; the pairs are written in increasing order of their first component, since B and B_x are scanned sequentially. These temporary files will be merged in Phase 3.

As proven in [12, Lemma 7], if at iteration h of the Gap algorithm we set $B[i] = \mathbf{1}$, then at any iteration $g \geq h+2$ processing the entry $Z^{(g)}[i]$ will not change the arrays $Z^{(g+1)}$ and B. Since the roles of the Z^{old} and Z^{new} files are swapped at each iteration, and at iteration h we scan $Z^{\mathsf{old}} = Z^{(h-1)}$ to update Z^{new} from $Z^{(h-2)}$ to $Z^{(h)}$, we can compute only the entries $Z^{(h)}[j]$ that are different from $Z^{(h-2)}[j]$. In particular, any range $[\ell, m]$ such that $B_x[\ell] = B_x[\ell+1] = \cdots = B_x[m] = \mathbf{1}$ can be added to a set of *irrelevant* ranges that the algorithm may skip in successive iterations (irrelevant ranges are defined in terms of the array B_x as opposed to the array B, since before skipping an irrelevant range we need to update both Z^{old} and Z^{new}). We read from one file the ranges to be skipped at the current iteration and simultaneously write to another file the ranges to be skipped at the next iteration (note that irrelevant ranges are created and consumed sequentially). Since skipping a single irrelevant range takes $\mathcal{O}(k+\sigma)$ time, an irrelevant range is stored only if its size is larger than a given threshold t and we merge consecutive irrelevant ranges whenever possible. In our experiments we used $t = \max(256, k+\sigma)$. In the worst case the space for storing irrelevant ranges could be $\mathcal{O}(n)$ but in actual experiments it was always less than $0.1n$ bytes.

As in the Gap algorithm, when all entries in B are nonzero, Z^{old} describes how the BWTs bwt_j $(j = 1, \ldots, k)$ should be merged to get $\mathsf{bwt}_{1\cdots k}$, and a final sequential scan of the input BWTs along with Z^{old} allows to write $\mathsf{bwt}_{1\cdots k}$ to disk, in sequential order. Our implementation can merge at most $2^7 = 128$ BWTs at a time because we use 7 bits to store each entry of Z^{old} and Z^{new}. These arrays are maintained on disk in two separate files; the additional bit of each byte are used to keep the current and the next copy of B. The bit array B_x is stored separately in a file of size $n/8$ bytes. To merge of a set of $k > 128$ BWTs we split the input in subsets of cardinality 128 and merge them in successive rounds. We have also implemented a semi-external version of the merge algorithm that uses n bytes of RAM. The i-th byte is used to store $Z^{\mathsf{old}}[i]$ and $Z^{\mathsf{new}}[i]$ (3 bits each), $B[i]$ and $B_x[i]$.

3.3 Phase 3: LCP merging

At the end of Phase 2 all LCP-values have been written to the temporary files F_h on disk as pairs $\langle i, \mathsf{lcp}[i]\rangle$. Each file F_h contains all pairs with second component equal to h in order of increasing first component. The computation of the LCP array is completed using a standard external memory multiway merge [20, ch. 5.4.1] of maxlcp sorted files, where $\mathsf{maxlcp} = \max_i(\mathsf{lcp}_{1\ldots k}[i])$ is the largest LCP value.

3.4 Analysis

During Phase 1, gSACA-K computes the suffix array for a sub-collection of total length m using $9m$ bytes. If the available RAM is M, the input is split into subcollections of size $\approx M/9$. Since gSACA-K runs in linear time, if the input collection has total size n, Phase 1 takes $\mathcal{O}(n)$ time overall.

A single iteration of Phase 2 consists of a complete scan of $Z^{(h-1)}$ except for the irrelevant ranges. Since the algorithm requires maxlcp iterations, without skipping the irrelevant ranges the algorithm would require maxlcp sequential scans of $\mathcal{O}(n)$ items. Reasoning as in [12, Theorem 8] we get that by skipping irrelevant ranges the overall amount of data *directly* read/written by the algorithm is $\mathcal{O}(n\,\mathsf{avelcp})$ items where avelcp is the arithmetic average of the entries in the final LCP array. However, if we reason in terms of disk blocks, every time we skip an irrelevant range we discard the current block and load a new one (unless the beginning of the new relevant range is inside the same block, in that case skipping the irrelevant range does not save any I/O). We can upper bound this extra cost, with an overhead of $\mathcal{O}(1)$ blocks for each irrelevant range skipped. Summing up, assuming the total number of skipped ranges is Ir and that each disk block consists of S words, the I/O complexity of Phase 2 is $\mathcal{O}(Ir + n\,\mathsf{avelcp}\log k/(S\log n))$ block I/Os, where k is the number of input BWTs. Although the preliminary experiments in Section 4 suggest that in practice Ir is small, for simplicity and uniformity with the previous literature we upper bound the cost of Phase 2 with $\mathcal{O}(n\,\mathsf{maxlcp})$ sequential I/Os, (corresponding to $\mathcal{O}(n\,\mathsf{maxlcp}\log k/(S\log n))$ block I/Os). As a future work, we plan to do a detailed theoretical and experimental analysis of the impact of skipping irrelevant ranges.

Phase 3 takes $\mathcal{O}(\lceil \log_K \mathsf{maxlcp}\rceil)$ rounds; each round merges K LCP files by sequentially reading and writing $\mathcal{O}(n)$ bytes of data. The overall cost of Phase 3 is therefore $\mathcal{O}(n\log_K \mathsf{maxlcp})$ sequential I/Os. In our experiments we used $K = 256$; since in our tests $\mathsf{maxlcp} < 2^{16}$ two merging rounds were always sufficient.

The above analysis suggests that Phase 2 is the most expensive phase of the eGap algorithm. Indeed, in our experiments we found that Phase 2 always took at least 95% of the overall running time.

4 Experiments

In this section we report some preliminary experiments on the eGap algorithm. Testing external memory algorithms is extremely time consuming since, to make a realistic external memory setting, one has to use an amount of RAM smaller than the size of the data. If more RAM is available, even if the algorithm is supposedly not using it, the operating system will use it to temporary store disk data and the algorithm will be no longer really working in external memory. This phenomenon will be apparent also from our experiments. For this reasons we used datasets of size 8GB, reported in Table 1, and a machine with 32GB of RAM but reduced at boot time to 1GB, to simulate input data much larger than the available

Table 1 Datasets used in our experiments. shortreads are DNA reads from human genome[5] trimmed to length 100. longreads are Illumina HiSeq 4000 paired-end RNA-seq reads from plant Setaria viridis[6] trimmed to length 300. pacbio are PacBio RS II reads from Triticum aestivum (wheat) genome[7] with different lengths. pacbio.1000 are the strings from pacbio trimmed to length 1,000. Columns 5 and 6 show the maximum and average lengths of the single strings. Columns 7 and 8 show the maximum and average LCPs of the collections.

Name	Size GB	σ	N. of strings	Max Len	Ave Len	Max LCP	Ave LCP
shortreads	8.0	6	85,899,345	100	100	99	27.90
longreads	8.0	5	28,633,115	300	300	299	90.28
pacbio.1000	8.0	5	8,589,934	1,000	1,000	876	18.05
pacbio	8.0	5	942,248	71,561	9,116	3,084	18.32

RAM, and 8GB, to simulate input data of approximately the same size as the available RAM and test also the semi external version of our algorithm. Note that for a 8GB input, the output BWT+LCP data has size 24GB, so even 8GB RAM is still significantly less than the input and the output combined.

We implemented eGap in ANSI C based on the source code of Gap [12] and gSACA-K [22]. Our algorithm was compiled with GNU GCC ver. 4.6.3, with optimizing option -O3. The source code is freely available at `https://github.com/felipelouza/egap/`. The experiments were conducted on a machine with GNU/Linux Debian 7.0/64 bits operating system using an Intel i7-3770 3.4 GHz processor with 8 MB cache, 32 GB of RAM and a 2.0 TB SATA hard disk with 7200 RPM and 64 MB cache.

We compared eGap with BCR+LCP[8] from the BEETL Library [1] which is the currently most used tool for the construction of BWT and LCP arrays in external memory. As a reference we also tested the external memory tool eGSA [23] that computes the Suffix and LCP arrays for a collection of sequences. However, we tested eGSA only using 32GB of RAM since the authors in [23] showed its running time degrades about 25 times when the RAM is restricted to the input size.

The results in Table 2 show that eGap's running time per input byte is roughly proportional to the average LCP. For example, if we look at the two pacbio datasets we see that they have widely different maximum LCPs, yet their running times are very close similarly to their average LCPs. According to the theoretical analysis in [1], BCR+LCP running time per input byte is proportional to the sequence length. Using 1GB RAM the tool BCR+LCP could not handle the shortreads dataset because of insufficient internal memory, and it stopped with an internal error after four days of computation on the pacbio.1000 dataset (we have contacted the authors about the error and plan to complete the comparison as soon as we obtain a stable version of the software). The tool BCR+LCP cannot handle input sequences of different lengths, so for the pacbio dataset we used the tool extLCP [11] by the same authors. However, extLCP appears to be not competitive on pacbio and for each instance we stopped it when it became clear that its running time was much higher than eGap's. In the future we also plan to include in the comparison two recent algorithms proposed in [4, 6]. In particular, the algorithm in [4] is also based on a merge strategy. First, for each h, it computes the sequence

[5] `ftp://ftp.sra.ebi.ac.uk/vol1/ERA015/ERA015743/srf/`
[6] `https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=ERR1942989`
[7] `https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR5816161`
[8] `https://github.com/BEETL/BEETL/`

Table 2 Running times in μ seconds per input byte.

Name	eGap			BCR+LCP			eGSA
	1GB	8GB	32GB	1GB	8GB	32GB	32GB
shortreads	17.19	3.76	2.87	×	5.65	3.96	2.08
longreads	52.39	9.75	6.76	18.54	16.01	10.88	1.89
pacbio.1000	24.88	3.54	1.81	×	54.00	36.96	1.89
pacbio	23.43	3.42	1.82	> 70	> 50	> 50	1.74

of characters preceding all length-h suffixes, ordered according to the lexicographical order of the length-h suffixes (a sort of partial BWT). Then, it uses a procedure inspired by the Holt-Macmillan algorithm [17] to merge all the partial BWTs to the final output.

Although incomplete, the results show that BCR+LCP is competitive for short reads or collections with a large average LCP, while eGap clearly dominates in datasets with long reads and relatively small average LCP. In particular, when the available RAM is at least equal to the size of the input, eGap can use the semi-external strategy described in Section 3 and becomes significantly faster. Note that using 32GB RAM both algorithms become much faster: even though they allocate for their use a small fraction of that RAM, the operating system uses the remaining RAM as a buffer and avoids many disk accesses. Using 32GB RAM eGSA turns out to be the fastest algorithm and its running time appears to be less influenced by the size of the average LCP. Another advantage is that it also computes the Suffix Array, but it has the drawback of using a large amount of disk working space: 340GB for a 8GB input vs 56GB used by eGap.

5 Applications

In this section we show that the eGap algorithm, in addition to the BWT and LCP arrays, can output additional information that can be used to design efficient *external memory* algorithms for three well known problems on sequence collections: the computation of maximal repeats, the all pairs suffix-prefix overlaps, and the construction of succinct de Bruijn graphs. For these problems we describe algorithms which are derived from known (internal memory) algorithms but they process the input data in a single sequential scan. In addition, the amount of RAM used by the algorithms is usually much smaller than the size of inputs since it grows linearly with the number of sequences and the maximum LCP value.

Our starting point is the observation that the eGap algorithm can also output an array which provides, for each bwt entry, the id of the sequence to which that entry belongs. In information retrieval this is usually called the Document Array, so in the following we will denote it by da. In Phase 1 the gSACA-K algorithm can compute the da together with the lcp and bwt using only additional $4n$ bytes of space to store the da entries. These partial da's can be merged in Phase 2 using the Z^{new} array in the same way as the BWT entries. In the following we use bwt, lcp, and da to denote the multistring BWT, LCP and Document Array of a collection of m sequences of total length n. We write s to denote the concatenation $\mathsf{s}_1 \cdots \mathsf{s}_m$ and sa to denote the suffix array of s. We will use s and sa to prove the correctness of our algorithms, but neither s nor sa are used in the computations.

5.1 Computation of Maximal Repeats

Different notions of maximal repeats have been used in the bioinformatic literature to model different notions of repetitive structure. We use a notion of maximal repeat from [15, Ch. 7]: we say that a string α is a *Type 1 maximal repeat* if α occurs in the collection at least twice

and every extension, i.e. $c\alpha$ or αc with $c \in \Sigma$, occurs fewer times. We consider also a more restrictive notion: we say that a string α is a *Type 2 maximal repeat* if α occurs in the collection at least twice and every extension of α occurs at most once.

We first show how to compute Type 1 maximal repeats with a sequential scan of the arrays bwt and lcp. The crucial observation is that we have a substring of length ℓ that prefixes sa entries $j, j+1, \ldots, i$ iff $\mathsf{lcp}[h] \geq \ell$ for $h = j+1, \ldots, i$, and both $\mathsf{lcp}[j]$ and $\mathsf{lcp}[i+1]$ are smaller than ℓ. To ensure that the repeat is maximal, we also require that there exists $h \in [j+1, i]$ such that $\mathsf{lcp}[h] = \ell$ and that $\mathsf{bwt}[j, i]$ contains at least two distinct characters.

During the scan, we maintain a stack containing pairs $\langle j, \mathsf{lcp}[h]\rangle$ such that $j \leq h$; in addition if$\langle j', \mathsf{lcp}[h']\rangle$ is below $\langle j, \mathsf{lcp}[h]\rangle$ on the stack, then $h' < j$. If, when we reach position $i \geq h$, the pair $\langle j, \mathsf{lcp}[h]\rangle$ is in the stack this means that all positions k between j and i we have $\mathsf{lcp}[k] \geq \mathsf{lcp}[h]$. To maintain this invariant, when we reach position i, if the current top pair $\langle j, \mathsf{lcp}[h]\rangle$ has $\mathsf{lcp}[h] < \mathsf{lcp}[i]$, then $\langle i, \mathsf{lcp}[i]\rangle$ is pushed on top of the stack. Otherwise, all pairs $\langle j, \mathsf{lcp}[h]\rangle$ with $\mathsf{lcp}[h] \geq \mathsf{lcp}[i]$ are popped from the stack; if $\hat{\jmath}$ is the index of the last pair popped from the stack, pair $\langle \hat{\jmath}, \mathsf{lcp}[i]\rangle$ is pushed on the stack. The rationale for the latter addition is that for all $\hat{\jmath} \leq j \leq i$ it is $\mathsf{lcp}[j] \geq \mathsf{lcp}[i]$ and therefore the prefix of length $\mathsf{lcp}[i]$ of $\mathsf{s}[\mathsf{sa}[j], n]$ is the same as the prefix of the same length of $\mathsf{s}[\mathsf{sa}[i], n]$. It is not difficult to prove that for each stack entry $\langle j, \mathsf{lcp}[h]\rangle$, it is $\mathsf{lcp}[j-1] < \mathsf{lcp}[h]$.

If entry $\langle j+1, \mathsf{lcp}[h]\rangle$ is removed from the stack at iteration $i+1$, by the above discussion $\mathsf{lcp}[j] < \mathsf{lcp}[h]$; $\mathsf{lcp}[i+1] < \mathsf{lcp}[h]$ (because $\langle j+1, \mathsf{lcp}[h]\rangle$ is being removed), and for $k = j+1, \ldots, i$ $\mathsf{lcp}[k] \geq \mathsf{lcp}[h]$. To ensure that we have found a Type 1 maximal repeat we only need to check that $\mathsf{bwt}[j, i]$ contains at least two distinct characters. To efficiently check this latter condition, for each stack entry $\langle j, \mathsf{lcp}[h]\rangle$ we maintain a bitvector b_j of size σ keeping track of the distinct characters in the array bwt from position $j-1$ to the next stack entry, or to the last seen position for the entry at the top of the stack. When $\langle j, \mathsf{lcp}[h]\rangle$ is popped from the stack its bitvector is or-ed to the previous stack entry in constant time; if $\langle j, \mathsf{lcp}[h]\rangle$ is popped from the stack and immediately replaced with $\langle j, \mathsf{lcp}[i]\rangle$ its bitvector survives as it is (essentially because it is associated with an index, not with a stack entry). Clearly, maintaining the bitvector does not increase the asymptotic cost of the algorithm.

To find Type 2 maximal repeats, we are interested in consecutive LCP entries $\mathsf{lcp}[j], \mathsf{lcp}[j+1], \ldots, \mathsf{lcp}[i], \mathsf{lcp}[i+1]$, such that $\mathsf{lcp}[j] < \mathsf{lcp}[j+1] = \mathsf{lcp}[j+2] = \cdots = \mathsf{lcp}[i] > \mathsf{lcp}[i+1]$. Indeed, this ensures that for $h = j, \ldots, i$ all suffixes $\mathsf{s}[\mathsf{sa}[h], n]$ are prefixed by the same string α of length $\mathsf{lcp}[j+1]$ and every extension αc occurs at most once. If this is the case, then α is a Type 2 maximal repeat if all characters in $\mathsf{bwt}[j, i]$ are distinct since this ensures that also every extension $c\alpha$ occurs at most once. In order to detect this situation, as we scan the lcp array we maintain a candidate pair $\langle j+1, \mathsf{lcp}[j+1]\rangle$ such that $j+1$ is the largest index seen so far for which $\mathsf{lcp}[j] < \mathsf{lcp}[j+1]$. When we establish a candidate at $j+1$ as above, we init a bitvector b marking entries $\mathsf{bwt}[j]$ and $\mathsf{bwt}[j+1]$. As long as the following values $\mathsf{lcp}[j+2], \mathsf{lcp}[j+3], \ldots$ are equal to $\mathsf{lcp}[j+1]$ we go on updating b and if the same position is marked twice we discard $\langle j+1, \mathsf{lcp}[j+1]\rangle$. If we reach an index $i+1$ such that $\mathsf{lcp}[i+1] > \mathsf{lcp}[j+1]$, we update the candidate and reinitialize b. If we reach $i+1$ such that $\mathsf{lcp}[i+1] < \mathsf{lcp}[j+1]$ and $\langle j+1, \mathsf{lcp}[j+1]\rangle$ has not been discarded, then a repeat of Type 2 (with $i-j+1$ repetitions) has been located.

Note that when our algorithms discover Type 1 or Type 2 maximal repeats, we know the repeat length and the number of occurrences, so one can easily filter out non-interesting repeats (too short or too frequent). In some applications, for example the MUMmer tool [28], one is interested in repeats that occur in at least r distinct sequences, maybe exactly once for each sequence. Since for these applications the number of distinct sequences is relatively

small, we can handle this requirements by simply scanning the da array simultaneously with the lcp and bwt arrays and keeping track of the sequences associated to a maximal repeat using a bitvector (or a union-find structure) as we do with characters in the bwt.

5.2 All pairs suffix-prefix overlaps

In this problem we want to compute, for each pair of sequences s_i s_j, the longest overlap between a suffix of s_i and a prefix of s_j. Our solution follows closely the one in [33] which in turn was inspired by an earlier Suffix-tree based algorithm [16]. The algorithm in [33] solves the problem using a Generalized Enhanced Suffix array (consisting of the arrays sa, lcp, and da) in $\mathcal{O}(n + m^2)$ time, which is optimal since there are m^2 overlaps. However, for large collections it is natural to consider the problem of reporting only the overlaps larger than a given threshold τ still spending constant time per reported overlap. Our algorithm solves this more challenging problem.

In the following we say that the suffix starting at $\mathsf{sa}[i]$ is *special* iff $\mathsf{s}[\mathsf{sa}[i] + \mathsf{lcp}[i+1]] = \$$ or, in other words, if the suffix starting at $\mathsf{sa}[i]$ is a substring of the suffix starting at $\mathsf{sa}[i+1]$ (not considering the end-marker \$). For example, in Fig. 1 (right) the suffixes $\mathsf{ab}\$_0$, $\mathsf{abc}\$_1$, $\mathsf{abcab}\$_0$ are all special. We can modify Phase 2 of our algorithm so that it outputs also a bit array xlcp such that $\mathsf{xlcp}[i] = \mathbf{1}$ iff the suffix starting at $\mathsf{sa}[i]$ is special. In the full paper we will formally prove that this modification does not increase the asymptotic cost and requires only $2n$ bits of disk working space.

Our algorithm consists of a sequential scan of the arrays bwt, lcp, and da, and xlcp. We maintain m distinct stacks, $\mathsf{stack}[1], \ldots, \mathsf{stack}[m]$, one for each input sequence; $\mathsf{stack}[k]$ stores only *special* suffixes belonging to sequence k. When the scanning reaches position j, we store the pair $\langle j, \mathsf{lcp}[j+1] \rangle$ in $\mathsf{stack}[\mathsf{da}[j]]$ if and only if $\mathsf{xlcp}[j] = \mathbf{1}$ and $\mathsf{lcp}[j+1] > \tau$. During the scanning we maintain the invariant that for all stack entries $\langle j, \mathsf{lcp}[j+1] \rangle$, $\mathsf{lcp}[j+1]$ is the length of longest common prefix between $\mathsf{s}[\mathsf{sa}[j], n]$ and $\mathsf{s}[\mathsf{sa}[i], n]$, where i is the next position to be scanned. To this end, when we reach position i we remove all entries $\langle j, \mathsf{lcp}[j+1] \rangle$ such that $\mathsf{lcp}[j+1] > \mathsf{lcp}[i+1]$. To do this spending constant time for removed entry requires some additional machinery: We maintain an array of lists top such that $\mathsf{top}[\ell]$ contains the indexes k for which the entry at the top of $\mathsf{stack}[k]$ has LCP component equal to ℓ (this array is a stripped down version of [33]'s list). In addition, we maintain an additional stack lcpStack containing, in increasing order, the values ℓ such that some $\mathsf{stack}[k]$ contains an entry with LCP component equal to ℓ.

At iteration i, we use lcpStack and the lists in $\mathsf{top}[\cdot]$ to reach all $\mathsf{stack}[k]$ containing entries with LCP component greater than $\mathsf{lcp}[i+1]$ and we remove them. After the removal, we update $\mathsf{top}[\ell]$ where ℓ is the LCP value now at the top of $\mathsf{stack}[k]$. Finally, if $\mathsf{xlcp}[i] = \mathbf{1}$ and $\mathsf{lcp}[i+1] > \tau$, we add $\langle i, \mathsf{lcp}[i+1] \rangle$ to $\mathsf{stack}[\mathsf{da}[i]]$; this requires that we also add $\mathsf{da}[i]$ to $\mathsf{top}[\mathsf{lcp}[i+1]]$, and that we remove $\mathsf{da}[i]$ from the list $\mathsf{top}[\ell]$ where ℓ is the previous top LCP value in $\mathsf{stack}[\mathsf{da}[i]]$ (to do this we need to maintain for each element at the top of the stack a pointer to its corresponding da entry in top). Since we perform a constant number of operations per entry, maintaining the above data structures takes $\mathcal{O}(n)$ time overall.

The computation of the overlaps is done as in [33]. When the scanning reaches position i, we check whether $\mathsf{bwt}[i] = \$$. If this is the case, then $\mathsf{s}[\mathsf{sa}[i], n]$ is prefixed by the whole sequence $\mathsf{s}_{\mathsf{da}[i]}$, hence the longest overlap between a prefix of $\mathsf{s}_{\mathsf{da}[i]}$ and a suffix of s_k is given by the element currently at the top of $\mathsf{stack}[k]$, since by construction these stacks only contain special suffixes whose overlap with $\mathsf{s}[\mathsf{sa}[i], n]$ is larger than τ. To spend time proportional to the number of reported overlaps, instead of accessing all stacks we access only those which are non-empty. This requires that we maintain an additional list containing all values ℓ such

that $\mathsf{top}[\ell]$ is non-empty. For each entry ℓ in this list, $\mathsf{top}[\ell]$ gives us the id of the sequences with a suffix-prefix overlap with $\mathsf{da}[i]$ of length ℓ. As in [33], we have to handle differently the case in which the whole $\mathsf{s}_{\mathsf{da}[i]}$ is a suffix of another sequence, but this can be done without increasing the overall complexity. Since we spend constant time for reported overlap, the overall cost of the algorithm, in addition to the scanning of the $\mathsf{bwt}/\mathsf{lcp}/\mathsf{xlcp}/\mathsf{da}$ arrays, is $\mathcal{O}(n + E_\tau)$, where E_τ is the number of suffix-prefix overlaps greater than τ.

5.3 Construction of succinct de Bruijn graphs

A recent remarkable application of compressed data structures is the design of efficiently navigable succinct representations of de Bruijn graphs [3, 7, 8]. Formally, a de Bruijn graph for a collection of strings consists of a set of vertices representing the distinct k-mers appearing in the collection, with a directed edge (u, v) iff there exists a $(k+1)$-mer α in the collection such that $\alpha[1, k]$ is the k-mer associated to u and $\alpha[2, k+1]$ is the k-mer associated to v.

The starting point of all de Bruijn graphs succinct representation is the BOSS representation [8], so called from the authors' initials. For simplicity we now describe the BOSS representation of a k-order de Bruijn graph using the lexicographic order of k-mers, instead of the co-lexicographic order as in [8], which means we are building the graph with the direction of the arcs reversed. This is not a limitation since arcs can be traversed in both directions (or we can apply our construction to the input sequences reversed).

Consider the N k-mers appearing in the collection sorted in lexicographic order. For each k-mer α_i consider the array C_i of distinct characters $c \in \Sigma \cup \{\$\}$ such that $c\alpha_i$ appears in the collection. The concatenation $W = C_1 C_2 \cdots C_N$ is the first component of the BOSS representation. The second component is a binary array *last*, with $|last| = |W|$, such that $last[j] = \mathbf{1}$ iff $W[j]$ is the last entry of some array C_i. Clearly, there is a bijection between entries in W and graph edges; in the array *last* each sequence $\mathbf{0}^i\mathbf{1}$ $(i \geq 0)$ corresponds to the outgoing edges of a single vertex with outdegree $i + 1$. Finally, the third component is a binary array W^-, with $|W^-| = |W|$, such that $W^-[j] = \mathbf{1}$ iff $W[j]$ comes from the array C_i, where α_i is the lexicographically smallest k-mer prefixed by $\alpha_i[1, k-1]$ and preceded by $W[j]$ in the collection. Informally, this means that α_i is the lexicographically smallest k-mer with an outgoing edge reaching $W[j]\alpha_i[1, k-1]$. Note that the number of $\mathbf{1}$'s in *last* and W^- is exactly N, i.e. the number of nodes in the de Bruijn graph.

We now show how to compute W, *last* and W^- by a sequential scan of the bwt and lcp array. The crucial observation is that the suffix array range prefixed by the same k-mer α_i is identified by a range $[b_i, e_i]$ of LCP values satisfying $\mathsf{lcp}[b_i] < k$, $\mathsf{lcp}[\ell] \geq k$ for $\ell = b_i+1, \ldots, e_i$ and $\mathsf{lcp}[e_i + 1] < k$. Since k-mers are scanned in lexicographic order, by keeping track of the corresponding characters in the array $\mathsf{bwt}[b_i, e_i]$ we can build the array C_i and consequently W and *last*. To compute W^- we simply need to keep track also of suffix array ranges corresponding to $(k-1)$-mers. Every time we set an entry $W[j] = c$ we set $W^-[j] = \mathbf{1}$ iff this is the first occurrence of c in the range corresponding to the current $(k-1)$-mers.

If, in addition to the bwt and lcp arrays, we also scan the da array, then we can keep track of which sequences contain any given graph edge and therefore obtain a succinct representation of the colored de Bruijn graph [29]. Finally, we observe that if our only objective is to build the k-order de Bruijn graph, then we can stop the phase 2 of our algorithm after the k-th iteration. Indeed, we do not need to compute the exact values of LCP entries greater than k, and also we do not need the exact BWT but only the BWT characters sorted by their length k context.

References

1 Markus J. Bauer, Anthony J. Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theor. Comput. Sci.*, 483:134–148, 2013.

2 Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *STOC*, pages 148–193. ACM, 2014.

3 Djamal Belazzougui, Travis Gagie, Veli Mäkinen, Marco Previtali, and Simon J. Puglisi. Bidirectional variable-order de Bruijn graphs. In *LATIN*, volume 9644 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2016.

4 Paola Bonizzoni, Gianluca Della Vedova, Yuri Pirola, Marco Previtali, and Raffaella Rizzi. Computing the BWT and LCP array of a set of strings in external memory. *CoRR*, abs/1705.07756, 2017.

5 Paola Bonizzoni, Gianluca Della Vedova, Yuri Pirola, Marco Previtali, and Raffaella Rizzi. An external-memory algorithm for string graph construction. *Algorithmica*, 78(2):394–424, 2017. doi:10.1007/s00453-016-0165-4.

6 Paola Bonizzoni, Gianluca Della Vedova, Yuri Pirola, Marco Previtali, and Raffaella Rizzi. Divide and conquer computation of the multi-string BWT and LCP array. In *Proc. Computability in Europe (CiE)*, 2018. To appear.

7 Christina Boucher, Alexander Bowe, Travis Gagie, Simon J. Puglisi, and Kunihiko Sadakane. Variable-order de Bruijn graphs. In *DCC*, pages 383–392. IEEE, 2015.

8 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *WABI*, volume 7534 of *Lecture Notes in Computer Science*, pages 225–235. Springer, 2012.

9 S. Burkhardt and J. Kärkkäinen. Fast lightweight suffix array construction and checking. In *Proc. 14th Symposium on Combinatorial Pattern Matching (CPM '03)*, pages 55–69. Springer-Verlag LNCS n. 2676, 2003.

10 Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital SRC Research Report, 1994.

11 Anthony J. Cox, Fabio Garofalo, Giovanna Rosone, and Marinella Sciortino. Lightweight LCP construction for very large collections of strings. *J. Discrete Algorithms*, 37, 2016.

12 Lavinia Egidi and Giovanni Manzini. Lightweight BWT and LCP merging via the Gap algorithm. In *SPIRE*, volume 10508 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2017.

13 P. Ferragina, T. Gagie, and G. Manzini. Lightweight data indexing and compression in external memory. *Algorithmica*, 2011.

14 Simon Gog and Enno Ohlebusch. Compressed suffix trees: Efficient computation and storage of LCP-values. *ACM Journal of Experimental Algorithmics*, 18, 2013. doi:10.1145/2444016.2461327.

15 D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

16 Dan Gusfield, Gad M. Landau, and Baruch Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Inf. Process. Lett.*, 41(4):181–185, 1992.

17 James Holt and Leonard McMillan. Merging of multi-string BWTs with applications. *Bioinformatics*, 30(24):3524–3531, 2014.

18 Juha Kärkkäinen and Dominik Kempa. LCP array construction in external memory. *ACM Journal of Experimental Algorithmics*, 21(1):1.7:1–1.7:22, 2016.

19 Juha Kärkkäinen and Dominik Kempa. Engineering a lightweight external memory suffix array construction algorithm. *Mathematics in Computer Science*, 11(2):137–149, 2017.

20 D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1998.

21 M. Oguzhan Külekci, Jeffrey Scott Vitter, and Bojian Xu. Efficient maximal repeat finding using the burrows-wheeler transform and wavelet tree. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(2):421–429, 2012.

22 Felipe A. Louza, Simon Gog, and Guilherme P. Telles. Inducing enhanced suffix arrays for string collections. *Theor. Comput. Sci.*, 678:22–39, 2017.

23 Felipe A. Louza, Guilherme P. Telles, Steve Hoffmann, and Cristina D. A. Ciferri. Generalized enhanced suffix array construction in external memory. *Algorithms for Molecular Biology*, 12(1):26:1–26:16, 2017.

24 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing.* Cambridge University Press, 2015.

25 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An extension of the Burrows-Wheeler transform. *Theor. Comput. Sci.*, 387(3):298–312, 2007.

26 G. Manzini. Two space saving tricks for linear time LCP computation. In *Proc. of 9th Scandinavian Workshop on Algorithm Theory (SWAT '04)*, pages 372–383. Springer-Verlag LNCS n. 3111, 2004.

27 G. Manzini and P. Ferragina. Engineering a lightweight suffix array construction algorithm. In *Proc. 10th European Symposium on Algorithms (ESA)*, pages 698–710. Springer Verlag LNCS n. 2461, 2002.

28 Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy, Rachel Coston, Steven L. Salzberg, and Aleksey V. Zimin. Mummer4: A fast and versatile genome alignment system. *PLoS Computational Biology*, 14(1), 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 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *SODA*, pages 408–424. SIAM, 2017.

31 G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007.

32 Ge Nong. Practical linear-time $O(1)$-workspace suffix sorting for constant alphabets. *ACM Trans. Inf. Syst.*, 31(3):15, 2013.

33 Enno Ohlebusch and Simon Gog. Efficient algorithms for the all-pairs suffix-prefix problem and the all-pairs substring-prefix problem. *Inf. Process. Lett.*, 110(3):123–128, 2010.

34 Enno Ohlebusch, Simon Gog, and Adrian Kügel. Computing matching statistics and maximal exact matches on compressed full-text indexes. In *SPIRE*, volume 6393 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2010.

35 William H. A. Tustumi, Simon Gog, Guilherme P. Telles, and Felipe A. Louza. An improved algorithm for the all-pairs suffix-prefix problem. *J. Discrete Algorithms*, 37:34–43, 2016.

Kermit: Guided Long Read Assembly using Coloured Overlap Graphs

Riku Walve[1]
Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki, Helsinki, Finland
riku.walve@helsinki.fi
https://orcid.org/0000-0003-0397-003X

Pasi Rastas[2]
Institute of Biotechnology, University of Helsinki, Helsinki, Finland
pasi.rastas@helsinki.fi

Leena Salmela[1]
Department of Computer Science, Helsinki Institute for Information Technology HIIT, University of Helsinki, Helsinki, Finland
leena.salmela@helsinki.fi
https://orcid.org/0000-0002-0756-543X

Abstract

With long reads getting even longer and cheaper, large scale sequencing projects can be accomplished without short reads at an affordable cost. Due to the high error rates and less mature tools, *de novo* assembly of long reads is still challenging and often results in a large collection of contigs. Dense linkage maps are collections of markers whose location on the genome is approximately known. Therefore they provide long range information that has the potential to greatly aid in *de novo* assembly. Previously linkage maps have been used to detect misassemblies and to manually order contigs. However, no fully automated tools exist to incorporate linkage maps in assembly but instead large amounts of manual labour is needed to order the contigs into chromosomes. We formulate the genome assembly problem in the presence of linkage maps and present the first method for guided genome assembly using linkage maps. Our method is based on an additional cleaning step added to the assembly. We show that it can simplify the underlying assembly graph, resulting in more contiguous assemblies and reducing the amount of misassemblies when compared to *de novo* assembly.

2012 ACM Subject Classification Applied computing → Sequencing and genotyping technologies, Mathematics of computing → Graph theory

Keywords and phrases Genome assembly, Linkage maps, Coloured overlap graph

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.11

1 Introduction

High-throughput, second generation, sequencing technologies made large scale *de novo* assemblies possible and commonplace. Their short read lengths however still pose a major problem to this day. Third generation, long read sequencing technologies, such as single-molecule real-time sequencing (SMRT) and Oxford Nanopore (ONT) are promising but their

[1] Supported by Academy of Finland (grants 308030 and 314170).
[2] Supported by Jane and Aatos Erkko Foundation.

error rates have made assemblies difficult in practice. Therefore most long read assemblers include an error correction step to reduce the error rate.

Recently introduced Minimap-Miniasm workflow [12] has given new insight towards an error correction-free pipeline for long read assemblies. Minimap finds useful overlaps in long reads with high error rates and makes long read-only assembly projects practical and highly efficient. However finding the overlaps between reads with high error rates becomes impractical in very large data sets and splitting the reads into smaller sets is not possible without additional information on the reads.

Compared to *de novo* assembly, where the only available information for the assembly are the reads, *guided genome assembly* has additional data that gives information on the positions of the reads. Normally this additional data is a reference genome of a closely related species. The reads can be aligned to the reference genome, which results in a linear ordering of the reads. This clearly makes it easier to then assemble the reads.

Directly guiding the assembly this way makes it hard to get an assembly that is higher quality than the reference. This becomes an issue when no high quality reference genome exists, or if the donor genome deviates too far from the reference genome. In this paper, we guide the assembly using linkage maps.

Linkage maps (also called genetic linkage maps or genetic maps) [17] are a useful technique to orient and place contigs within a chromosome and to detect misassembled contigs. The linkage maps themselves consists of variable genetic markers, typically called relative to some draft assembly. The markers are derived from a set of variations, such as single-nucleotide variations (SNVs), found from a sequenced *cross*, a population of related individuals. SNVs that are close to each other in the genome are more likely to be inherited together than SNVs that are more distant from each other. Linkage maps can therefore be constructed by genotyping the individuals in the cross and examining the probabilities of SNVs being inherited together. With a large enough set of variations in population, the linkage map becomes dense enough to potentially place long reads directly.

In this paper, we formulate the genome assembly problem in the presence of linkage maps. We propose the first method to directly use linkage maps in genome assembly. Our method disentangles the overlap graph by removing false edges based on the linkage map. Our experimental results show that the method is able to simplify the overlap graph and we further show that our method decreases the number of misassemblies and improves the N50 statistic as compared to *de novo* assembly without linkage maps.

This linkage map-guided genome assembly can be seen as a generalisation of reference-guided genome assembly. With a hypothetical linkage map of evenly spaced markers, the linkage map-guided assembly becomes essentially reference-guided assembly as each read can be placed on the genome unambiguously.

We have implemented our method in a tool called Kermit which is freely available at `https://github.com/rikuu/kermit`.

2 Related Work

Despite the emergence of long read sequencing technologies like PacBio and ONT and the development of long read assemblers [11, 12, 7, 10], auxiliary long range data is still needed to organise the resulting scaffolds or contigs to chromosomes for large eukaryotic genomes. Depending on the characteristics of the species of interest such data may include optical mapping data, Hi-C data, or linkage maps.

Linkage maps consist of a set of markers, typically SNVs, on a genome. The location of the markers with respect to each other is at least approximately known. Typically a linkage map successfully assigns the markers to chromosomes and a partial order of the markers within each chromosome is known. Markers can be localised on contigs or scaffolds and the linkage map can be used to correct the scaffolds and to order them into chromosomes [1, 18, 4]. However, currently the ordering of scaffolds based on a linkage map is a time-consuming, error-prone and mostly manual process as no fully automated tools exist [8]. Furthermore, all current tools use linkage maps as a post processing step after assembly, whereas our work integrates the linkage maps already in the assembly phase.

Chromonomer [5] attempts to correct and orient scaffolds based on a linkage map. It first finds a non-conflicting set of markers in the linkage map. It then assigns orientation to scaffolds containing more than one recombination point and splits scaffolds if they conflict with the linkage map. Finally, Chromonomer produces a visualisation of the linkage map and the scaffolds. Another tool that facilitates the visualisation of linkage maps and corresponding genome assemblies is ArkMAP [15]. However, ArkMAP focuses on visual exploration including cross species comparisons and as such does not support automatic ordering of scaffolds into chromosomes.

Similarly to linkage maps also optical mapping data can be used for improving genome assemblies. Also for optical mapping data the main focus has been to use the optical map in a post processing step after genome assembly. AGORA [14] is one of the few methods integrating optical map data with genome assembly. It is a de Bruijn graph based assembler that uses optical map data to eliminate alternative paths that are not consistent with the optical map. The method by Alipanahi et al. [2] for disentangling the de Bruijn graph using optical map data is more related to our work. They map the long reads to the genome wide optical map and use this mapping to produce a positional de Bruijn graph which resolves most ambiguities in the de Bruijn graph. In our work we similarly first get a preliminary ordering of long reads based on a linkage map and then use this ordering to disentangle the overlap graph.

3 Definitions

3.1 De Novo Assembly

De novo assembly, the problem of assembling a genome given only a set of reads, has historically been given solutions in two different categories. The de Bruijn graph-based (DBG) assembly algorithms, such as SPAdes [3], and Overlap-Layout-Consensus (OLC) algorithms, such as Canu [11] and Miniasm [12].

OLC-assemblers attempt to first find pair-wise overlaps between reads which is the bottleneck of this approach. From these overlaps, a directed graph of the set of reads, called an overlap graph, is laid out by assigning edges between reads if an overlap is observed. The graph is then simplified by removing all transitive edges. Ideally the graph would be simple enough to unambiguously spell a genome assembly by following the edges from one side of the graph to the other. This is actually never the case and some sophisticated method for transforming the graph as close as possible to such a case is the goal of the layout step.

DBG-assemblers attempt to simplify the entire process by not looking for overlaps between the reads. Instead they take all possible $k-1$ length overlaps between substrings of length k from the set of reads to construct a de Bruijn graph. The methods for finding the genome from the overlap graph in the OLC setup mostly apply here too.

Though clearly simpler than the OLC assembly algorithms, the effectiveness of further splitting the reads somewhat diminishes the usefulness of using long reads for genome assembly. In this paper we are only looking at the OLC category of assembly algorithms, more specifically, we are looking at Miniasm [12], a very simplistic and highly efficient implementation of the ideas in OLC-assemblers.

Unlike traditional OLC-assemblers, Miniasm only implements the overlap and layout steps. This reduces the base-level quality of the resulting assembly but does not greatly affect the large scale structure. It also does not use any overly sophisticated method for finding the genome in the overlap graph.

To choose an assembly path, Miniasm finds *unitigs* which are maximal non-branching paths in the overlap graph. Such paths are simple to find and intuitively give the maximal unambiguous and nonrepetitive sequences that the overlap graph can spell without changing the graph. The problem of finding the unitigs from the overlap graph can be stated as follows:

▶ **Definition 1.** *Unitig assembly problem.* Given a directed graph $G = (V, E)$, find all maximal paths $P = v_1 v_2 \cdots v_n$ such that

$$\forall v_i \in \{v_1, \ldots, v_{n-1}\}, \deg^+ v_i = \deg^- v_{i+1} = 1.$$

Unitigs can be efficiently found by first looking for a vertex with either zero or more than one incoming edge and exactly one outgoing edge. Following the outgoing edges until we find a vertex with more than one incoming edge or zero or more than one outgoing edge constructs a path spelling a unitig.

3.2 Guided Assembly

Reference-guided genome assembly can be done by aligning the read set to a reference and partitioning the reads based on the alignments into smaller, similar sets of reads [19]. The small sets are then assembled into contigs and later the set of contigs are assembled into super-contigs.

For linkage map-guided assembly, we partition the reads based on the linkage map. A linkage map is a set of markers M which are assigned to a set of bins B. Each marker $m \in M$ belongs to a single bin $b \in B$. The bins are further assigned to chromosomes and within each chromosome the order of the bins is given.

We assume now that each read has been assigned a set of bins based on the linkage map. Now this coarse-grained ordering tells if a subset of reads clearly belong before or after another subset of reads. We encode this ordering into the overlap graph by assigning each vertex a set of colours representing the bins, resulting in a *coloured overlap graph*.

A coloured overlap graph is thus a directed graph $G = (V, E)$ accompanied by a colouring $c : V \to \mathcal{P}(\mathbb{N})$, where $\mathcal{P}(\mathbb{N})$ is the power set of natural numbers. In the coloured overlap graph we define the *colour consistent indegree* $\deg_c^-$ of a vertex v_i to be the number of in-neighbours of v_i that have at least one colour that is the same as or adjacent to a colour of v_i, i.e.

$$\deg_c^- v_i = |\{v_j | (v_j, v_i) \in E \text{ and } \exists c_j \in c(v_j), c_i \in c(v_i) \text{ s.t. } |c_j - c_i| \leq 1\}|.$$

Similarily we define the *colour consistent outdegree* $\deg_c^+$ as

$$\deg_c^+ v_i = |\{v_j | (v_i, v_j) \in E \text{ and } \exists c_j \in c(v_j), c_i \in c(v_i) \text{ s.t. } |c_j - c_i| \leq 1\}|.$$

The problem of guided assembly can then be modelled as finding a *rainbow path* in the coloured overlap graph. A rainbow path is a path such that no two vertices have the same colour [6]. We use a modified variant of rainbow paths; we allow paths to reuse a colour in consecutive vertices and we require the colours of a path to be consecutive and increasing. I.e. colour i must be followed by colour $i+1$ on the path. A more formal definition of this assembly problem can be stated as:

▶ **Definition 2.** *Guided unitig assembly problem.* Given a directed graph $G = (V, E)$, and a colouring $c : V \to \mathcal{P}(\mathbb{N})$, find all maximal paths $P = v_1 v_2 \cdots v_n$ such that

$$\forall v_i \in \{v_1, \ldots, v_{n-1}\} \deg_c^+ v_i = \deg_c^- v_{i+1} = 1$$

and

$$\forall c \in c(v_i), c \geq \max\left(c(v_{i-1})\right).$$

We note that the above definition only recovers forward strand paths in the coloured overlap graph. However due to the structure of the graph, for each reverse strand path there also exists a corresponding forward strand path in the graph.

Rather than modifying the layout step of the OLC-assembly pipeline, we implement a graph cleaning step, which attempts to remove edges that make unitigs not be rainbow paths.

4 Methods

4.1 Overview of Our Method

Constructing a linkage map [17] involves generating a draft assembly and localising the markers, which are typically single nucleotide variations, on this draft assembly. The markers are then further placed into bins based on the hereditary patterns seen in a cross-bred population of individuals. The bins are assigned to chromosomes and the order of the bins within each chromosome is known. Thus the markers give a partial order of the genome.

The input to our method is the draft assembly on which the linkage map was built, the linkage map, and long reads. We first use Minimap2 [13] to map the long reads on the draft assembly to assign colours (i.e. bin numbers) to the long reads. Miniasm [12] is then used to build the overlap graph of the long reads and missing colours are propagated in the overlap graph. The overlap graph is then cleaned based on the colour assignments of the reads. Finally the unitigs are read from the cleaned overlap graph. Figure 1 shows an overview of our method. The colouring, propagation, and cleaning steps are discussed in more details in the following subsections.

4.2 Colouring

To colour the underlying overlap graph, we map all the reads against the draft genome using Minimap2 [13] and store the longest mappings for each read. We extend each of the longest mappings into naive linear "alignments" by stretching the start and end positions to cover the entire read. As the insertion errors are the dominant error type [20], we limit the number of characters by which we extend the mappings (by default we use a limit of 250 bp).

All the extended mappings are then stored in a simple query structure, where each chromosome in the reference genome is split into equal length blocks. For each marker in the linkage map, we query the index for reads that overlap with the marker and assign every overlapping read the colour the marker in the linkage map belongs to.

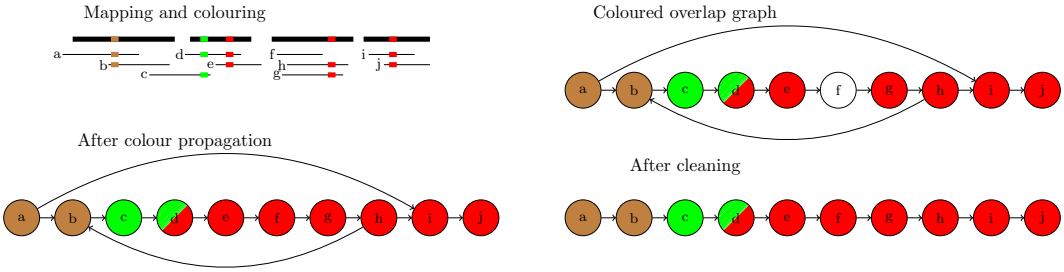


Figure 1 Overview of our method. First the reads are mapped to the draft assembly and assigned colours (top left). Each colour represents one bin in the linkage map. In this example we have three bins (brown, green and red) and the ordering of the bins is brown < green < red. All bins belong to the same chromosome. Miniasm is then used to construct the overlap graph which is augmented with the colours. Next vertex f is coloured through the colour propagation process. Finally we remove the edges (a, i) and (h, b) because they have inconsistent colourings.

As there can still be reads with no mapping to the reference, we attempt to colour them using the overlap graph. For each uncoloured vertex u we find the set of coloured vertices $V_c(u)$ that are reachable from u by paths that traverse only uncoloured vertices. The uncoloured vertex u is then given all the colours that the vertices in $V_c(u)$ have, i.e. $c(u) = \bigcup_{v \in V_c(u)} c(v)$. The set $V_c(u)$ can be found by a breadth first search on the graph starting at vertex u.

If a coloured vertex is too far from the uncoloured vertex, we get an over-approximation of the colour for the vertex. To reduce this effect, we apply a limit to the depth of the search (by default 10). We can also get conflicting colours from the propagation. If there are missing colours between the propagated colours, we are skipping some part of the genome entirely. As we cannot use the colouring to usefully clean the graph, we simply remove the read from the graph entirely.

4.3 Graph Cleaning

To make sure any unitig path is a valid rainbow path, we remove any edges between vertices with inconsistent colourings from the overlap graph. If the vertices share a colour, the colourings are always consistent as they can be merged without affecting the connectivity of the colours in the graph. Edges between vertices with different colours are inconsistent if there are no adjacent colours between the vertices. Such an edge could never be part of a rainbow path because the path would not have consecutive colours.

Therefore we define an edge (v_i, v_j) to be *inconsistently coloured* if there are no colours $c_i \in c(v_i)$ and $c_j \in c(v_j)$ such that the colours c_i and c_j would be the same or consecutive, i.e. $|c_i - c_j| \leq 1$. Our graph cleaning step remove

5 Results

We implemented our method in a tool called Kermit. Kermit uses Miniasm [12] to find the overlaps between the reads and to perform the layout step to find unitigs. In between these two steps Kermit colours the vertices of the overlap graph and cleans the graph by removing inconsistently coloured edges as explained in the previous section. We compared Kermit to the Miniasm pipeline [12]. Both the overlap and the mapping steps were done using Minimap2 v2.9 [13], an improved implementation of Minimap. All experiments were run on 32 GB RAM machines equipped with 8 cores.

Table 1 Summary of read data used in the experiments.

Data set	Reads	Total bases	Coverage	Accession
S. cerevisiae (simulated)	36,639	486,048,334	39.98	simulated
C. elegans (simulated)	296,230	3,930,689,562	39.99	simulated
S. cerevisiae	52,208	690,899,144	56.83	PacBio DevNet[1)]
C. elegans	740,776	8,118,404,281	82.59	PacBio DevNet[2)]
H. erato	10,818,653	27,094,241,328	60.89	SRR3476970 SRR4039325
B. pendula	1,898,360	19,032,363,776	49.71	ERR2003767 ERR2003768

1) `https://github.com/PacificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs`
2) `https://github.com/PacificBiosciences/DevNet/wiki/C.-elegans-data-set`

Table 2 Summary of linkage maps used in the experiments.

Data set	Markers	Marker density	Bins	Bin density	Reference
S. cerevisiae (simulated)	100,009	0.008	19,283	0.002	simulated
C. elegans (simulated)	750,004	0.008	162,601	0.002	simulated
H. erato	2,781,314	0.007	145,863	0.002	Van Belleghem et al. [4] Generated by LepMap3 [17]
B. pendula	2,979,993	0.007	925,123	0.002	Salojärvi et al. [18]

5.1 Data

We performed experiments both on simulated and real data. All data used in the experiments is summarized in Tables 1 and 2.

First we performed experiments using both simulated reads and a simulated linkage map for *S. cerevisiae* and *C. elegans*. We simulated reads using SimLoRD [20] to a coverage of 40x, sampling read lengths from a real PacBio data set with shortest ($<$ 10,000bp) reads filtered out. To generate a simulated linkage map, markers were randomly placed on the reference and binned such that the bins are separated by at least 200bp. The number of markers was chosen to give a marker density similar to real linkage maps. The distance for binning is arbitrary; as real linkage maps are based on fragmented and misassembled draft genomes, the bins are rarely contiguous. The chosen values give similar densities of bins as the real linkage maps as shown in Table 2. These experiments on simulated read and linkage map data allow us to evaluate the colouring and cleaning steps because the origin of each read is known.

To understand how our method performs on real data, we first ran experiments using real PacBio reads for *S. cerevisiae* and *C. elegans* but still using the simulated linkage map as no real linkage maps exist for these species. Good reference genomes are available for these genomes so we could evaluate also the correctness of the resulting assemblies on these data sets. Finally to show that our method works on real linkage maps, we ran experiments on real PacBio reads and real linkage maps for *H. erato* and *B. pendula*. The genomes for these species are in draft stage so we could not reliably measure the correctness of these assemblies.

Table 3 Number of reads fully inside and outside their acceptable colour ranges.

	Reads inside	Reads outside
S. cerevisiae	36,562 (99.79%)	31 (0.08%)
C. elegans	296,134 (99.97%)	3 (0.001%)

Table 4 Number of edges supported by the positions the reads were simulated from. Graphs marked cleaned are also using the graph cleaning steps that are already implemented in Miniasm.

	Graph	True edges	False edges
S. cerevisiae	Miniasm	76,538 (99.39%)	466 (0.60%)
	Kermit	76,518 (99.93%)	52 (0.07%)
	Miniasm cleaned	7,146 (99.92%)	6 (0.08%)
	Kermit cleaned	7,114 (100.0%)	0 (0.0%)
C. elegans	Miniasm	668,012 (99.80%)	1,306 (0.19%)
	Kermit	667,970 (99.99%)	58 (0.003%)
	Miniasm cleaned	60,416 (99.95%)	28 (0.05%)
	Kermit cleaned	60,356 (99.997%)	2 (0.003%)

5.2 Colouring

We first evaluated the results of the colouring and propagation steps on the simulated reads and linkage maps. Because the reads are simulated, we know for each read the position where it derives from. Therefore we can also deduce the correct colours for each read as follows. For every read there are two markers in the linkage map that form the upper and lower limit of acceptable colours the read can get. All the colours should be between the colour of last marker before the read and the first marker after the read.

We are mostly interested in the set of reads that are fully within the correct range, i.e. no colour given to the read is incorrect, and the set of reads that are fully outside the correct range.

As can be expected, Table 3 shows that a vast majority of the reads are completely inside their colour ranges. The small amount of reads outside the range are reads that have been mapped to an entirely wrong position of the reference.

5.3 Cleaning

To evaluate the effectiveness of removing colour crossing edges, we find the simulated positions of the two reads corresponding to each edge in the graph and check whether they overlap in the reference genome. We consider those overlapping reads to be true in the sense that using that single edge in a contig would spell the correct sequence.

Miniasm already implements a cleaning step which is solely based on the read data. We counted the number of true and false edges both with and without this cleaning step to understand how each cleaning step affects the overlap graph.

Table 4 shows how Kermit is able to improve the percentages of true edges by removing false edges in the graph. Though the amounts of false edges removed is relatively small, any single wrong edge used in the assembly can cause a misassembly or break a unitig.

Table 5 Assembly statistics for simulated *S. cerevisiae* and *C. elegans* reads and simulated linkage maps.

	S. cerevisiae		*C. elegans*	
Assembly	Miniasm	Kermit	Miniasm	Kermit
# of contigs	26	22	291	261
Total length	11,831,837 (97.32%)	11,728,421 (96.47%)	102,040,817 (103.81%)	101,632,493 (103.40%)
N50	605,399	640,779	2,337,914	2,293,633
NGA50	565,122	585,849	2,070,983	2,337,914
Misassemblies	2	1	7	7

Table 6 Assembly statistics for real *S. cerevisiae* and *C. elegans* reads and simulated linkage maps.

	S. cerevisiae		*C. elegans*	
Assembly	Miniasm	Kermit	Miniasm	Kermit
# of contigs	31	29	146	141
Total length	12,118,143 (99.68%)	11,997,376 (98.69%)	106,229,975 (108.08%)	103,811,685 (105.62%)
N50	732,688	763,111	2,851,252	2,851,288
NGA50	345,801	376,187	252,615	254,858
Misassemblies	61	58	1,870	1,768

5.4 Assembly

Lastly, we compare the actual assemblies produced by our tool to those produced by Miniasm. To get a better picture of the assemblies, we also applied a consensus tool, Racon v1.2.0 [16], on the *S. cerevisiae* and *C. elegans* assemblies. On the *H. erato* and *B. pendula* assemblies Racon was very slow so we did not run it on those assemblies. The assembly statistics were generated with QUAST v4.6.1 [9].

Table 5 shows how the assembly statistics are improved using Kermit on the simulated *S. cerevisiae* and *C. elegans* reads and simulated linkage maps. The number of contigs is reduced and the NGA50 statistic is increased indicating a more contiguous assembly. Also the number of misassemblies is either reduced or stays the same.

Next we evaluated Kermit on the real read data and simulated linkage map of *S. cerevisiae* and *C. elegans*. Table 6 shows that also in this case the number of contigs and the number of misassemblies is reduced, whereas the NGA50 statistic is increased.

Finally we ran experiments on real read data and real linkage maps (*H. erato* and *B. pendula*). For these species only draft assemblies are available and thus we could not validate the produced assemblies and compute the number of misassemblies or the NGA50 statistics. Table 7 shows that for both data sets the number of contigs is reduced and N50 is increased indicating a more contiguous assembly. We also note that both Miniasm and Kermit struggle on the *H. erato* data. We believe this is largely due to the fact that we needed to pool PacBio reads from two experiments using two different individuals to have enough reads for assembly. This introduces heterogeneity to the data and makes assembly challenging.

Table 7 Assembly statistics for real *H. erato* and *B. pendula* reads and real linkage maps.

	H. erato		*B. pendula*	
Assembly	Miniasm	Kermit	Miniasm	Kermit
# of contigs	7,444	6,091	2,201	1,587
Total length	327,725,353 (86.24%)	280,881,758 (73.92%)	473,300,369 (107.57%)	425,356,395 (96.67%)
N50	58,892	60,356	435,830	539,400

Table 8 Wall clock times for all steps taken by the tools. The consensus phase was very slow on the big genomes of *H. erato* and *B. pendula* so it was not run on those data sets.

	Tool	Overlap	Map	Colour	Layout	Consensus	Total
S. cerevisiae (simulated)	Miniasm	52s	-	-	6s	4min 52s	5min 50s
	Kermit	52s	6s	0s	6s	3min 29s	4min 38s
C. elegans (simulated)	Miniasm	9min 55s	-	-	1min 58s	28min 19s	40min 17s
	Kermit	9min 55s	2min 17s	5s	55s	28min 57s	42min 9s
S. cerevisiae	Miniasm	1min 9	-	-	8s	2min 45s	4min 2s
	Kermit	1min 09s	10s	1s	8s	2min 44s	4min 12s
C. elegans	Miniasm	16min 54s	-	-	4min	53min 28s	1h 14min
	Kermit	16min 54s	4min 8s	11s	2min 29s	50min 29s	1h 14min
H. erato	Miniasm	8h 40min	-	-	1h 30min	-	10h 10min
	Kermit	8h 40min	9min	2min	2h 42min	-	11h 32min
B. pendula	Miniasm	3h 24min	-	-	1h 32min	-	4h 57min
	Kermit	3h 24min	9min	17s	1h 37min	-	5h 11min

5.5 Performance

We recorded the wall clock time for all experiments. Table 8 shows that Kermit needs at most 5% more time than Miniasm on all data sets except *H. erato* on which it needs 13% more time. In some cases the total running time is even reduced. Additionally, we see that the consensus step using Racon easily dominates the pipeline in terms of time to complete.

6 Conclusions

We defined guided assembly with linkage maps by extending the unitig assembly model. Our method, Kermit, is implemented as a graph cleaning step and the contigs are generated with a simple unitig algorithm. As such the graph cleaning step could be used as a preprocessing step of a more complicated traversal algorithm for retrieving the contigs. Colouring the reads also leads naturally into non-overlapping bins of reads, that can be assembled independently. This allows massive parallelism in the assembly and could make more sophisticated assembly algorithms practical.

When defined as an independent graph cleaning step, our method guiding the assembly could be applied not only to other OLC-assemblers, but also to DBG-assemblers. In this case, the colours would be assigned to reads and the k-mers would get all colours present in reads where they derive from.

Our experiments show that Kermit successfully removes false edges from the overlap graph. Furthermore we showed that with only a modest increase in runtime Kermit improves the contiguity and correctness of assembly as compared to the Miniasm pipeline.

References

1. V. Ahola, R. Lehtonen, P. Somervuo, et al. The Glanville fritillary genome retains an ancient karyotype and reveals selective chromosomal fusions in Lepidoptera. *Nature Communications*, 5:4737, 2014.
2. B. Alipanahi, L. Salmela, S.J. Puglisi, M. Muggli, and C. Boucher. Disentangled long-read de Bruijn graphs via optical maps. In R. Schwartz and K. Reinert, editors, *WABI 2017*, volume 88 of *LIPIcs*, pages 1:1–1:14, Dagstuhl, Germany, 2017.
3. A. Bankevich, S. Nurk, D. Antipov, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol.*, 19(5):455–477, 2012.
4. S.M. Van Belleghem, P. Rastas, A. Papanicolalaou, et al. Complex modular architecture around a simple toolkit of wing pattern genes. *Nature Ecology & Evolution*, 1:0052, 2017.
5. J. Catchen. Chromonomer. http://catchenlab.life.illinois.edu/chromonomer/, 2015. Accessed: 2018-04-27.
6. G. Chartrand, GL. Johns, KA. McKeon, and P. Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1):85–98, 2008.
7. C.-S. Chin, P. Peluso, F.J. Sedlazeck, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature Methods*, 13:1050–1054, 2016.
8. J.L. Fierst. Using linkage maps to correct and scaffold de novo genome assemblies: methods, challenges, and computational tools. *Frontiers in Genetics*, 6:220, 2015.
9. A. Gurevich, V. Saveliev, N. Vyahhi N, and G. Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.
10. M. Kolmogorov, J. Yuan, Y. Lin, and P. Pevzner. Assembly of long error-prone reads using repeat graphs. In *Proc. RECOMB 2018*, pages 261–263, 2018.
11. S. Koren, B.P. Walenz, K. Berlin, J.R. Miller, N.H. Bergman, and A.M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res.*, 27:722–736, 2017.
12. H. Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.
13. H. Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 2018. (To appear).
14. H.C. Lin, S. Goldstein, L. Mendelowitz, S. Zhou, J. Wetzel, D.C. Schwartz, and M. Pop. AGORA: assembly guided by optical restriction alignment. *BMC Bioinformatics*, 13:189, 2012.
15. T. Paterson and A. Law. ArkMAP: integrating genomic maps across species and data sources. *BMC Bioinformatics*, 14:246, 2013.
16. R. Vaser R, I. Sovic, N. Nagarajan, and M. Sikic. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27:737–746, 2017.
17. P. Rastas. Lep-MAP3: robust linkage mapping even for low-coverage whole genome sequencing data. *Bioinformatics*, 33(23):3726–3732, 2017.
18. J. Salojärvi, O.P. Smolander, K. Nieminen, et al. Genome sequencing and population genomic analyses provide insights into the adaptive landscape of silver birch. *Nature Genetics*, 49:904–912, 2017.
19. K. Schneeberger, S. Ossowski, F. Ott, et al. Reference-guided assembly of four diverse Arabidopsis thaliana genomes. *PNAS*, 108(25):10249–10254, 2011.
20. B.K. Stöcker, J. Köster, and S. Rahmann. SimLoRD: Simulation of long read data. *Bioinformatics*, 32(17):2704–2706, 2016.

A Succinct Solution to Rmap Alignment

Martin D. Muggli
Department of Computer Science, Colorado State University
martin.muggli@colostate.edu
https://orcid.org/0000-0002-9283-0049

Simon J. Puglisi[1]
Department of Computer Science, University of Helsinki, Finland
https://orcid.org/0000-0001-7668-7636

Christina Boucher
Department of Computer and Information Science and Engineering, University of Florida
https://orcid.org/0000-0001-9509-9725

Abstract

We present KOHDISTA, which is an index-based algorithm for finding pairwise alignments between single molecule maps (*Rmaps*). The novelty of our approach is the formulation of the alignment problem as automaton path matching, and the application of modern index-based data structures. In particular, we combine the use of the Generalized Compressed Suffix Array (GCSA) index with the wavelet tree in order to build KOHDISTA. We validate KOHDISTA on simulated *E. coli* data, showing the approach successfully finds alignments between Rmaps simulated from overlapping genomic regions. Lastly, we demonstrate KOHDISTA is the only method that is capable of finding a significant number of high quality pairwise Rmap alignments for large eukaryote organisms in reasonable time. KOHDISTA is available at `https://github.com/mmuggli/KOHDISTA/`.

2012 ACM Subject Classification Applied computing → Bioinformatics

Keywords and phrases Optical mapping, index based data structures, FM-index, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.12

Funding MDM, SJP, and CB were funded by the National Science Foundation (1618814).

Acknowledgements The authors would like to thank Jouni Sirén for many thoughtful conversations regarding the GCSA.

1 Introduction

There is a current resurgence in generating diverse types of data, to be used alone or in concert with short read data, in order to overcome the limitations of short read data. Data from an optical mapping system [3, 4] is one such example and has itself become more practical with falling costs of high-throughput methods. For example, the current BioNano Genomics Irys System requires one week and $1,000 USD to produce the Rmap data for an average size eukaryote genome, whereas, it required $100,000 and six months in 2009[2]. These technological advances and the demonstrated utility of optical mapping in genome assembly [16, 21, 22, 2, 5] have driven several recent tool development efforts [20, 10, 13].

[1] SJP was also supported in part by the Academy of Finland via grant number 294143.
[2] `http://www.bionanogenomics.com/press-releases/bionano-genomics-launches-irys-a-novel-platform-for-complex-human-genome-analysis/`

Genome-wide optical maps are ordered high-resolution restriction maps that give the position of occurrence of restriction cut sites corresponding to one or more restriction enzymes. These genome-wide optical maps are assembled using an overlap-layout-consensus approach using raw optical map data, which are referred to as *Rmaps*. Hence, Rmaps are akin to reads in genome sequencing. To date, however, there is no efficient, non-proprietary method for finding pairwise alignments between Rmaps, which is the first step in assembling genome-wide maps.

Several existing methods are superficially applicable to Rmap pairwise alignments but all programs either struggle to scale to even moderate size genomes or require significant further adaptation to the problem. Several methods exhaustively evaluate all pairs of Rmaps using dynamic programming. One of these is the method of Valouev et al. [19], which is capable of solving the problem exactly but requires over 100,000 CPU hours to compute the alignments for rice [18]. The others are SOMA [15] and MalignerDP [13] which are designed only for semi-global alignments instead of overlap alignments, which are required for assembly.

Other methods reduce the number of map pairs to be individually considered by initially finding seed matches and then extending them through more intensive work. These include OMBlast [10], OPTIMA [20], and MalignerIX [13]. These, along with MalignerDP, were designed for a related alignment problem of aligning consensus data but cannot consistently find high quality Rmap pairwise alignments in reasonable time as we show later. This is unsurprising since these methods were designed for either already assembled optical maps or *in silico* digested sequence data for one of their inputs, both having a lower error rate than Rmap data.

Our contributions. In this paper, we present a fast, error-tolerant method for performing pairwise Rmap alignment that makes use of a novel FM-index based data structure. Although the FM-index can naturally be applied to short read alignment [11, 9], it is nontrivial to apply it to Rmap alignment. The difficulty arises from: (1) the abundance of missing or false cut sites, (2) the fragment sizes require inexact fragment-fragment matches (e.g. 1,547 bp and 1,503 bp represent the same fragment), (3) the Rmap sequence alphabet consists of all unique fragment sizes and is so extremely large (e.g., over 16,000 symbols for the goat genome). The second two challenges render inefficient the standard FM-index backward search algorithm, which excels at exact matching over small alphabets. The first (and most-notable) challenge requires a more complex index-based data structure be used to create an aligner that is robust for insertion and deletion of cut sites. To overcome the mismatch cut site challenge while still accommodating the other two, we develop KOHDISTA, an index-based Rmap alignment program that is capable of finding all pairwise alignments in large eukaryote organisms.

We first abstract the problem to that of approximate-path matching in a directed acyclic graph (DAG). The KOHDISTA method then indexes a set of Rmaps represented as a DAG, using a modified form of the *generalized compressed suffix array (GCSA)*, which is a variant of the FM-index developed by Siren et al. [17]. The principle insight of our work is that while GCSA is able to efficiently match all similar paths concurrently, it was designed for indexing variations observed in a collection of sequences. In contrast, our work indexes variations that are instead speculative, based on the Rmap error profile. Lastly, we demonstrate that challenges posed by the inexact fragment sizes and alphabet size can be overcome, specifically in the context of the GCSA, via careful use of a wavelet tree [7, 14].

We verify our approach on simulated *E. coli* Rmap data by showing that KOHDISTA achieves similar sensitivity and specificity to Valouev et al., and with more permissive alignment acceptance criteria 90% of Rmap pairs simulated from overlapping genomic regions.

We also show the utility of our approach on larger eukaryote genomes by demonstrating that existing published methods require more than 151 hours of CPU time to find all pairwise alignments in the plum Rmap data; whereas, KOHDISTA requires 31 hours. Thus, we present the first fully-indexed method capable of finding all match patterns in the pairwise Rmap alignment problem.

2 Background

Throughout we consider a string (or sequence) $S = S[1..n] = S[1]S[2]\dots S[n]$ of $|S| = n$ symbols drawn from the alphabet $[1..\sigma]$. For $i = 1, \dots, n$ we write $S[i..n]$ to denote the *suffix* of S of length $n - i + 1$, that is $S[i..n] = S[i]S[i+1]\dots S[n]$, and $S[1..i]$ to denote the *prefix* of S of length i. $S[i..j]$ is the *substring* $S[i]S[i+1]\dots S[j]$ of S that starts at position i and ends at j. Given a sequence $S[1,n]$ over an alphabet $\Sigma = \{1, \dots, \sigma\}$, a character $c \in \Sigma$, and integers i,j, $\mathsf{rank}_c(S,i)$ is the number of times that c appears in $S[1,i]$, and $\mathsf{select}_c(S,j)$ is the position of the j-th occurrence of c in S.

Overview of Optical Mapping. From a computer science viewpoint, restriction mapping (by optical or other means) can be seen as a process that takes in two sequences: a genome $\mathsf{A}[1,n]$ and a restriction enzyme's recognition sequence $\mathsf{B}[1,b]$, and produces an array (sequence) of integers C, the *genome restriction map*, which we define as follows. First define the array of integers $\mathsf{C}[1,m]$ where $\mathsf{C}[i] = j$ if and only if $\mathsf{A}[j..j+b] = \mathsf{B}$ is the ith occurrence of B in A. Then $\mathsf{R}[i] = (\mathsf{C}[i] - \mathsf{C}[i-1])$, with $\mathsf{R}[1] = \mathsf{C}[1] - 1$. In words, R contains the distance between occurrences of B in A. For example, if we let B be `act` and A = `atacttactggactactaaact` then we would have $\mathsf{C} = 3, 7, 12, 15, 20$ and $\mathsf{R} = 2, 4, 5, 3, 5$. In reality, R is a consensus sequence formed from millions of erroneous Rmap sequences. The optical mapping system produces millions of Rmaps for a single genome. It is performed on many cells of an organism and for each cell there are thousands of Rmaps (each at least 250 Kbp in length in publicly available data). The Rmaps are then assembled to produce a genome-wide optical map. Like the final R sequence, each Rmap is an array of lengths — or fragment sizes — between occurrences of B in A.

There are three types of errors that an Rmap (and hence with lower magnitude and frequency, also the consensus map) can contain: (1) missing and false cuts, which are caused by an enzyme not cleaving at a specific site, or by random breaks in the DNA molecule, respectively; (2) missing fragments that are caused by *desorption*, where small (< 1 Kbp) fragments are lost and so not detected by the imaging system; and (3) inaccuracy in the fragment size due to varying fluorescent dye adhesion to the DNA and other limitations of the imaging process. Continuing again with the example above where $\mathsf{R} = 2, 4, 5, 3, 5$ is the error-free Rmap: an example of an Rmap with the first type of error could be $\mathsf{R}' = 6, 5, 3, 5$ (the first cut site is missing so the fragment sizes 2, and 4 are summed to become 6 in R'); an example of a Rmap with the second type of error would be $\mathsf{R}'' = 2, 4, 3, 5$ (the third fragment is missing); and lastly, the third type of error could be illustrated by $\mathsf{R}''' = 2, 4, 7, 3, 5$ (the size of the third fragment is inaccurately given).

Frequency of Errors. In the optical mapping system, there is a 20% probability that a cut site is missed and a 0.15% probability of a false break per Kbp, i.e., error type (1) occurs in a fragment. Popular restiction enzymes in optical mapping experiments recognize a 6 bp sequence giving an expected cutting density of 1 per 4096 bp. At this cutting density, false breaks are less common than missing restriction sites (approx. $0.25 * .2 = .05$ for missing sites

vs. 0.0015 for false sites per bp). The inaccuracy of the fragment sizes, i.e, error type (3), follows a normal distribution with mean and variance assumed to be 0 bp and $\ell\sigma^2$ ($\sigma = .58$ kbp), respectively [19].

Suffix Arrays, BWT and Backward Search. The suffix array [12] SA_X (we drop subscripts when they are clear from the context) of a sequence X is an array $\mathsf{SA}[1..n]$ which contains a permutation of the integers $[1..n]$ such that $\mathsf{X}[\mathsf{SA}[1]..n] < \mathsf{X}[\mathsf{SA}[2]..n] < \cdots < \mathsf{X}[\mathsf{SA}[n]..n]$. In other words, $\mathsf{SA}[j] = i$ iff $\mathsf{X}[i..n]$ is the j^{th} suffix of X in lexicographic order. For a sequence Y, the Y-interval in the suffix array SA_X is the interval $\mathsf{SA}[s..e]$ that contains all suffixes having Y as a prefix. The Y-interval is a representation of the occurrences of Y in X. For a character c and a sequence Y, the computation of $c\mathsf{Y}$-interval from Y-interval is called a *left extension*.

The Burrows-Wheeler Transform $\mathsf{BWT}[1..n]$ is a permutation of X such that $\mathsf{BWT}[i] = \mathsf{X}[\mathsf{SA}[i] - 1]$ if $\mathsf{SA}[i] > 1$ and \$ otherwise [1]. We also define $\mathsf{LF}[i] = j$ iff $\mathsf{SA}[j] = \mathsf{SA}[i] - 1$, except when $\mathsf{SA}[i] = 1$, in which case $\mathsf{LF}[i] = I$, where $\mathsf{SA}[I] = n$. Ferragina and Manzini [6] linked BWT and SA in the following way. Let $\mathsf{C}[c]$, for symbol c, be the number of symbols in X lexicographically smaller than c. The function $\mathsf{rank}(\mathsf{X}, c, i)$, for sequence X, symbol c, and integer i, returns the number of occurrences of c in $\mathsf{X}[1..i]$. It is well known that $\mathsf{LF}[i] = \mathsf{C}[\mathsf{BWT}[i]] + \mathsf{rank}(\mathsf{BWT}, \mathsf{BWT}[i], i)$. Furthermore, we can compute the left extension using C and rank. If $\mathsf{SA}[s..e]$ is the Y-interval, then $\mathsf{SA}[\mathsf{C}[c] + \mathsf{rank}(\mathsf{BWT}, c, s), \mathsf{C}[c] + \mathsf{rank}(\mathsf{BWT}, c, e)]$ is the $c\mathsf{Y}$-interval. This is called *backward search* [6], and a data structure supporting it is called an *FM-index*.

To support rank queries in backward search, a data structure called a *wavelet tree* (see [7]) can be used. It occupies $n \log \sigma + o(n \log \sigma)$ bits of space and supports rank queries in $O(\log \sigma)$ time. Wavelet trees also support a variety of more complex queries on the underlying string efficiently (see, e.g. [7]). One such query we will use in this paper is to return the set X of distinct symbols occurring in $S[i, j]$, which takes $O(|X| \log \sigma)$ time.

3 The Pairwise Rmap Alignment Problem

Given a genome $\mathsf{A}[1, n]$ and a restriction enzyme's recognition sequence $\mathsf{B}[1, b]$, the optical mapping system produces Rmaps, which are arrays of lengths—or fragment sizes—between occurrences of B in A. The background section provides details on the optical mapping process. Producing Rmap data is an error prone process. Thus, three types of errors can occur: (1) missing and false cuts that delimit fragments; (2) missing fragments; and (3) inaccuracy in the fragment sizes. For example, let $\mathsf{R} = 2, 4, 5, 3, 5$ be an error-free Rmap, then an example of an Rmap with the first type of error could be $\mathsf{R}' = 6, 5, 3, 5$ (the first cut site is missing so the fragment sizes 2, and 4 are summed to become 6 in R'); an example of a Rmap with the second type of error would be $\mathsf{R}'' = 2, 4, 3, 5$ (the third fragment is missing); and lastly, the third type of error could be illustrated by $\mathsf{R}''' = 2, 4, 7, 3, 5$ (the size of the third fragment is inaccurately given)

The pairwise Rmap alignment problem aims to align one Rmap (the *query*) R_q against the set of all other Rmaps in the dataset (the *target*). We denote the target database as $\mathsf{R}_1 \ldots \mathsf{R}_n$, where each R_i is a sequence of m_i fragment sizes, i.e, $\mathsf{R}_i = [f_{i1}, .., f_{im_i}]$. An alignment between two Rmaps is a relation between them comprising groups of zero or more consecutive fragment sizes in one Rmap associated with groups of zero or more consecutive fragments in the other. For example, given $\mathsf{R}_i = [4, 5, 10, 9, 3]$ and $\mathsf{R}_j = [10, 9, 11]$ one possible alignment is $\{[4, 5], [10]\}, \{[10], [9]\}, \{[9], [11]\}, \{[3], []\}$. A group may contain more than one fragment

(e.g. [4, 5]) when the restriction site delimiting the fragments is absent in the corresponding group of the other Rmap (e.g [10]). This can occur if there is a false restriction site in one Rmap, or there is a missing restriction site in the other. Since we cannot tell from only two Rmaps which of these scenarios occurred, for the purpose of our remaining discussion it will be sufficient to consider only the scenario of missed (undigested) restriction sites.

4 Methods

We now describe the algorithm behind KOHDISTA. Three main insights enable our index-based aligner for Rmap data: 1) abstraction of the alignment problem to a finite automaton; 2) use of the GCSA for storing and querying the automaton; and 3) modification of backward search to use a wavelet tree in specific ways to account for the Rmap error profile.

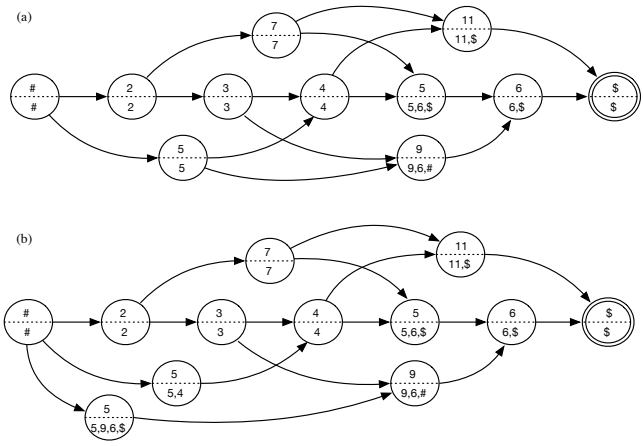
4.1 Finite Automaton

Continuing with the example in the background section, we want to align $\mathsf{R}' = 6, 5, 3, 5$ to $\mathsf{R}''' = 2, 4, 7, 3, 5$ and vice versa. To accomplish this we cast the Rmap alignment problem to that of matching paths in a finite automaton. A finite automaton is a directed, labeled graph that defines a *language*, or a specific set of sequences composed of vertex labels. A sequence is recognized by an automaton if it contains a matching path: a consecutive sequence of vertex labels equal to the sequence. We represent the target Rmaps as an automaton and the query as a path in this context.

The automaton for our target Rmaps can be constructed as follows. First concatenate the $\mathsf{R}_1 \ldots \mathsf{R}_n$ together into a single sequence with each Rmap separated by a special symbol which will not match any query symbol. Let R^* denote this concatenated sequence. Hence, $\mathsf{R}^* = [f_{11}, .., f_{1m_1}, \ldots, f_{n1}, .., f_{nm_n}]$. Then, construct an initial finite automaton $\mathsf{A} = (V, E)$ for R^* by creating a set of vertices $v_1^i..v_m^i$, one vertex labeled with each fragment length and edges connecting them.Also, introduce to A a *starting vertex* v_1 labeled with # and a *final vertex* v_f labeled with the character \$. All other vertices in A are labeled with integral values. This initial set of vertices and edges is called the *backbone*. The backbone by itself is only sufficient for finding alignments with no missing cut sites in the query. The backbone of an automaton constructed for a set containing R' and R'' would be $\#, 6, 5, 3, 5, 999, 2, 4, 3, 5\$$, using 999 as an unmatchable value. Next, extra vertices ("skip vertices") and extra edges are added to A to allow for the automaton to accept all valid queries. Figure 1a(a) illustrates the construction of A for a single Rmap with fragment sizes $2, 3, 4, 5, 6$.

4.1.1 Skip Vertices and Skip Edges

We introduce extra vertices labeled with *compound fragments* to allow missing cut sites (first type of error) to be taken into account in querying the target Rmaps. We refer to these as *skip vertices* as they provide alternative path segments which skip past two or more backbone vertices. Thus, we add a *skip vertex* to A for every $o + 1$ length run of consecutive vertices in the backbone where $1 < o < order$ and $order$ is the maximum number of consecutive missed cut sites to be accommodated. First order skip vertices are each labeled with the sum of two consecutive backbone vertices. Second order skip vertices are each labeled with the sum of three consecutive backbone vertices. The vertex labeled with 7 connecting 2 and 5 in 1a(a) is an example of a skip vertex. Likewise, $5, 9, 11$ are other skip vertices.



(a) An example automaton for an Rmap with fragment size sequence $2, 3, 4, 5, 6$. The top half of vertices contains the label, which models a fragment size in Kbp. The common prefixes of all suffixes spellable from a vertex is written in the bottom half. Note that there is no ordering of vertices such that all their corresponding suffixes are in lexicographic order; the leftmost vertex labelled with "5" spells suffixes beginning "5,4,..." as well as the suffix "5,9,6,$" while the rightmost 5 spells the suffix "5,6,$". (b) shows the prefix sorted automaton corresponding to the one in (a). The leftmost vertex 5 has been duplicated and the outgoing edges of the previous version have been divided between the new replacement instances. This also divides the suffixes spellable from the prior version. Now the three 5 vertices can be ordered based on their common prefixes as ["5,4,...","5,6,$", "5,9,6, $"].

	BWT	M	F
$	6	1	1
	11		0
2	#	1	1
		0	
3	2	1	1
		0	
4	3	1	1
	5	0	0
5,4	#	1	1
5,6,$	4	1	1
	7		0
5,9,6,$	#	1	1
6,$	5	1	1
	9		0
7	2	1	1
		0	
9,6,$	3	1	1
	5		0
11,$	4	1	1
	7		0
#	$	1	1
		0	
		0	

(b) Table listing the three arrays storing the automaton in memory: BWT, M, and F. Each row in the table delimits elements associated with a particular vertex.

Figure 1 Example automata and corresponding memory representation.

Finally, we add *skip edges* which provide paths around vertices with small labels in the backbone. These allow a query with a missing fragment to still match.[3] Hence, the addition of skip edges allow for desorption (the second type of error) to be taken into account in querying the target Rmaps.

4.2 Generalized Compressed Suffix Array

We index the automaton with the GCSA [17] for efficient storage and path querying. The GCSA is a generalization of the FM-index for automata and we will explain the GCSA by drawing on the definition of the (more widely known) FM-index. As stated in the background section, the FM-index is based on the deep relationship between the SA and the BWT data structures of the input string X. The BWT of an input string is formed by sorting all characters of the string by the lexicographic order of the suffix immediately following each character. The main properties the FM-index exploits in order to perform queries efficiently are a) $\mathsf{BWT}[i] = \mathsf{X}[\mathsf{SA}[i] - 1]$; and b) given that $\mathsf{SA}[i] = j$, and $\mathsf{C}[c]$ gives the position of the first suffix in SA prefixed with character c, then using small auxiliary data structures we can

[3] Different smallness thresholds for query and target bias toward this scenario, avoiding backtracking in the search.

quickly determine $k = \mathsf{C}[\mathsf{BWT}[i]] + \mathsf{rank}(\mathsf{BWT}, \mathsf{BWT}[i], i)$, such that $\mathsf{SA}[k] = j - 1$. The first of these properties is simply the definition of the BWT. The second is, because the symbols of X occur in the same order in both the single character prefixes in the suffix array and in the BWT, given a set of sorted suffixes, prepending the same character onto each suffix does not change their order. Thus, if we consider all the suffixes in a range of SA which are preceded by the same symbol c, that subset will appear in the same relative order in (another part of) SA: as a contiguous subinterval of the interval that contains all the suffixes beginning with c. Thus by knowing where a symbol's run begins in the SA and the rank of an instance of that symbol, we can identify the SA position beginning with that instance from its position in BWT. A rank data structure over the BWT thus constitutes a sufficient compressed index of the suffix array needed for traversal.

To generalize the FM-index to automata (from strings), we need to efficiently store the vertices and edges in a manner such that the FM-index properties still hold, allowing the GCSA to support queries efficiently. An FM-index's compressed suffix array for a string S encodes a relationship between each suffix S and its left extension. Hence, this suffix array can be generalized to edges in a graph that represent a relationship between vertices. The compressed suffix array for a string is a special case where the vertices are labeled with the string's symbols in a non-branching path.

4.2.1 Prefix-sorted Automata

Just as backward search for strings is linked to suffix sorting, backward searching in the BWT of the automaton requires us to be able to sort the vertices (and a special set of the paths) of the automaton in a particular way. In [17] this property is called *prefix-sortedness*. Let $A = (V, E)$ be a finite automaton, let $v_{|V|}$ denote its terminal vertex, and let $v \in V$ be a vertex. We say v is *prefix-sorted* by prefix $p(v)$ if the labels of all paths from v to $v_{|V|}$ share a common prefix $p(v)$, and no path from any other vertex $u \neq v$ to $v_{|V|}$ has $p(v)$ as a prefix of its label. Automaton A is prefix-sorted if all vertices are prefix-sorted. See Figure 1a for an example of a non-prefix sorted automaton and a prefix sorted automaton. A non-prefix sorted automaton can be made prefix sorted through a process of duplicating vertices and their incoming edges but dividing their outgoing edges between the new instances (see [17]).

Clearly the prefixes $p(v)$ allow us to sort the vertices of a prefix-sorted automaton into lexicographical order. Moreover, if we consider the list of outgoing edges (u, v), sorted by pairs $(p(u), p(v))$, they are also sorted by the sequences $\ell(u)p(v)$, where $\ell(u)$ denotes the label of vertex u. This (dual sortedness) property allows backward searching to work over the list of vertex labels (sorted by $p(v)$) in the same way that is does for the symbols of a string ordered by their following suffixes in normal backward search for strings.

Each vertex has a set of one or more preceding vertices and therefore, a set of predecessor labels in the automaton. These predecessor label sets are concatenated to form the BWT. The sets are concatenated in the order defined by the above mentioned lexicographic ordering of the vertices. Each element in BWT then denotes an edge in the automaton. Another array of bits, F, marks a '1' for the first element of BWT corresponding to a vertex and a '0' for all subsequent elements in that set. Thus, the predecessor labels, and hence the associated edges, for a vertex with rank r are $\mathsf{BWT}[\mathsf{select}(r)..\mathsf{select}(r+1)]$. Another array, M, stores the out degree of each vertex and allows the set of vertex ranks associated with a BWT interval to be found using rank() queries.

4.3 Exact Matching: GCSA Backward Search

Exact matching with the GCSA is similar to the standard FM-index backward search algorithm. As outlined in the background section, FM-index backward search proceeds by finding a succession of lexicographic ranges that progressively match longer and longer suffixes of the query string, starting from the rightmost symbol of the query. The search maintains two items — a lexicographic range and an index into the query string — and the property that the path prefix associated with the lexicographic range is equal to the suffix of the query marked by the query index. Initially, the query index is at the rightmost symbol and the range is $[1..n]$ since every path prefix matches the empty suffix. The search continues using GCSA's backward search step function, which takes as parameters the next symbol (to the left) in the query (i.e. fragment size in R_q) and the current range, and returns a new range. The query index is advanced leftward after each backward search step. In theory, since the current range corresponds to a consecutive range in the BWT, the backward search could use select() queries on a bit vector F to determine all the edges adjacent to a given vertex and then two FM-index LF() queries are applied to the limits of the current range to obtain the new one. GCSA's implementation uses one succinct bit vector per alphabet symbol to encode which symbols precede a given vertex instead of F. Finally, this new range, which corresponds to a set of edges, is mapped back to a set of vertices using rank() on the M bit vector.

4.4 Inexact Matching: GCSA Backward Search Using a Wavelet Tree

We modified GCSA backward search in the following ways: (1) we used a wavelet tree to allow efficient retrieval of substitution candidates; (2) we modified the search process to combine consecutive query fragments into compound fragments so as to match fragments in R^* missing the interposing restriction site; and (3) we introduced backtracking, in order to both try size substitution candidates as well as various combinations of compound fragments. These modifications are further detailed below.

First, in order to accommodate possible errors in fragment size, we determine a set, D, of candidate fragment sizes that are similar to the next fragment of R_q to be matched in the query. These candidates are determined by enumerating the distinct symbols in the currently active backward-search range of the BWT[4] using the wavelet tree algorithm of Gagie et al. [7]. This method was proposed by Muggli et al. [14] for use with an FM-index but was not directly applicable to the originally proposed implementation of GCSA. This is because some of GCSA's theoretical constructs (i.e. F) were substituted in implementation for efficiency reasons. In order to apply the aforementioned wavelet tree method, we thus resurrect the previously theoretical only bit array F (which we encode succinctly) as well as symbol array BWT (which we encoded with a wavelet tree) into KOHDISTA using the SDSL-Lite library by Gog et al. [8].

To accommodate possible restriction sites that are present in the query Rmap but absent in target Rmaps, we generate compound fragments (i.e. new symbols) by summing pairs and triples of consecutive query fragment size and then querying the wavelet tree for substitutions of these compound fragments. This summing of multiple consecutive fragments is complementary to the skip vertices in the target automaton and accommodates missed restriction sites in the target, just as the skip vertices accommodate missed sites in the query.

[4] Recall that this active range, when applied to a lexicographic range, represents the suffixes whose prefixes are the matched portion of the query, while the same range of the BWT contains possible extension symbols.

Lastly, since there may be multiple match candidates in the BWT interval of R^* for a compound fragment generated from R_q and multiple compound fragments generated at a given position in R_q, we employ the common practice of adding backtracking to backward search (as is done, for example in the works of Li et al. and Langmead et al.). This is so that each candidate size returned to the search algorithm from the wavelet tree is evaluated; i.e., for a given compound fragment size f generated from R_q, every possible candidate fragment size, f', that can be found in R^* in the range $f-t \ldots f+t$ and in the interval $s \ldots e$ (of the BWT of R^*) for some tolerance t is used as a substitute in the backward search.

5 Results and Discussion

We evaluated KOHDISTA against the other available optical map alignment software. Our experiments measured runtime, peak memory, and alignment quality on simulated *E. coli* Rmaps and experimentally generated plum Rmaps. All experiments were performed on Intel Xeon computers with $\geq$ 16 GB RAM running 64-bit Linux.

5.1 Performance on Simulated E.coli Rmap Data

To verify the correctness of our method, we simulated a read set from a 4.6 Mbp *E. coli* reference genome as follows: we started with 1,400 copies of the genome, and then generated 40 random loci within each. These loci form the ends of molecules that would undergo digestion. Molecules smaller than 250 Kbp were discarded leaving 272 molecules with a combined length equating to 35x coverage depth. The cleavage sites for the XhoI enzyme were then identified within each of these simulated molecules. We removed 20% of these at random from each simulated molecule to model partial digestion. Finally, normally distributed noise was added to each fragment with a standard deviation of .58 kb per 1 kb of the fragment. Simulated molecule pairs having 16 common conserved digestion sites become the "ground truth"[5] data for testing our method with the others. Although a molecule would align to itself, these are not included in the ground truth set. This method of simulation was based on the *E. coli* statistics given by Valouev et al. [18] and resulting in a molecule length distribution as observed in publicly available Rmap data from OpGen, Inc.

Most of the tools were designed for less noisy data but in theory could address all the data error types required. For tools with tunable parameters, we tried aligning the *E. coli* Rmaps with combinations of parameters for each method related to its alignment score thresholds and error model parameters. We used parameterization giving results similar to those for the default parameters of Valouev et al.'s method to the extent such parameters did not significantly increasing each tool's runtime. These same parameterization were used in the next section on plum data.

Even with tuning, we were unable to obtain pairwise alignments on *E. coli* for two methods. We found OPTIMA only produced self alignments with its recommended overlap protocol and report its resource use in Table 1. For MalignerIX, even when we relaxed the parameters to account for the greater sizing error and mismatch cut site frequency, it was also only able to find self alignments. This is expected as by design it only allows missing sites in one sequence in order to run faster. Thus no further testing was performed with

[5] Due to repeats in the restriction map, and apparent repeats at the resolution attainable through optical measurement, some alignments beyond these are expected.

Table 1 Performance on simulated *E. coli* dataset. KOHDISTA (lax) demonstrates that our indexing and search method is capable of finding the majority of ground truth alignments when the search is pruned to the more relaxed thresholds of $\chi^2 < .02$, Binom. $< .5$.

Method	Time	Memory	Align-ments	Recall	Precision
KOHDISTA	20 s.	19.0 MB	907	702 / 4,305 (16%)	702 / 907 (77%)
KOHDISTA (lax)	373 s.	18.3 MB	8,545	3,925 / 4,305 (91%)	3,925 / 8,545 (46%)
Valouev et al.	148 s.	4.0 MB	742	699 / 4,305 (16%)	699 / 742 (94%)
MalignerDP	47 s.	6.0 MB	1,959	1,296 / 4,305 (30%)	1,296 / 1959 (66%)
OMBlast	116 s.	2,078 MB	1,008	806 / 4,305 (19%)	806 / 1008 (80%)
RefAligner	31 s.	81.2 MB	992	958 / 4,305 (22%)	948 / 992 (97%)
MalignerIX	4 s.	6.0 MB	0	0 / 4,305 (0%)	0 / 0 (N/A)
OPTIMA	455 s.	10,756.5 MB	0	0 / 4,305 (0%)	0 / 0 (N/A)

Table 2 Performance on Plum.

Method	Time	Memory	Alignments
KOHDISTA	31 hours	7.4 GB	16,109,151
Valouev et al.	678 hours	60 MB	6,387
MalignerDP	214 hours	784 MB	568,744
OMBlast	151 hours	12.3 GB	424,730
RefAligner	90 hours	374 MB	10,039

MalignerIX or OPTIMA. We did not test SOMA [15] as earlier investigation indicate it would not scale to larger genomes [14]. We omit TWIN [14] as it needs all cut sites to match.

Results on *E. coli* are presented in Table 1. KOHDISTA uses χ^2 and binomial CDF thresholds to prune the backtracking search when deciding whether to extend alignments to progressively longer alignments. More permissive match criteria, using higher thresholds, allows more Rmaps to be reached in the search and thus to be considered aligned, but it also results in less aggressive pruning in the search, thus lengthening runtime. As an example, note that when KOHDISTA was configured with a much relaxed CDF threshold of .5 and a binomial CDF threshold of .7, it found 3,925 of the 4,305 (91%) ground truth alignments, but slowed down considerably. This illustrates the index and algorithm's capability in handling all error types.

5.2 Performance on Plum Rmap Data

The Beijing Forestry University and other institutes assembled the first plum (*Prunus mume*) genome using short reads and optical mapping data from OpGen Inc. We test the various available alignment methods on the 139,281 plum Rmaps from June 2011 available in the GigaScience repository. These Rmaps were created with the BamHI enzyme and have a coverage depth of 135x of the 280 Mbp genome. For the plum dataset, we ran all the methods which approach the statistical performance of the Valouev et al. method when measured on *E. coli*. Thus, we omitted MalignerIX and OPTIMA because they had 0% recall and precision on *E. coli*. Our results on this plum dataset are summarized in Table 2.

KOHDISTA was the fastest and obtained more alignments than the competing methods. When configured with a χ^2 CDF threshold of .02, it took 31 hours of CPU time to test all Rmaps for pairwise alignments in the plum Rmap data. This represents a 21x speed-up over the 678 hours taken by the exhaustive Valouev et al. method. The other non-proprietary

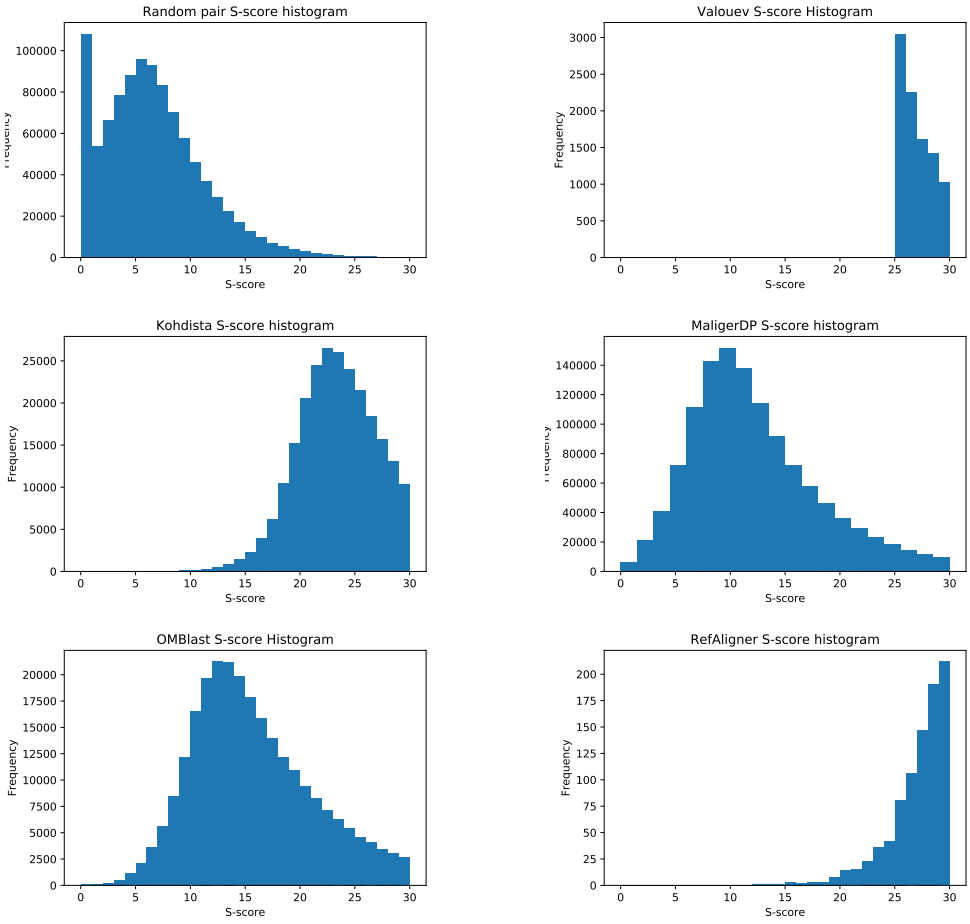


Figure 2 All alignments found on plum were realigned using Valouev et al.'s dynamic programming method. Their method finds the optimal alignment using a function balancing size agreement and cut site agreement known as an s-score. (a) The s-score distribution for random pairs. (b) The Valouev et al. software considers any pair with an s-score > 25 to be aligned. (c) KOHDISTA alignments tend to have significantly higher s-scores than random. (d) MalignerDP alignments tend to have slightly higher s-scores than random. (e) OMBlast alignments tend to have higher s-scores than random. (f) BioNano's commercial RefAligner method alignments tends to have a significantly higher s-scores than random.

methods, MalignerDP and OMBlast, took 214 hours and 151 hours, respectively. These results represent a 6.9x and 4.8x speed-up over MalignerDP and OMBlast. All methods used less than 13 GB of RAM and thus, were considered practical from a memory perspective.

To measure the quality of the alignments, we scored each pairwise alignment using the scoring scheme of Valouev et al. and present histograms of these alignment scores in Figure 2. For comparison, we also scored and present the histogram for random pairs of Rmaps. The Valouev et al. method produces very few but high-scoring alignments and although it could theoretically be altered to produce a larger number of alignments, the running time makes this prospect impractical (678 hours). Although KOHDISTA and RefAligner produce high-quality alignments, RefAligner produced very few alignments (10,039) and required almost 5x more time to do so. OMBlast and Maligner required significantly more time and produced significantly lower quality alignments.

6 Conclusion

In this paper, we demonstrate how finding pairwise alignments in Rmap data can be modelled as approximate-path matching in a directed acyclic graph, and combining the GCSA with the wavelet tree results in an index-based data structure for solving this problem. We implement this method and present results comparing KOHDISTA with competing methods. By demonstrating results on both simulated *E. coli* Rmap data and real plum Rmaps, we show that KOHDISTA is capable of detecting high scoring alignments in efficient time. In particular, KOHDISTA detected the largest number of alignments in 31 hours. RefAligner, a proprietary method, produced very few high scoring alignments (10,039) and requires almost 5x more time to do so. OMBlast and Maligner required significantly more time and produced significantly lower quality alignments. The Valouev et al. method produced high scoring alignments but required more than 21x time to do.

References

1 M. Burrows and D.J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

2 S. Chamala et al. Assembly and validation of the genome of the nonmodel basal angiosperm *amborella*. *Science*, 342(6165):1516–1517, 2013.

3 Aston Christopher and Schwartz David C. Optical mapping in genomic analysis, 2006. doi:10.1002/9780470027318.a1421.

4 E.T. Dimalanta, A. Lim, R. Runnheim, C. Lamers, C. Churas, D.K. Forrest, J.J. de Pablo, M.D. Graham, S.N. Coppersmith, S. Goldstein, and D.C. Schwartz. A microfluidic system for large DNA molecule arrays. *Analytical Chemistry*, 76(18):5293–5301, 2004.

5 Y. Dong et al. Sequencing and automated whole-genome optical mapping of the genome of a domestic goat (*capra hircus*). *Nature Biotechnology*, 31(2):136–141, 2013.

6 P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.

7 T. Gagie, G. Navarro, and S. J. Puglisi. New algorithms on wavelet trees and applications to information retrieval. *Theoretical Computer Science*, 426-427:25–41, 2012.

8 Simon Gog et al. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014. doi:10.1007/978-3-319-07959-2_28.

9 B. Langmead et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

10 Alden King-Yung Leung et al. Omblast: alignment tool for optical mapping using a seed-and-extend approach. *Bioinformatics*, page btw620, 2016.

11 H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–60, 2009.

12 U. Manber and G. W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Scientific Computing*, 22(5):935–948, 1993.

13 L. M. Mendelowitz et al. Maligner: a fast ordered restriction map aligner. *Bioinformatics*, 32(7):1016–1022, 2016.

14 M.D. Muggli, S.J. Puglisi, and C. Boucher. Efficient indexed alignment of contigs to optical maps. In *Proc. of WABI*, pages 68–81, 2014.

15 N. Nagarajan, T. D Read, and M. Pop. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, 24(10):1229–1235, 2008.

16 S. Reslewic et al. Whole-genome shotgun optical mapping of *Rhodospirillum Rubrum*. *Applied Environmental Microbiology*, 71(9):5511–5522, 2005.

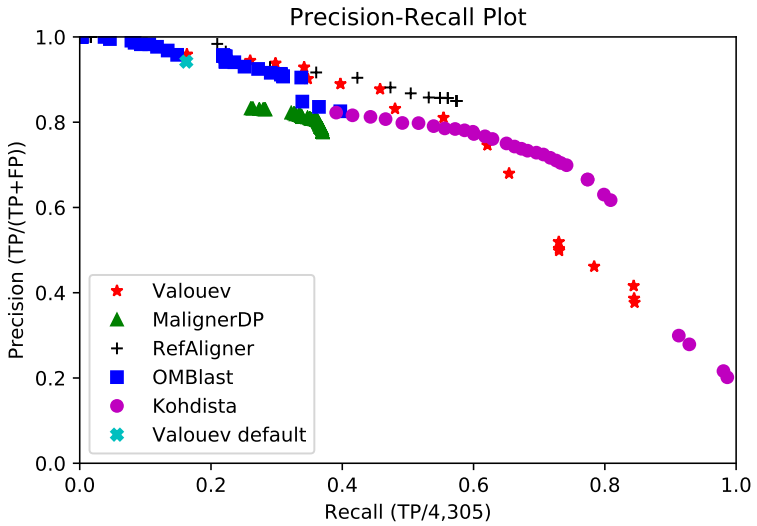


Figure 3 Precision-Recall plot of successful methods on simulated *E. coli*.

17 J. Sirén et al. Indexing graphs for path queries with applications in genome research. *IEEE/ACM TCBB*, 11(2):375–388, 2014.

18 A. Valouev et al. An algorithm for assembly of ordered restriction maps from single DNA molecules. *Proc Natl Acad Sci*, 103(43):15770–15775, 2006.

19 A. Valouev et al. Alignment of optical maps. *J. Comp Bio*, 13(2):442–462, 2006.

20 Davide Verzotto et al. Optima: Sensitive and accurate whole-genome alignment of error-prone genomic maps by combinatorial indexing and technology-agnostic statistical analysis. *GigaScience*, 5(1):2, 2016.

21 S. Zhou et al. A whole-genome shotgun optical map of *Yersinia pestis* strain KIM. *Applied and Environmental Microbiology*, 68(12):6321–6331, 2002.

22 S. Zhou et al. Shotgun optical mapping of the entire *Leishmania major* Friedlin genome. *Molecular and Biochemical Parasitology*, 138(1):97–106, 2004.

A Practical Indexing Considerations

A.1 Pruning the Search

Alignments are found by incrementally extending candidate partial alignments (paths in the automaton) to longer partial alignments by choosing one of several compatible extension matches (adjacent vertices to the end of a path in the automaton). To perform this search efficiently, we prune the search by computing the χ^2 and binomial CDF statistics of the partial matches and use thresholds to ensure reasonable size agreement of the matched compound fragments, and the frequency of putative missing cut sites. These values alter the precision and recall as well as runtime. The statistical performance tradeoff of KOHDISTA and competing methods is shown in Figure 3.

A.1.1 Size Agreement

We use the Chi-square CDF statistic to assess size agreement. This assumes the fragment size errors are independent, normally distributed events. For each pair of matched compound

fragments in a partial alignment, we take the mean between of the two as the assumed true length and compute the expected standard deviation using this mean. Each compound fragment deviates from the assumed true value by half the distance between them. These two deviation values contribute two degrees of freedom to the Chi-square calculation. Thus each deviation is normalized by dividing by the expected standard deviation, these are squared, and summed across all compound fragments to generate the χ^2 statistic. We use the standard χ^2 CDF function to compute the area under the curve of the probability mass function up to this χ^2 statistic, which gives the probability two Rmap segments from common genomic origin would have a χ^2 statistic no more extreme than observed. This probability is compared to KOHDISTA's chi-squared-cdf-thresh and if smaller, the candidate compound fragment is assumed to be a reasonable match and the search continues.

A.1.2 Cut Site Error Frequency

We use the Binomial CDF statistic to assess the probability of the number of cut site errors in a partial alignment. This assumes missing cut site errors are independent, Bernoulli processes events. We account for all the putatively conserved cut sites on the boundaries and those delimiting compound fragments in both partially aligned Rmaps plus twice the number of missed sites as the number of Bernoulli trials. We use the standard binomial CDF function to compute the sum of the probability density function up to the number of non-conserved cut sites in a candidate match. Like the size agreement calculation above, this gives the probability two Rmaps of common genomic origin would have the number of non-conserved sites seen or fewer in the candidate partial alignment under consideration. This is compared to the binom-cdf-thresh to decide whether to consider extensions to the given candidate partial alignment. Thus, given a set of Rmaps and input parameters ρ_L and ρ_U, we produce the set of all Rmap alignments that have a chi-square CDF statistic less than ρ_U and a binomial CDF statistic less than ρ_L. Both of these are subject to the additional constraint of a maximum consecutive missed restriction site run between aligned sites of δ and a minimum aligned site set cardinality of 16.

A.1.2.1 Pruning Queries

One side effect of summing consecutive fragments in both the search algorithm and the target data structure is that several successive search steps with agreeing fragment sizes will also have agreeing sums of those successive fragments. In this scenario, proceeding deeper in the search space will result in wasted effort. To reduce this risk, we maintain a table of scores obtained when reaching a particular lexicographic range and query cursor pair. We only proceed with the search past this point when either the point has never been reached before, or has only been reached before with inferior scores.

A.1.2.2 Wavelet Tree Cutoff

The wavelet tree allows efficiently finding the set of vertex labels that are predecessors of the vertices in the current match interval intersected with the set of vertex labels that would be compatible with the next compound fragment to be matched in the query. However, when the match interval is sufficiently small (< 750) it is faster to scan the vertices in BWT directly.

A.1.2.3 Quantization

The alphabet of fragment sizes can be large considering all the measured fragments from multiple copies of the genome. This can cause an extremely large branching factor for the initial symbol and first few extensions in the search. To improve the efficiency of the search, the fragment sizes are initially quantized, thus reducing the size of the effective alphabet and the number of substitution candidates under consideration at each point in the search. Quantization also increases the number of identical path segments across the indexed graph which allows a greater amount of candidate matches to be evaluated in parallel because they all fall into the same BWT interval during the search. This does, however, introduce some quantization error into the fragment sizes, but the bin size is chosen to keep this small in comparison to the sizing error.

A.1.2.4 Example Traversal

A partial search for a query Rmap [3 kb, 7 kb, 6 kb] in Figure 1a and Table 1b given an error model with a constant 1 kb sizing error would proceed with steps: 1. Start with the semi-open interval matching the empty string [0..12). 2. A wavelet tree query on BWT would indicate the set of symbols {5, 6, 7} is the intersection of two sets: 1.) The set of symbols that would all be valid left extensions of the (currently empty) match string and 2.) The set of size appropriate symbols that match our next query symbol (i.e. 6 kb, working from the right end of our query) in light of the expected sizing error (i.e. 6kb +/- 1 kb). 3. We would then do a GCSA backward search step on the first value in the set (5) which would yield the new interval [4..7). This new interval denotes only nodes where each node's common prefix is compatible with the spelling of our current backward traversal path through the automaton (i.e. our short path of just [5] does not contradict any path spellable from any of the three nodes denoted in the match interval). 4. A wavelet tree query on the BWT for this interval for values 7 kb +/- 1 kb would return the set of symbols 7. 5. Another backward search step would yield the new interval [8..9). At this point our traversal path would be [7, 5] (denoted as a left extension of a forward path that we are building by traversing the graph backward). The common prefix of each node (only one node here) in our match interval (i.e. [7 kb]) is compatible with the path [7, 5]. This process would continue until backward search returns no match interval or our scoring model indicates our repeatedly left extended path has grown too divergent from our query. At this point backtracking would occur to find other matches (e.g. at some point we would backward search using the value 6 kb instead of the 5 kb obtained in step 2.)

A.1.2.5 Parameters Used

We tried OPTIMA with both "p-value" and "score" scoring and the allMaps option and report the higher sensitivity "score" setting. We followed the OPTIMA-Overlap protocol of splitting Rmaps into k-mers, each containing 12 fragments as suggested in [20]. For OMBlast, we adjusted parameters maxclusteritem, match, fpp, fnp, meas, minclusterscore, and minconf. For MalignerDP, we adjusted parameters max-misses, miss-penalty, sd-rate, min-sd, and max-miss-rate and additionally filtered the results by alignment score. Though unpublished, for comparison we also include the proprietary RefAligner software from BioNano. For RefAligner we adjusted parameters FP, FN, sd, sf, A, and S. For Kohdista, we adjusted parameters chi-squared-cdf-thresh and binom-cdf-thresh. For Valouev, we adjusted score_thresh and t_score_thresh variables in the source. In Table 1 we report statistical and computational performance for each method.

OMBlast was configured with parameters meas=3000, minconf=0.09, minmatch=15 and the rest left at defaults. RefAligner was run with parameters FP=0.15, sd=0.6, sf=0.2, sr=0.0, se=0.0, A=15, S=22 and the rest left at deafults. MalignerDP was configured with parameters ref-max-misses=2, query-miss-penalty=3, query-max-miss-rate=0.5, min-sd=1500, and the rest left at defaults.

The software of Valouev et al. was run with default parameters except we reduced the maximum compound fragment length (their δ parameter) from 6 fragments to 3. We observed the software of Valouev et al. rarely included alignments containing more than two missed restriction sites in a compound fragment.

Spalter: A Meta Machine Learning Approach to Distinguish True DNA Variants from Sequencing Artefacts

Till Hartmann
Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen, University Hospital Essen, 45122 Essen, Germany
Till.Hartmann@uni-due.de
https://orcid.org/0000-0002-6993-347X

Sven Rahmann
Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen, University Hospital Essen, 45122 Essen, Germany
Bioinformatics, Computer Science XI, TU Dortmund, Dortmund, Germany
Sven.Rahmann@uni-due.de
https://orcid.org/0000-0002-8536-6065

Abstract

Being able to distinguish between true DNA variants and technical sequencing artefacts is a fundamental task in whole genome, exome or targeted gene analysis. Variant calling tools provide diagnostic parameters, such as strand bias or an aggregated overall quality for each called variant, to help users make an informed choice about which variants to accept or discard. Having several such quality indicators poses a problem for the users of variant callers because they need to set or adjust thresholds for each such indicator. Alternatively, machine learning methods can be used to train a classifier based on these indicators. This approach needs large sets of labeled training data, which is not easily available.

The new approach presented here relies on the idea that a true DNA variant exists independently of technical features of the read in which it appears (e.g. base quality, strand, position in the read). Therefore the *nucleotide separability* classification problem – predicting the nucleotide state of each read in a given pileup based on technical features only – should be near impossible to solve for true variants. Nucleotide separability, i.e. achievable classification accuracy, can either be used to distinguish between true variants and technical artefacts directly, using a thresholding approach, or it can be used as a meta-feature to train a separability-based classifier. This article explores both possibilities with promising results, showing accuracies around 90%.

2012 ACM Subject Classification Applied computing → Sequencing and genotyping technologies, Computing methodologies → Unsupervised learning, Computing methodologies → Supervised learning by classification

Keywords and phrases variant calling, sequencing error, technical artefact, meta machine learning, classification

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.13

1 Introduction

Apparent differences between a reference genome and a sequenced sample may either be due to biological variance or to technical artefacts (e.g. caused by flawed library preparation or sequencing machine bias). For downstream analysis, e.g., finding disease-related genes, or personal disease risk assessment, it is important to be able to distinguish between these two

kinds of causes. Therefore, variant callers, such as the one in GATK [3], include measures, such as strand bias or overall variant quality, to assess the likelihood of a technical sequencing or processing artefact, and conversely, to provide a measure of confidence for each detected variant. For this, variant callers typically rely on assumptions about error distributions or require prior or expert knowledge [1]. Since these assumptions may not hold for every sequencing and processing pipeline or no prior knowledge may be available, we develop a different approach. For simplicity, our presentation focuses on single nucleotide variants (SNVs). In principle, it can be applied to longer variants (e.g. short indels) at the expense of a more complex technical implementation.

The idea is that a true DNA variant exists independently of technical features of the read where it appears, e.g. base quality, position in the read, mapping quality or strand of the read. Therefore the classification problem to predict the nucleotide state of each read in a given pileup based on only technical features should be unlearnable for a true variant. In other words, high classification accuracy on the *nucleotide separability* classification problem on a pileup suggests that the called variant is a technical artefact. Separability alone can be used to decide between a true DNA variant and a technical artefact. This yields a method that does not need labeled training data with known true variants. If ground truth information about which pileups exhibit true variants (and which exhibit technical artefacts) is available, it is possible to train a machine learning model on the separability values, resulting in a meta machine learning task. We here explore both possibilities.

Our work, called `spalter`, differs from related approaches that also use machine learning techniques to classify pileups, like SNPSVM [4] or DeepVariant [5], in that it does not necessarily need labeled training data. `Spalter` employs classification methods for computing separability values, which *can*, but do not need to be used for training another classifier using labeled target data (true variant vs. technical artefact). If such is not available, a thresholding rule can be applied on separability at the cost of accuracy. In contrast, SNPSVM learns only the latter classifier in conjunction with 'hand-crafted' features (such as 'base quality mean' or 'read position mean'). DeepVariant encodes information in RGBA images (in newer versions, data is encoded in multichannel tensors instead), which are then used to train a convolutional neural network (CNN). Training the CNN is computationally expensive, especially without dedicated graphic processing units (GPUs) or tensor processing units (TPUs). `Spalter`, in contrast, is neither restricted to a single model architecture nor does it require extensive CPU or memory resources.

2 Idea

For an arbitrary site in the reference sequence, let $\mathcal{B}$ be the multiset of bases in that site's pileup, and let s be the base in the reference sequence at that site. If some bases in $\mathcal{B}$ disagree with s, i.e. there is a certain (minimum) fraction of bases which are not equal to s, we ask whether this is due to technical artefacts or biological variation.

However, there is more information available for each base of a pileup than just the nucleotide represented, and we gather and store the purely *technical* information that should be independent of the presence or absence of a biological variant, such as the *base quality* of a base (assigned by the sequencer), the *strand* of the read, or whether the read is the first or second of a read pair.

If, for a given pileup, it is possible to *separate* bases that match the respective reference base from those that do not by making use of the aforementioned technical features and simple classifiers, we are inclined to believe that a technical artefact is present (depending

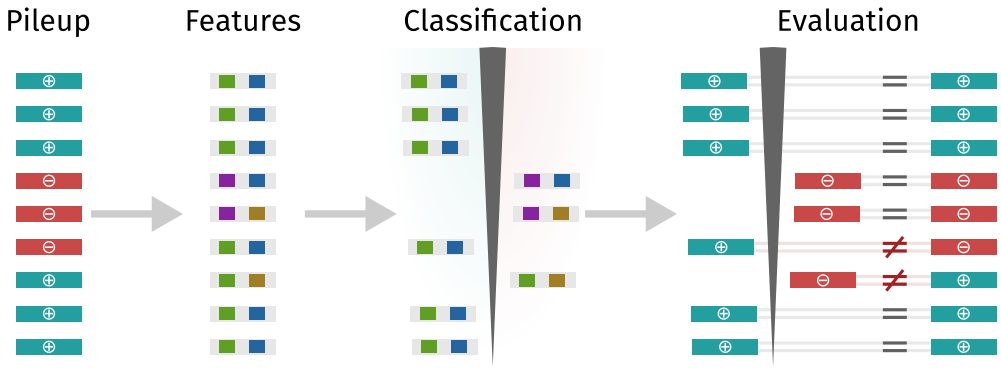


Figure 1 Computing separability profiles. For a given pileup, labels ⊕ (base matches reference base) and ⊖ (base does not match reference base) are determined for every base (rectangle) in the pileup (stack of rectangles). Also, for each base a technical feature vector (grey rectangle with colored squares, one square for each feature) is derived from its read and accompanying auxiliary data (such as base quality or read strand). Using these labels and feature vectors, a simple classification model is trained and its training accuracy is evaluated and interpreted as *separability*.

on the degree to which separation was successful).

The reasoning behind this assumption is that the sequencing process is consistent regardless of whether samples contain SNVs or not. This implies that it should not be possible to tell reference from alternative bases by looking at technical features only. Hence, if it is indeed possible to tell them apart, a pattern was recognized which is most likely explained by technical biases during some stage of the sequencing process.

For a given pileup's bases $\mathcal{B}$, we derive base labels $\oplus$ if the base matches the reference base s and $\ominus$ if it does not. We then train a classifier to predict the base label only from the technical features (which should work badly for true biological variants). The *training accuracy* of this classifier is called *separability* of $\mathcal{B}$ (Figure 1).

A low separability suggests that there is a true variant in $\mathcal{B}$. As a first step, the separability of a pileup can thus be used *directly* for decision making (true variant or technical artefact), or, in a further step, as a feature for training a classifier based on separability. The first step does not need labeled training examples (in the sense of manually annotated 'true SNV' and 'technical artefact' labels; of course, a supervised classification problem is solved based on known nucleotide status for each read). The second step does need labeled training examples and uses (meta-)features (separability) from the classification problem of the first step. We hence call `spalter` a pseudo-supervised meta machine learning approach.

3 Method

To disambiguate between the different kinds of features used throughout the paper, features associated with bases and reads (such as base quality or read strand) are called *technical* features, while features constructed as explained in subsection 3.1 are called *meta features* or *separability profile*.

Algorithm 1 Building a separability profile for a single pileup `p`, using a set of classifiers $\mathcal{C}$ and a set of feature combinations $\mathcal{F}$. A $\mathcal{C} \times \mathcal{F}$ matrix `S` with separability values is returned.

```
1  function separability_profile(p, 𝒞, ℱ):
2      S = empty real 𝒞 × ℱ matrix
3      ℓ = [base == reference_base for base in p.bases]
4      f = extract_features(p)          # technical features
5      for (C, F) in 𝒞 × ℱ:             # for each combination (each F ∈ 2^f)
6          M = C.train(F,ℓ)             # train model M
7          ℓ′ = M.predict(F)            # predict labels ℓ′
8          s = rel_accuracy(ℓ, ℓ′)      # evaluate training accuracy (see text)
9          S[C,F] = s                   # store s in meta feature matrix
10     return S
```

3.1 Computing the separability profile

Since the idea is to discern between SNVs and technical artefacts on the assumption that they behave differently with respect to separability, the first step is to build a separability profile[1] for each pileup. Algorithm 1 outlines the routine for building a separability profile for a single pileup `p` for a given set $\mathcal{C}$ is a set of classifiers and a set $\mathcal{F}$ of feature combinations from a basic set f of technical features.

We assume that the given pileup `p` has a given minimum alternative allele fraction (default: ≥ 0.1) and a given minimum coverage (default: ≥ 10)[2]. A label vector ℓ is built by comparing every base in the pileup to the reference base at the pileup's position (line 3). By default, the following four technical features are extracted *for each base* in the pileup (line 4), resulting in a technical feature matrix of dimension number of reads in `p` times number of features:

1. Read pair index (binary): Is the read containing the base the first or the second of its read pair?
2. Read strand (binary): Is the read on the forward or the reverse strand?
3. Neighbour base quality (continuous): Average base quality of the respective base and its direct neighbours (from the same read).
4. End proximity (continuous): Distance of the base to either end of its read, transformed to be in $[0, 1]$. The closer the base to the center of its read, the lower the end proximity and vice versa.

In order to build a separability *profile*, we train one model (lines 6–9) for each combination of classifier and feature subset (line 5) in order to obtain information about the difficulty of separation (a) with a limited set of features and (b) with different classifiers. Note that increasing the number of technical features or increasing the number of classifiers, while increasing the level of detail of separability profiles, will not change the general appearance and behaviour of separability profiles; they will always exhibit a gradual increase of accuracy with both increasing number of technical features used for training and classifier complexity, as can be seen in Figure 2. Calculating separability is straightforward: After training a model (line 6), its prediction (line 7) is used to calculate *relative* training accuracy $s = \frac{a-p}{1-p}$, where a is the model's accuracy (fraction of correct predictions) and p is the accuracy of a

[1] Initially, we assumed that a single separability value might suffice; however, the resulting accuracy was not satisfactory.

[2] Since we are training classifiers, these defaults ensure that there are at least 10 examples available for training, albeit with heavily skewed class sizes $(1 : 9)$.

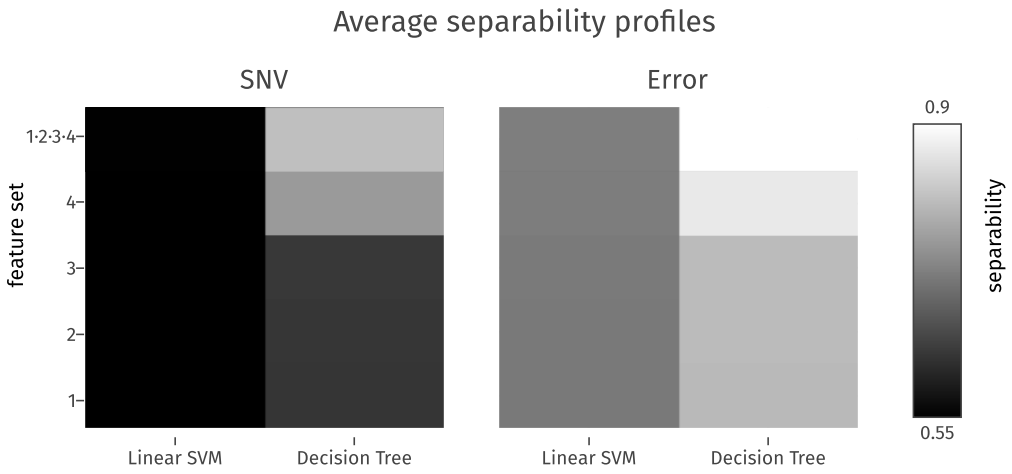


Figure 2 Average separability profiles for 2327321 SNV and 5618509 erroneous sites, using 2 classifiers and $4 + 1$ different feature combinations (numbers correspond to those of features listed in subsection 3.1). Each cell corresponds to the training accuracy of one classifier (x-axis) trained with a certain subset of features (y-axis). On average, separability is higher for erroneous sites than for SNV sites.

'largest class' classifier (line 8), i.e. an accuracy of p is trivial to achieve. Finally, the resulting relative accuracy value – called *separability* – is stored in the separability profile `S` (line 9).

3.2 Distinguishing between errors and SNVs

Having built a separability profile for each pileup, the task is to utilise these to distinguish between sites with technical artefacts and sites with true SNVs.

One way to do this is to use a thresholding approach: Summarise separability profiles (per pileup) for example by averaging the values to a single mean separability value and apply a threshold to the averages. Any site whose mean separability is below the threshold is deemed a likely true variant site, while values above the threshold imply a site with technical sequencing artefacts (the easier it is to separate disagreeing bases in a pileup, the more likely becomes a technical artefact).

Another approach lies in using separability profiles as features to train a classification model; this necessitates labeled data, i.e. information about whether a pileup/site is deemed a true SNV site or not.

Both approaches have been implemented in `spalter`. For the first approach (classification on mean separability values with a single threshold), the threshold defaults to 0.8; for the second approach (classification on separability profiles with an arbitrary supervised classifier), the classifier defaults to a random forest consisting of 19 trees with a maximum depth of 4 (but can be set to an arbitrary supervised classifier). We provide pre-trained default models, which have been evaluated as described in section 4.

4 Evaluation

In order to evaluate `spalter`, we used datasets from CHM-eval/syndip [2], a benchmark specifically designed for small variant callers. Among other things, it consists of the *de*

Table 1 Different separability thresholds for different bins of read depths lead to an overall increase in accuracy compared to the accuracy achieved using a single global threshold.

read depth	20–24	25–29	30–34	35–39	40–44	45–49	50–54	55–59
separability threshold	0.2	0.81	0.82	0.82	0.82	0.82	0.81	0.80
accuracy	92.9%	89.6%	91.8%	92.0%	94.7%	95.1%	96.0%	95.8%

novo assembly of PacBio sequences of two different complete hydatidiform mole (CHM) cell lines (i.e. completely homozygous) as well as Illumina HiSeq-X10 reads of a 1:1 mixture of the DNA of the two cell lines. We applied different methods to the CHM-eval dataset with known SNV sites from `full.37m.vcf.gz` (indels excluded). We also used the NA12878 dataset from Genome In A Bottle to be able to compare spalter with SNPSVM. Note that we could not successfully run SNPSVM on either dataset[3], which is why we use the results given in [4] instead, where a dataset was obtained based on the NA12878 exome (i.e. in-house sequencing on an Illumina HiSeq 2000, variants for training determined using GATK's best practices guidelines and filtering these with data from the 1000 Genomes project).

Separability profiles were computed using different subsets of the technical features (every single feature and all features) listed in subsection 3.1. The classifiers used for the nucleotide separability problem were logistic regression (using the stochastic gradient descent implementation in Rust, available from the `rustlearn` crate[4]), linear support vector machines, radial basis function support vector machines with default parameters and decision trees with a maximum depth of 4 (always using their `rustlearn` implementations).

In a first step, we use *only decisions based on thresholded average separability values*. Average separability values are computed as the average over the different feature combinations and the decision tree classifier as shown in Figure 2 (right column of panels). With a threshold of 0.8 (which is not an optimised threshold), an accuracy of 89% is obtained for the CHM-eval dataset, while the accuracy for the GIAB dataset is 87.2%. For CHM-eval, in detail, we discover 85.2% of the true SNVs, and 91.7% of our SNV calls are correct; we discover 93.6% of the true technical artefacts, and 88.4% of our artefact calls are correct with this threshold. For GIAB, we discover 65.3% of the true SNVs, and 95.3% of our SNV calls are correct; we discover 98.4% of the true technical artefacts, and 84.8% of our artefact calls are correct with this threshold.

Since SNPSVM did not finish successfully on either dataset, the comparison between the recall and precision values for SNPSVM reported in [4] and the values for spalter reported above is not a strong one, especially considering that the number of variants examined is roughly $4 \cdot 10^4$ for SNPSVM and $3 \cdot 10^6$ for spalter. Still, performance on the CHM-eval dataset is comparable to SNPSVM's performance, which achieved comparable recall with higher precision. We note, however, that SNPSVM needs labeled training data to achieve this performance, while our results are based on simple separability thresholding and an educated guess for the threshold. Note that it may well be possible to estimate a better threshold from the resulting average separability distribution automatically.

The accuracy can be improved if different (optimal) thresholds are chosen based on the pileup's read depth (coverage) and labeled training data, as shown in Table 1.

Similarly, higher accuracies can be obtained if separability is used as a feature in a meta-classification task: The classifiers used for the decision problem (true variant / technical

[3] This is probably due to a 0/1 indexing error in its source.
[4] see `https://maciejkula.github.io/rustlearn/`

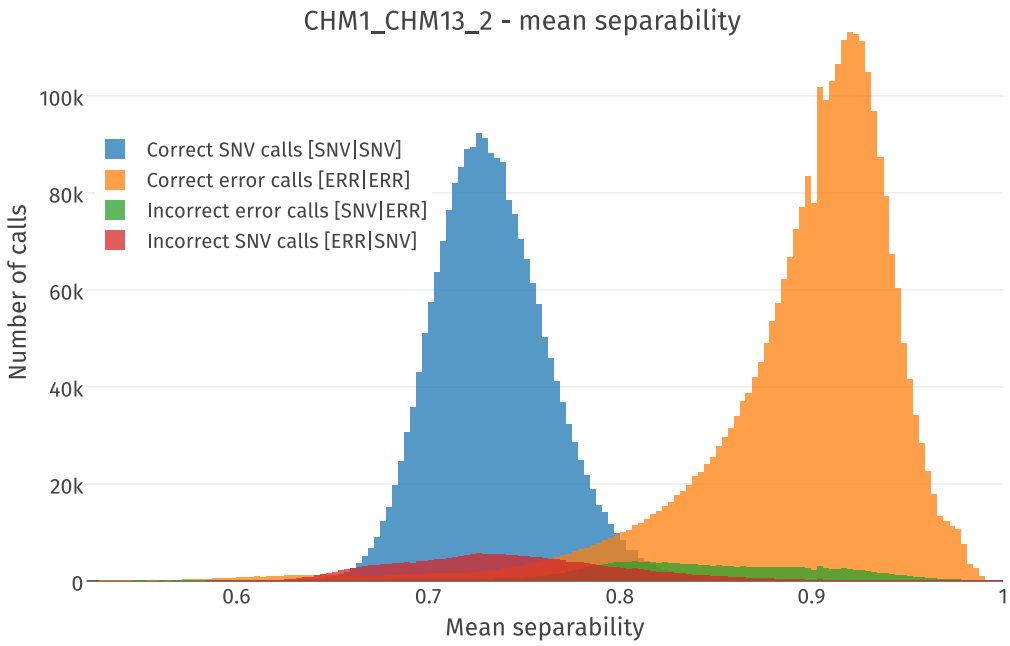


Figure 3 Histograms of mean separability values for dataset CHM1_CHM13_2. Correctly called SNVs (blue) and errors (orange) are clearly distinct, with the latter exhibiting higher mean separability values. Incorrectly called SNV sites (red) have a mean separability distribution with its mean close to that of the distribution of correct SNV calls, while incorrectly called errors (green) occur most often around the intersection of correct SNV and error calls.

artefact) were random forests and decision trees (with maximum depth 4). Training sample size was varied between 10^i for $i \in \{4, 5, 6\}$, where 10^4 data points correspond to roughly 0.001% of available data. We found that both training and testing accuracy on CHM-eval were extremely consistent (for all tested combinations between 87.4% (lowest minus one standard deviation) and 92.9% (highest plus one standard deviation)), with the mean at 91.5%, which is higher than the overall accuracy using simple thresholding.

We conclude that (a) the introduced meta features are indeed useful for training classification models, and (b) a tiny fraction of the data is sufficient to train a suitable classification model.

To explicitly show that the claim 'error likelihood increases with separability' holds, we examined the empirical distributions of mean separability values as shown in Figure 3. Indeed, sites that exhibit technical artefacts tend to have higher mean separability values than true SNV sites.

5 Discussion

Using meta features (separability profiles) to train classifiers to distinguish technical artefacts from true SNVs leads to accurate models, especially given the fact that only abstract separability information is used for training. When using random labels for training instead, accuracy is close to 50%, which indicates that separability profiles do not facilitate overfitting. It remains to be determined if models derived from separability profiles are 'portable', i.e.

can be applied to different datasets successfully (without adjustments to the models).

The advantage of this approach is that separability as a single abstract notion can be used to distinguish between errors and SNVs, while the *only* assumption made is that errors grow increasingly more likely with increasing separability. This also entails that it is possible to increase classification accuracy by either making additional assumptions or manually adding features (such as alternative allele fraction). Here we wanted to show that separability in itself is a powerful feature and did not further explore this possibility.

Also, the extraction of meta features (computation of separability profiles) presented here is an elaborate way to automatically derive *constant size* feature matrices for any kind of data with varying item sizes (here: differing numbers of reads between pileups) and suspected intrinsic labels (here: base matches / does not match reference base), i.e. the approach is not restricted to this particular application in bioinformatics.

Another interesting avenue to explore is that, while there are (combinatorially) many different ways to assign labels to bases in a pileup, the intrinsic one (base matches / does not match the reference base) is the most 'natural'. So it may be viable to try to find errors not site-wise but read-wise by comparing separability regarding different label vectors (e.g. ones where a single base has its label flipped).

The source code of `spalter` can be obtained from `https://bitbucket.org/tillux/spalter/`.

References

1 Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*, 2012.

2 Heng Li, Jonathan M Bloom, Yossi Farjoun, Mark Fleharty, Laura D Gauthier, Benjamin Neale, and Daniel MacArthur. New synthetic-diploid benchmark for accurate variant calling evaluation. *bioRxiv*, 223297, 2017.

3 Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A. DePristo. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, 2010. `doi:10.1101/gr.107524.110`.

4 Brendan D O'Fallon, Whitney Wooderchak-Donahue, and David K Crockett. A support vector machine for identification of single-nucleotide polymorphisms from next-generation sequencing data. *Bioinformatics*, 29(11):1361–1366, 2013.

5 Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T. Afshar, Sam S. Gross, Lizzie Dorfman, Cory Y. McLean, and Mark A. DePristo. Creating a universal SNP and small indel variant caller with deep neural networks. *bioRxiv*, 2018. `doi:10.1101/092890`.

Essential Simplices in Persistent Homology and Subtle Admixture Detection

Saugata Basu[1]
Department of Mathematics, Purdue University, West Lafayette, IN 47906, USA
sbasu@math.purdue.edu
https://orcid.org/0000-0002-2441-0915

Filippo Utro
Computational Biology Center, IBM T. J. Watson Research,
Yorktown Heights, NY 10598, USA
futro@us.ibm.com
https://orcid.org/0000-0003-3226-7642

Laxmi Parida
Computational Biology Center, IBM T. J. Watson Research,
Yorktown Heights, NY 10598, USA
parida@us.ibm.com
https://orcid.org/0000-0002-7872-5074

Abstract

We introduce a robust mathematical definition of the notion of essential elements in a basis of the homology space and prove that these elements are unique. Next we give a novel visualization of the essential elements of the basis of the homology space through a rainfall-like plot (RFL). This plot is data-centric, i.e., is associated with the individual samples of the data, as opposed to the structure-centric barcodes of persistent homology. The proof-of-concept was tested on data generated by SimRA that simulates different admixture scenarios. We show that the barcode analysis can be used not just to detect the presence of admixture but also estimate the number of admixed populations. We also demonstrate that data-centric RFL plots have the potential to further disentangle the common history into admixture events and relative timing of the events, even in very complex scenarios.

2012 ACM Subject Classification Applied computing → Life and medical sciences

Keywords and phrases population admixture, topological data analysis, persistent homology, population evolution

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.14

1 Introduction

A fascinating way to study the relationship between multiple individuals is to understand their potential common history implicated by the genetic signatures of the individuals [6, 7]. If the individuals were bacteria, then their common history is bound to be captured by a tree; while sexually reproducing organisms such as humans have the common history represented as directed acyclic graphs (DAG). A layer of complexity is introduced to the common history through populations. While the common history of individuals *within* a population is captured by a DAG, populations can admix i.e., individuals of two populations can interbreed, at a specific time or time-interval, leaving behind yet a different kind of

[1] The work was partially supported by NSF grant DMS-1620271.

imprint on the genome of the individuals [4, 11]. See Fig 1 for an illustration: A, B, C, and D are extant populations whose common (evolutionary) history is as shown by the structure on the left where the *scaffold* is shown as the DAG with dashed lines. The implicit direction is assumed to flow downwards: the past on the top and the present at the bottom leaf nodes. Note that the scaffold is not a tree; hence there are two admixed populations: C and D. But A and B are not admixed. The figure on the left shows the *macro-level* structure while two pictures on the right show the *micro-level* structure, i.e., transmission of genetic material over time (across generations) in the individuals. The interested reader is directed to [9] for a detailed exposition.

It is easy to appreciate that the DAG due to sexual reproduction within the population (micro-level) is intricately entangled with the DAG due to the admixing populations (macro-level). However, the data available for study is the genome of the individuals of the extant populations only and not necessarily other intermediate (possibly extinct) populations. In fact, in [9] it was shown that given a mixture of populations (possibly without population labels) persistent homology can be used to detect if any admixing event had occurred in their common history. In this paper we extend the analysis to identifying multiple (not just presence or absence of at least one) admixture events. In other words, the question then is whether it is possible to tease apart some essentials of the macro structures (such as the 2 cycles in the structure on the left in the figure) from a collection of individuals which have randomly been drawn from multiple extant populations.

In our experiments we use SimRA [1] to simulate the population data to define the gold-truth. Then we employ persistent homology to address our questions. In literature, barcode diagrams [2, 5, 8] have been widely used to succintly represent persistent homology (see Definition 2 below): in each dimension, the start point of a bar marks the birth and the end point marks the death of a non-zero homology class in that dimension. To detect admixture, not only do we utilize the arrangement (pattern) of the bars in the different dimension, but we also need to associate them to the individuals or the data points.

Each bar in the bar code diagram represents a non-zero homology class – and each such class can be represented by a cycle i.e. by a linear combination of simplices with vanishing boundary. However, this representation is not unique for several reasons – for example, a homology class is defined only up to boundaries (cycles which bound one higher dimensional linear combinations of simplices). We would like to associate to each bar, a unique set of simplices (whose vertices represent individuals). The problem of associating a particular cycle to a bar is an interesting problem in its own right and several approaches has been taken by researchers leading to difficult optimizing problems (for example, computing the shortest length cycle in a homology class). In this paper, we take a different approach. We give a mathematical definition of a well defined (non-empty) set of essential simplices associated to each bar of the persistence diagram that we compute (see Definition 9 below). In this way we resolve the inherent ambiguity of choosing a representative cycle for each homology class. We believe that this new notion of essential simplices can have implications for a host of applications going beyond the one described in this paper.

In our experiments, we observe that the clustering of the irreducible cycles in the barcode plot capture the admixing events in the population history. This is reinforced by using essential simplices, that further segregates the individuals.

Roadmap. In the next section we give the mathematical underpinnings for the essential simplices and in Section 3 we apply this to simulated population data. In Section 3.1 we describe the visualization of the essential simplices and summarize the results.

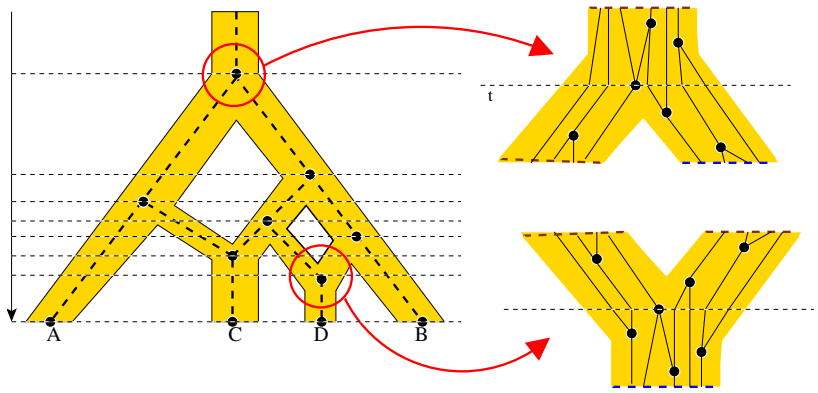


Figure 1 Macro-level structure on the left while two micro-level structures are shown on the right for four populations A, B, C, and D. See text for further details.

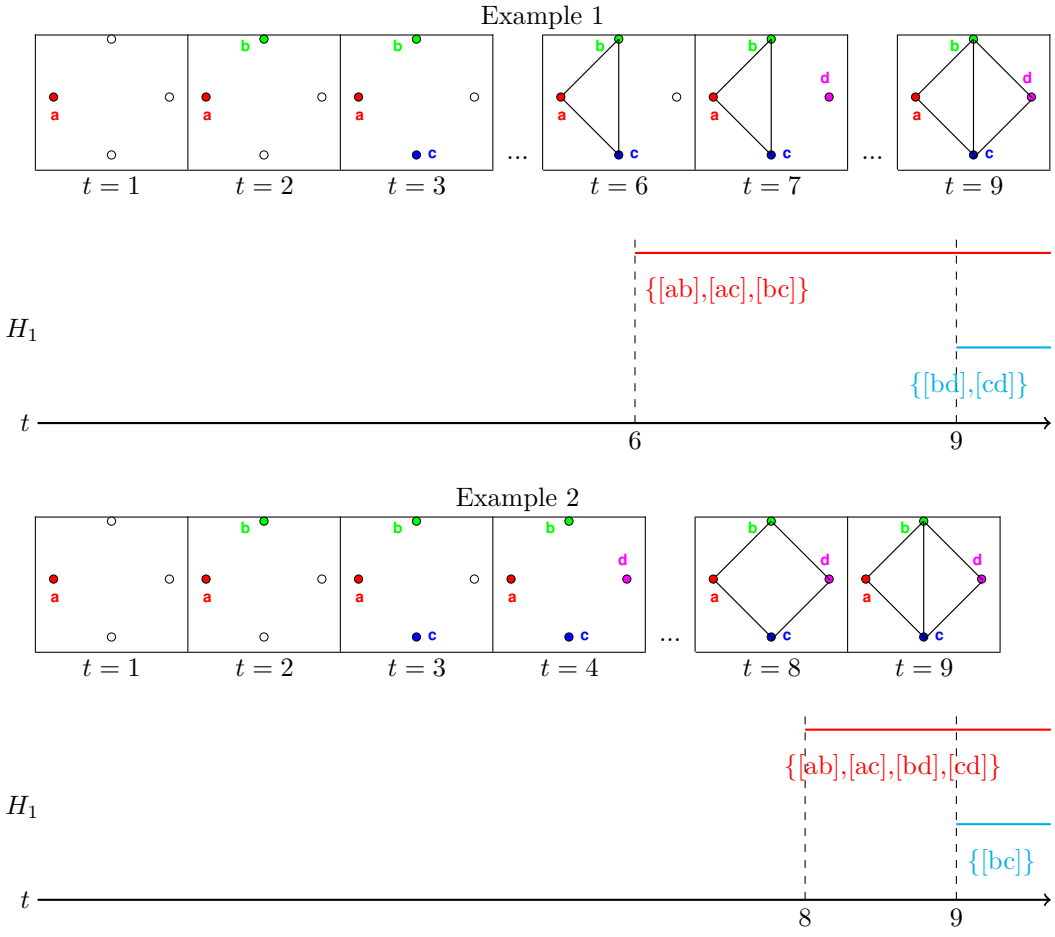


Figure 2 Bar codes for H_1 and their associated essential simplices for the filtrations in Example 10.

2 Filtrations of finite simplicial complexes, persistent homology, and essential simplices

One important problem in persistent homology theory is to associate to a homology class an actual cycle representing the homology class. Several methods have been developed to choose such a class optimally (with respect to various cost functions). In this paper we choose a different approach. Given a filtered simplicial complex, where the filtration satisfies a certain property (see Definition 3 and Proposition 5 below), we associate to each bar of the bar diagram of the p-th persistent homology of a filtration of a simplicial complex a non-empty set of p-simplices of the simplicial complex in a canonical way – which we call the set of *essential* simplices. Roughly speaking, a p-simplex σ is essential for a bar in the diagram of the p-th persistent homology if and only if σ appears with a non-zero coefficient in any linear combination of p-simplices representing the homology cycle corresponding to the bar. The rest of this section will make this intuitive notion rigorous.

We consider homology with coefficients in $\mathbb{Z}_2$, and which we omit from the notation. We also assume familiarity with basic simplicial homology theory, and denote by $H_*(K)$ the homology groups of a simplicial complex K with coefficients in $\mathbb{Z}_2$. Abusing notation a bit, we will denote by the same letter K the set of simplices of a simplicial complex K, and for $p \geq 0$, we will denote by $K^{(p)}$ the set of simplices of dimension p in K, so that

$$K = \cup_{p \geq 0} K^{(p)}.$$

We now recall the basic definitions pertaining to persistent homology.

Let $\mathcal{F}$ denote a filtration of a finite simplicial complex K given by : $\emptyset = \cdots = K_{-1} = K_0 \subset K_1 \subset \cdots \subset X_s \subset K_{s+1} \subset \cdots \subset K_N = K_{N+1} = \cdots = K$. Here each K_i is a subcomplex of K.

▶ **Notation 1.** *For $s \leq t$, we let $i_n^{s,t} : H_n(K_s) \longrightarrow H_n(K_t)$, denote the homomorphism induced by the inclusion $K_s \hookrightarrow K_t$.*

With the same notation as in the previous section we define:

▶ **Definition 2.** [2] For each triple (n, s, t) with $s \leq t$ the corresponding *persistent homology group*, $H_n^{s,t}(\mathcal{F})$ is defined by

$$H_n^{s,t}(\mathcal{F}) \quad = \quad \mathrm{Im}(i_n^{s,t}).$$

Note that $H_n^{s,t}(\mathcal{F}) \subset H_n(K_t)$, and $H_n^{s,s}(\mathcal{F}) = H_n(K_s)$.

We will consider only a special kind of filtration on simplicial complexes which are induced by orderings on the simplices of the complex satisfying the following property.

▶ **Definition 3** (Admissible ordering)**.** Let K be a finite simplicial complex. We call a total ordering $<$ (of the simplices) of K to be admissible if it satisfies the condition that $\sigma \prec \tau \Rightarrow \sigma < \tau$ for all simplices $\sigma, \tau \in K$. We will denote by $\mathrm{rk}_< : K \to [0, \mathrm{card}(K) - 1]$ the rank function of the ordering $<$.

▶ Remark 4. Note that the rank function, $\mathrm{rk}_<$, corresponding to an admissible ordering $<$ of the simplices of a simplicial complex K, is a discrete Morse function on K in the sense of Forman [3], for which every simplex is a critical simplex (in the sense of discrete Morse theory).

▶ **Proposition 5.** *Let K be a finite simplicial complex, and $<$ an admissible ordering of K. For $s \in [0, \mathrm{card}(K) - 1]$ let $K_s = \{\sigma \in K \mid \mathrm{rk}_<(\sigma) \leq s\}$. Then, $K_0 \subset K_1 \subset \cdots \subset K_{\mathrm{card}(k)-1} = K$ is a filtration of simplicial complexes. (We extend as usual the above filtration by setting $K_i = \emptyset$ for $i < 0$, and $K_j = K$ for all $j \geq \mathrm{card}(K)$, will refer to this filtration as the one induced by the ordering $<$.)*

Proof. The proof is easy and omitted. ◀

▶ **Proposition 6.** *Let $<$ be an admissible ordering of a finite simplicial complex K and let $\mathcal{F}$ denote the induced filtration of K. Then, for each $s, 0 \leq s \leq \text{card}(K) - 1$ and $p \geq 0$,* $\dim H_p(K_{s+1})/i_p^{s,s+1}(H_p(K_s)) \leq 1$.

Proof. Note that the rank function $\text{rk}_<$ is a discrete Morse function for which every simplex is critical (cf. Remark 4). The proposition is a consequence of the basic results of discrete Morse theory. ◀

▶ Remark 7. As a consequence of Proposition 6, we have that for each $p \geq 0$, and $s \geq 0$, the bar diagram for the p-dimensional persistent homology corresponding to an admissible filtration has at most one bar starting at time s.

Moreover, if $\sigma \in K^{(p)}$ with $\text{rk}_<(\sigma) = s$, then if there exists a cycle of the form $\sigma + \sum_{\tau \neq \sigma, \text{rk}_<(\tau) < \text{rk}_<(\sigma)} n_\tau \cdot \tau, \in Z_p(K_s)$, then and $H_p(K_s)/i_*^{s-1,s}(H_p(K_s)) \neq 0$, and this cycle represents the unique non-zero class in $H_p(K_s)/i_*^{s-1,s}(H_p(K_s)) \neq 0$.

▶ **Notation 8.** *For any filtration $\mathcal{F}$ and $p \geq 0$, we will denote by $\text{Bar}_p(\mathcal{F})$ to be the set of pairs (s,t) where each pair (s,t) corresponds to a bar starting at time s and ending at time t, in the bar diagram of the p-dimensional persistent homology of $\mathcal{F}$. For $c = (s,t) \in \text{Bar}_p(\mathcal{F})$, we denote $s(c) = s$.*

We are now in a position to define the set of essential simplices associated to a bar of a filtration induced by an admissible ordering.

▶ **Definition 9** (Essential simplices)**.** Let $\mathcal{F}$ be a filtration of a finite simplicial complex K induced by an admissible ordering. Suppose that for some $s \geq 0$, $H_p(K_s)/i_*^{s-1,s}(H_p(K_s)) \neq 0$, and suppose that $\dim H_p(K_s)/i_*^{s-1,s}(H_p(K_{s-1})) = 1$. Then, there is a unique bar $c \in \text{Bar}_p(\mathcal{F})$, with $s(c) = s$, in the bar diagram of $\mathcal{F}$. We call a p-simplex σ_0 to be *essential* with respect to c, if σ_0 satisfies the following condition.

For all $z = \sum_\sigma n_\sigma \cdot \sigma \in Z_p(K_s)$ (where the sum if taken over all p-simplices σ in the complex K_s), such that the image of z in $H_p(K_s)/i_p^{s-1,s}(H_p(K_{s-1}))$ under the canonical homomorphism is not zero, the coefficient n_{σ_0} of σ_0 is not equal to 0.

We will denote the set of essential simplices corresponding to c by Σ_c.

Before proceeding further we consider some examples.

▶ **Example 10** (Example of filtrations induced by admissible orderings and essential simplices)**.** Consider a graph on 4 vertices labelled a, b, c, d with 5 edges $[ab], [ac], [bc], [bd], [cd]$ (see Figure 2). Let $<$ be the admissible order

$$[a] < [b] < [c] < [ab] < [ac] < [bc] < [d] < [bd] < [cd].$$

Consder the bar diagram of the 1-dimensional homology for the induced filtration. It has two bars, $c = (6, \infty), c' = (9, \infty)$. It is easy to check that

$$\begin{aligned} \Sigma_c &= \{[ab], [ac], [bc]\}, \\ \Sigma_{c'} &= \{[bd], [cd]\}. \end{aligned}$$

Now consider the same graph but with a different admissible ordering. Let $<'$ be the order defined by

$$[a] <' [b] <' [c] <' [d] <' [ab] <' [ac] <' [bd] <' [cd] <' [bc].$$

The bar diagram of the 1-dimensional homology for the induced filtration again has two bars, $d = (8, \infty), d' = (9, \infty)$, and in this case we have

$$\begin{aligned} \Sigma_d &= \{[ab], [ac], [bd], [cd]\}, \\ \Sigma_{d'} &= \{[bc]\}. \end{aligned}$$

▶ Remark 11. As discussed earlier the set of essential simplices associated to a bar in the bar diagram corresponding to the filtration induced by an admissible ordering intuitively consists of simplices that must be present in any cycle representation of the homology cycle being born at that moment (the start time of the bar). In applications, this set of simplices can thus be considered as essential for the existence of the bar.

▶ **Theorem 12.** *Let $\mathcal{F}$ be a filtration induced by a perfect function, $p \geq 0$ and $c \in \mathrm{Bar}_p(\mathcal{F})$. Suppose that $z = \sum_{\sigma \in \Sigma} n_\sigma \cdot \sigma \in Z_p(K_s)$ (where the sum if taken over all p-simplices σ in the complex K_s), is such that its image in $H_p(K_s)/i_*^{s-1,s}(H_p(K_{s-1}))$ under the canonical homomorphism is not zero, and $n_\sigma \neq 0$ for each $\sigma \in \Sigma$.*

Then,

$$\Sigma_c = \Sigma \setminus \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'}. \tag{1}$$

Proof. Let $\sigma \in \Sigma_c$. We prove that $\sigma \in \Sigma \setminus \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'}$. By the defining property of Σ_c, it is clear that $\sigma \in \Sigma$. Now suppose that $\sigma \in \Sigma_{c'}$ for some c' with $s' = s(c') < s(c)$. Then, there exists a cycle,

$$\sum_\tau m_\tau \cdot \tau, \tag{2}$$

representing the class $H_p(K_{s'})/i_*^{s'-1,s'}(H_p(K_{s'-1}))$, with $m_\sigma \neq 0$, and $\mathrm{rk}_<(\tau) < s(c)$ for each τ with $m_\tau \neq 0$. Thus, there exists a relation

$$\sigma = \sum_{\tau \neq \sigma} m_\tau \cdot \tau \mod i_*^{s'-1,s'}(H_p(K_{s'-1})) \tag{3}$$

with $\partial_p \sigma = \partial_p \left(\sum_{\tau \neq \sigma} m_\tau \cdot \tau \right)$, and $\mathrm{rk}_<(\tau) < s(c)$ for each $\tau \neq \sigma$ with $m_\tau \neq 0$. Moreover, it is clear from the definition of persistent homology and the fact that $s > s'$, that, $\sigma = \sum_{\tau \neq \sigma} m_\tau \cdot \tau$ mod $i_*^{s-1,s}(H_p(K_{s-1}))$ as well.

Thus, we can substitute for σ in (2) by the right hand side of (2) and thus obtain an equivalent expression for the cycle representing the non-zero class in $H_p(K_s)/i_*^{s-1,s}(H_p(K_{s-1}))$ which does not contain σ, thus contradicting the fact that $\sigma \in \Sigma_c$. This proves that $\sigma \notin \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'}$, proving that

$$\sigma \in \Sigma \setminus \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'}.$$

This proves the inclusion $\Sigma_c \subset \Sigma \setminus \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'}$.

We now prove the reverse inclusion.

Suppose that $\Sigma \setminus \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'} \not\subset \Sigma_c$. Let

$$\sigma \in \Sigma \setminus \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'} \setminus \Sigma_c$$

having the maximal rank. Clearly, $\mathrm{rk}_<(\sigma) \leq s(c)$. Now clearly, if $\mathrm{rk}_<(\sigma) = s(c)$, then $\sigma \in \Sigma_c$. Otherwise, if $\sigma \notin \Sigma_c$, there exists an expression for a cycle

$$\sum_\tau m'_\tau \cdot \tau, \tag{4}$$

congruent to the cycle in (4), with $m'_\sigma = 0$, and $m'_\tau = m_\tau$ for all τ satisfying $m_\tau \neq 0$, and $\mathrm{rk}_<(\tau) > \mathrm{rk}_<(\sigma)$. Subtracting the expression (4) from that in (2) we get a cycle,

$$\sigma + \sum_\tau m''_\tau \cdot \tau$$

with $\mathrm{rk}_<(\tau) < \mathrm{rk}_<(\sigma)$ for all τ with $m''_\tau \neq 0$. This implies (using the second part of Remark 7) that $\sigma \in \Sigma_{c''}$, where $c'' \in \mathrm{Bar}_p(\mathcal{F})$ with $s(c'') = \mathrm{rk}_<(\sigma)$. This contradicts the fact that $\sigma \notin \bigcup_{c' \in \mathrm{Bar}_p(\mathcal{F}), s(c') < s(c)} \Sigma_{c'}$. This finishes the proof of the reverse inclusion. ◀

▶ Remark 13. Theorem 12 furnishes us with an algorithm for computing the set Σ_c of essential simplices for each bar $c \in \mathrm{Bar}_p(\mathcal{F}), p \geq 0$, once we have an algorithm for computing a representative cycle for each such bar. Let $c \in \mathrm{Bar}_p(\mathcal{F})$ with $s(c) = s_0$, and we have computed (using an algorithm for computing persistent homology) the set $\Sigma \subset K^{(p)}$ of p-simplices appearing in a cycle representing a homology class corresponding to c. Assuming by induction that we have computed $\Sigma_{c'}$ for bars $c' \in \mathrm{Bar}_p(\mathcal{F})$ with $s(c') < s_0$, we can compute Σ_c using (1).

One filtered simplicial complex that plays an important role in applications of persistent homology theory, including the one in this paper, is the so called Vietoris-Rips complex associated to a weighted graph or equivalently a finite set V equipped with a distance function $w : V \times V \to \mathbb{R}$, satisfying $w(v, v) = 0$ for all $v \in V$.

▶ **Definition 14** (Vietoris-Rips filtration)**.** Let $M = (V, w)$ be a pair, where V is a finite set and $w : V \times V \to \mathbb{R}_{\geq 0}$ is a map (which need not be a metric on V) satisfying $w(v, v) = 0$ for all $v \in V$.

Let $K = 2^V$ denote the simplicial complex corresponding to the simplex with vertices elements of the set V. For any real number $d \geq 0$, we denote by K_d the sub-complex of K, defined by setting for each $0 \leq p \leq \mathrm{card}(V) - 1$, $K^{(p)} = \{[v_0, \ldots, v_p] \mid w(v_i, v_j) \leq d, 0 \leq i, j \leq p\}$.

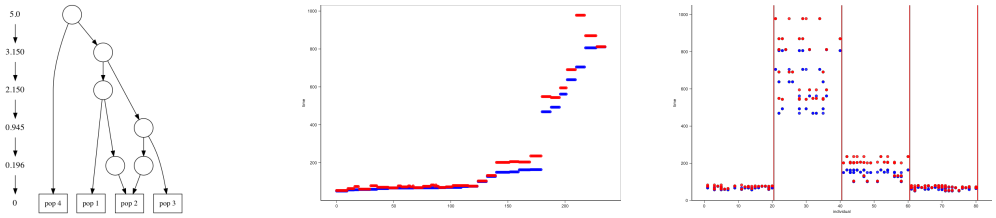
Clearly, if $d \leq d'$, $K_d \subset K_{d'}$, and there exists a a finite set of $0 = d_0 < d_1 < \cdots < d_N$ such that, $K_{d_i} \neq K_{d_{i+1}}$, and $K_{d_i} = K_{d_i + t}$ for $0 \leq t < d_{i+1} - d_i$, for all $i, 0 \leq i < N$.

We call the above filtration the *Vietoris-Rips filtration associated to* M.

▶ Remark 15. Note that for a generic weight function w on $V \times V$, the corresponding Vietoris-Rips filtration is identical to that induced by an admissible ordering (up to breaking ties) in an obvious manner.

3 Experiments

Population simulation. We specify the scaffold to the simulator for a variety of configurations that ranges form zero to three admixed populations. To avoid any unintended bias each population was simulated with the same number of individuals and similar population parameters: mutation rate 4×10^{-8} mut/bp/gen, recombination rate 0.3×10^{-8} cM/Mb/gen and segment length 150Kb and effective population size of 10^4. Each simulated scenario is repeated at least five times. SimRA generates the set of individual haplotypes for each of the populations which is fed to detection algorithm for processing as follows.



(a) Populations scaffold

(b) RFL plot of essential simplices of 1-D cycles

(c) Sorted RFL plot

Figure 3 (a) The scaffold of the evolution scenario of four populations. Only populations 2 is admixed. (b) The corresponding RFL plot of the essential simplices of the one-dimensional cycles. The blue dot marks the birth and the red dot marks the death of the corresponding irreducible cycle. Notice the obvious kink in the plot. (c) Here the known population labels of the individuals are used to put the data in the four buckets.

Persistent Homology. Note that all the individuals of all the populations are put in a single group, i.e., we keep the population label aside and do not use them in any of the computations. Next we create a distance matrix between all pairs of individuals using the Hamming distance metric. The graph embedding of this distance matrix is the complete graph with each vertex corresponding to an individual and edge weight is the distance between the pair of vertices.

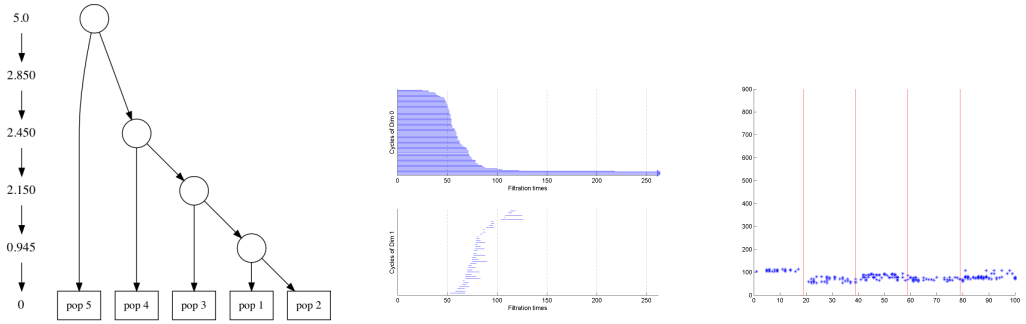
Next the Vietoris-Rips filtration (cf. Definition 14) is constructed on this embedding graph. The zero and one-dimensional persistent homology bar diagrams of the Vietoris-Rips filtration are computed using JavaPlex V 4.3.1 [10]. The set of essential simplices associated to the bars in the bar diagram are then computed using Remark 13.

Recall that the dimension of the zero-dimensional homology group of a simplicial complex counts the number of connected components of the simplicial complex, while the dimension of the one-dimensional homology group counts the number of independent one-dimensional cycles which do not bound. The barcode plots display individual cycles representing non-zero one-dimensional homology classes are born and when they disappear. The top half of each barcode plot for the simulation experiments shows the persistence of zero-dimensional homology and the bottom half shows that of one-dimensional homology. While short cycles can be due to noise, longer (persistent) cycles represent fundamental topological structures within the genetic distance matrix.

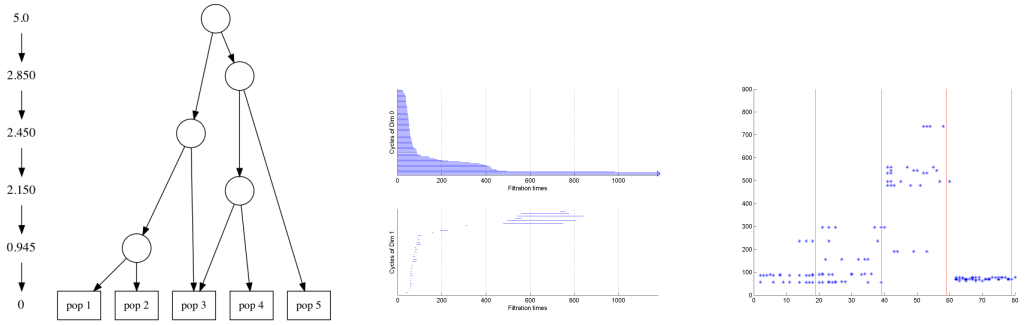
3.1 Visualization of the essential simplices

Note that the bar plot helps in visualizing the cycles representing non-zero homology classes each dimension. Similarly the RFL plot is meant to visualize the set of essential simplices associated to the different bars in the bar diagram (cf. Definition 9). The individual data plots are on the x-axis, in any order natural to the given problem. The y-axis represents the filtration time and shows the birth and death time of each individual as two dots in two different colors. See Fig 3 for an example with four populations. The RFL plot is shown in Fig 3 (b). The individuals here cluster into two groups with the esssential simplices of the late bars corresponding to the admixed population in this example.

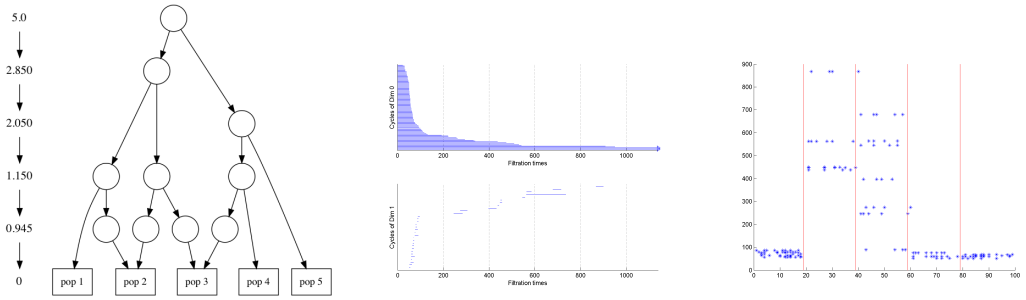
Fig 4 shows four distinct scenarios of admixed populations. The persistent homology of the Vietoris-Rips filtration are shown as bars. Notice that the 1-dimensional bars cluster into different groups whose transitions roughly correspond to the number of admixed populations. This is an empirical observation which we are currently continuing to probe. Further, the



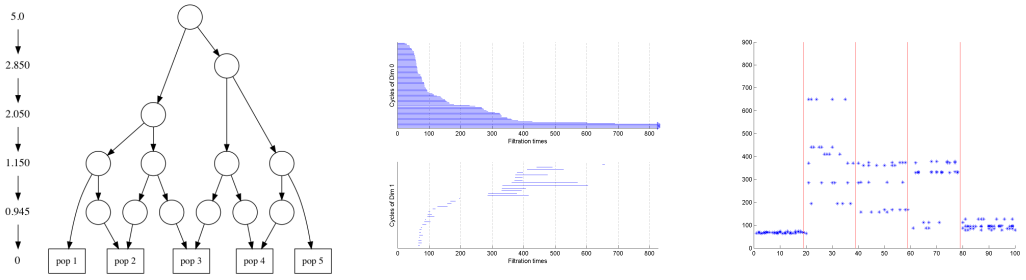
(a) No admixed populations.



(b) 1 population admixed.



(c) 2 populations admixed.



(d) 3 populations admixed.

Figure 4 Each row corresponds to a distinct scenario, whose scaffold is shown on the left. Each scenario has 100 individuals distributed equally amongst the populations. The bar plot of the persistent homology groups of the Vietoris-Rips filtration (cf. Definition 14) of the graph embedding of the distance matrix of the data points (of each scaffold) is shown in the center. The top half shows the persistence of the zero-dimensional while the bottom half shows that of the one-dimensional homologies. The corresponding RFL plot of the essential simplices of the irreducible one-dimensional cycles is shown on the right. For the latter only the birth points are shown. Also, in the RFL plot the individuals of the populations (x-axis) are separated based on their input labels, for convenience.

birth points on the RFL plot cluster at different filtration time points in the reverse order (recall that the time on the scaffold goes from present to past since the reference is 0 at present, for convenience). Thus the number of admixing events and their relative timing can be deduced from the RFL plots in combination with bar diagram. See Fig 4 for the details of the four scenarios.

4 Conclusion

We introduced the notion of essential simplices. This enables us to study the role of individuals in the persistent homology space in an unambiguous manner. Through simulations we show that the clustering of the bars in the bar diagram captures some of the admixing events in the population history and the relative timing of the events. This is reinforced by using essential simplices that further segregate the individuals. We believe that the notion of essential simplices is general enough to be of use in other applications as well.

References

1 Anna P. Carrieri, Filippo Utro, and Laxmi Parida. Sampling arg of multiple populations under complex configurations of subdivision and admixture. *Bioinformatics*, 32(7):1048–1056, 2016.

2 Herbert Edelsbrunner and John L. Harer. *Computational topology: An Introduction*. American Mathematical Society, Providence, RI, 2010.

3 Robin Forman. A user's guide to discrete Morse theory. *Sém. Lothar. Combin.*, 48:Art. B48c, 35, 2002.

4 Matthew L. Freedman, Christopher A. Haiman, Nick Patterson, Gavin J. McDonald, Arti Tandon, Alicja Waliszewska, Kathryn Penney, Robert G. Steen, Kristin Ardlie, Esther M. John, Ingrid Oakley-Girvan, Alice S. Whittemore, Kathleen A. Cooney, Sue A. Ingles, David Altshuler, Brian E. Henderson, and David Reich. Admixture mapping identifies 8q24 as a prostate cancer risk locus in african-american men. *Proceedings of the National Academy of Sciences*, 103(38):14068–14073, 2006.

5 Robert Ghrist. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45:61–75, 2008.

6 Mark Jobling, Edward Hollox, Matthew Hurles, Toomas Kivisild, and Chris Tyler-Smith. *Human evolutionary genetics*. Garland Science, UK, 2013.

7 M.J. Kearsey and H.S. Pooni. *The Genetical Analysis of Quantitative Traits*. Stanley Thornes, UK, 2004.

8 P. Y. Lum, G. Singh, A. Lehman, T. Ishkanov, M. Alagappan, J. Carlsson, G. Carlsson, and Mikael Vilhelm Vejdemo Johansson. Extracting insights from the shape of complex data using topology. *Scientific Reports*, 3, 2013.

9 Lammi Parida, Filippo Utro, Deniz Yorukoglu, Anna Paola Carrieri, David Kuhn, and Saugata Basu. Topological signatures for population admixture. In Teresa M. Przytycka, editor, *Research in Computational Molecular Biology*, pages 261–275. Springer International Publishing, 2015.

10 Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. JavaPlex: A research software package for persistent (co)homology. In Han Hong and Chee Yap, editors, *Proceedings of ICMS 2014*, Lecture Notes in Computer Science 8592, pages 129–136, 2014. Software available at `http://appliedtopology.github.io/javaplex/`.

11 J.D. Wall and M.F. Hammer. Archaic admixture in the human genome. *Current opinion in genetics & development*, 16(6):606–610, 2006.

Minimum Segmentation for Pan-genomic Founder Reconstruction in Linear Time

Tuukka Norri
Department of Computer Science, University of Helsinki, Helsinki, Finland
tuukka.norri@helsinki.fi
https://orcid.org/0000-0002-8276-0585

Bastien Cazaux
Department of Computer Science, University of Helsinki, Helsinki, Finland
bastien.cazaux@helsinki.fi
https://orcid.org/0000-0002-1761-4354

Dmitry Kosolobov
Department of Computer Science, University of Helsinki, Helsinki, Finland
dkosolobov@mail.ru
https://orcid.org/0000-0002-2909-2952

Veli Mäkinen
Department of Computer Science, University of Helsinki, Helsinki, Finland
veli.makinen@helsinki.fi
https://orcid.org/0000-0003-4454-1493

Abstract

Given a threshold L and a set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of m strings (haplotype sequences), each having length n, the minimum segmentation problem for founder reconstruction is to partition $[1, n]$ into set P of disjoint segments such that each segment $[a, b] \in P$ has length at least L and the number $d(a, b) = |\{R_i[a, b] \colon 1 \leq i \leq m\}|$ of distinct substrings at segment $[a, b]$ is minimized over $[a, b] \in P$. The distinct substrings in the segments represent founder blocks that can be concatenated to form $\max\{d(a, b) \colon [a, b] \in P\}$ founder sequences representing the original $\mathcal{R}$ such that crossovers happen only at segment boundaries. We give an optimal $O(mn)$ time algorithm to solve the problem, improving over earlier $O(mn^2)$. This improvement enables to exploit the algorithm on a pan-genomic setting of input strings being aligned haplotype sequences of complete human chromosomes, with a goal of finding a representative set of references that can be indexed for read alignment and variant calling. We implemented the new algorithm and give some experimental evidence on the practicality of the approach on this pan-genomic setting.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms, Applied computing → Bioinformatics

Keywords and phrases Pan-genome indexing, founder reconstruction, dynamic programming, positional Burrows–Wheeler transform, range minimum query

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.15

Funding This work is partially supported by the Academy of Finland (grant 309048).

1 Introduction

A key problem in *pan-genomics* is to develop a sufficiently small, efficiently queriable, but still descriptive representation of the variation common to the subject under study [1]. For example, when studying human population, one would like to take all publicly available

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen; Article No. 15; pp. 15:1–15:15

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

variation datasets (e.g. [19, 4, 20]) into account. Many approaches encode the variation as a graph [16, 9, 18, 2, 10, 8] and then one can encode the different haplotypes as paths in this graph [13, 17]. An alternative was proposed in [22], based on a compressed indexing scheme for a multiple alignment of all the haplotypes [11, 14, 23, 5, 7]. In either approach, scalability is hampered by the encoding of all the haplotypes.

We suggest to look for a smaller set of representative haplotype sequences to make the above pan-genomic representations scalable.

Finding such set of representative haplotype sequences that retain the original contiguities as well as possible, is known as the *founder sequence reconstruction* problem [21]. In this problem, one seeks a set of d founders such that the original m haplotypes can be mapped with minimum amount of *crossovers* to the founders. Here a crossover means a position where one needs to jump from one founder to another to continue matching the content of the haplotype in question. Unfortunately, this problem in NP-hard even to approximate within a constant factor [15].

For founder reconstruction to be scalable to the pan-genomic setting, one would need an algorithm to be nearly linear to the input size. With this is mind, we study a relaxation of founder reconstruction that is known to be polynomial time solvable: Namely, when limiting all the crossovers to happen at the same locations, one obtains a *minimum segmentation* problem specific to founder reconstruction [21]. A dynamic programming algorithm given in [21] has complexity $O(n^2m)$, where m is the number of haplotypes and n is the length of each of them.

In this paper, we improve the running time of solving the minimum segmentation problem of founder reconstruction to the optimal $O(mn)$ (linear in the input size).

We also implement the new algorithm, as well as a further heuristic that aims to minimize crossovers over the segment boundaries (yielded by the optimal solution to the minimum segmentation problem). In our experiments, we show that the approach is practical on human genome scale setting. Namely, we apply the implementation on a multiple alignment representing 5009 haplotypes of human chromosome 6, and the result is 130 founder sequences with the average distance of two crossovers being 9358 bases. Preserving such long contiguities in just 2.5% of the original input space is promising for the accuracy and scalability of the short read alignment and variant calling motivating our study.

The main technique behind the improvement is the use of *positional Burrows–Wheeler transform* (pBWT) [3], and more specifically its extension to larger alphabets [12]. While the original dynamic programming solution uses $O(nm)$ time to look for the best preceding segment boundary for each column of the input, we observe that at most m values in pBWT determine segment boundaries where the number of distinct founder substrings change. Minimums on the already computed dynamic programming values between each such interesting consecutive segment boundaries give the requested result. However, it turns out that we can maintain the minimums directly in pBWT internal structures (with some modifications) and have to store only the last L computed dynamic programming values, thus spending only $O(m + L)$ additional space, where L is the input threshold on the length of each segment. The segmentation is then reconstructed by standard backtracking approach in $O(n)$ time using an array of length n.

2 Notation and Problem Statement

For a string $s = c_1c_2 \cdots c_n$, denote by $|s|$ its length n. We write $s[i]$ for the letter c_i of s and $s[i, j]$ for the *substring* $c_ic_{i+1} \cdots c_j$. An analogous notation is used for arrays. For any numbers i and j, the set of integers $\{x \in \mathbb{Z} \colon i \leq x \leq j\}$ (possibly empty) is denoted by $[i, j]$.

The input for our problem is the set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of strings of length n, called *recombinants*. A set $\mathcal{F} = \{F_1, \ldots, F_d\}$ of strings of length n is called a *founder set* of $\mathcal{R}$ if for each string $R_i \in \mathcal{R}$, there exists a partition P_i of the segment $[1, n]$ into disjoint subsegments such that, for each $[a, b] \in P_i$, the string $R_i[a, b]$ is equal to $F_j[a, b]$ for some $j \in [1, d]$. The partition P_i together with the mapping of the segments $[a, b] \in P_i$ to substrings $F_j[a, b]$ is called a *parse of* R_i *in terms of* $\mathcal{F}$, and a set of parses for all $R_i \in \mathcal{R}$ is called a *parse of* $\mathcal{R}$ *in terms of* $\mathcal{F}$. The integers a and $b + 1$, for $[a, b] \in P_i$, are called *crossover points*; thus, in particular, 1 and $n + 1$ are always crossover points.

It follows from the definition that, in practice, it makes sense to consider founder sets only for pre-aligned recombinants. Throughout the paper we implicitly assume that this is the case, although all our algorithms, clearly, work in the unaligned setting too but the produce results may hardly make any sense.

We consider the problem of finding a "good" founder set $\mathcal{F}$ and a "good" corresponding parse of $\mathcal{R}$ according to a reasonable measure of goodness. Ukkonen [21] pointed out that such measures may contradict each other: for instance, a minimum founder set obviously has size $d = \max_{j \in [1,n]} |\{R_1[j], \ldots, R_m[j]\}|$, but parses corresponding to such set may have unnaturally many crossover points; conversely, $\mathcal{R}$ is a founder set of itself and the only crossover points of its trivial parse are 1 and $n + 1$, but the size m of this founder set is in most cases unacceptably large. Following Ukkonen's approach, we consider compromise parameterized solutions. The *minimum founder set problem* is, given a bound L and a set of recombinants $\mathcal{R}$, to find a smallest founder set $\mathcal{F}$ of $\mathcal{R}$ such that there exists a parse of $\mathcal{R}$ in terms of $\mathcal{F}$ in which the distance between any two crossover points is at least L (the crossover points may belong to parses of different recombinants, i.e., for $[a, b] \in P_i$ and $[a', b'] \in P_j$, where P_i and P_j are parses of R_i and R_j, we have either $a = a'$ or $|a - a'| \geq L$).

It is convenient to reformulate the problem in terms of segmentations of $\mathcal{R}$. A *segment* of $\mathcal{R} = \{R_1, \ldots, R_m\}$ is a set $\mathcal{R}[j, k] = \{R_i[j, k] \colon R_i \in \mathcal{R}\}$. A *segmentation* of $\mathcal{R}$ is a collection S of disjoint segments that covers the whole $\mathcal{R}$, i.e., for any distinct $\mathcal{R}[j, k]$ and $\mathcal{R}[j', k']$ from S, $[j, k]$ and $[j', k']$ do not intersect and, for each $x \in [1, n]$, there is $\mathcal{R}[j, k]$ from S such that $x \in [j, k]$. The *minimum segmentation problem* [21] is, given a bound L and a set of recombinants $\mathcal{R}$, to find a segmentation S of $\mathcal{R}$ such that $\max\{|\mathcal{R}[j, k]| \colon \mathcal{R}[j, k] \in S\}$ is minimized and the length of each segment from S is at least L; in other words, the problem is to compute

$$\min_{S \in S_L} \max\{|\mathcal{R}[j, k]| \colon \mathcal{R}[j, k] \in S\}, \tag{1}$$

where S_L is the set of all segmentations in which all segments have length at least L.

The minimum founder set problem and the minimum segmentation problem are, in a sense, equivalent: any segmentation S with segments of length at least L induces in an obvious way a founder set of size $\max\{|\mathcal{R}[j, k]| \colon \mathcal{R}[j, k] \in S\}$ and a parse in which all crossover points are located at segment boundaries (and, hence, at distance at least L from each other); conversely, if $\mathcal{F}$ is a founder set of $\mathcal{R}$ and $\{j_1, \ldots, j_p\}$ is the sorted set of all crossover points in a parse of $\mathcal{R}$ such that $j_q - j_{q-1} \geq L$ for $q \in [2, p]$, then $S = \{\mathcal{R}[j_{q-1}, j_q - 1] \colon q \in [2, p]\}$ is a segmentation of $\mathcal{R}$ with segments of length at least L and $\max\{|\mathcal{R}[j, k]| \colon \mathcal{R}[j, k] \in S\} \leq |\mathcal{F}|$.

Our main result is an algorithm that solves the minimum segmentation problem in the optimal $O(mn)$ time. The solution normally does not uniquely define a founder set of $\mathcal{R}$: for instance, if the built segmentation of $\mathcal{R} = \{baaaa, baaab, babab\}$ is $S = \{\mathcal{R}[1, 1], \mathcal{R}[2, 3], \mathcal{R}[4, 5]\}$, then the possible founder sets induced by S are $\mathcal{F}_1 = \{baaab, babaa\}$ and $\mathcal{F}_2 = \{baaaa, babab\}$. In other words, to construct a founder set, one concatenates fragments of recombinants corresponding to the found segments in a certain order. We return to this ordering problem in Sect. 4 and now focus on the details of the segmentation problem.

Hereafter, we assume that the input alphabet Σ is the set $[0, |\Sigma|-1]$ of size $O(m)$, which is a natural assumption considering that the typical alphabet size is 4 in our problem. It is sometimes convenient to view the set $\mathcal{R} = \{R_1, \ldots, R_m\}$ as a matrix with m rows and n columns. We say that an algorithm processing the recombinants $\mathcal{R}$ is streaming if it reads the input from left to right "columnwise", for each k from 1 to n, and outputs an answer for each set of recombinants $\{R_1[1,k], \ldots, R_m[1,k]\}$ immediately after reading the "column" $\{R_1[k], \ldots, R_m[k]\}$. The main result of the paper is the following theorem.

▶ **Theorem 1.** *Given a bound L and recombinants $\mathcal{R} = \{R_1, \ldots, R_m\}$, each having length n, there is an algorithm that computes* (1) *in a streaming fashion in the optimal $O(mn)$ time and $O(m+L)$ space. Using an additional array of length n, one can also find in $O(n)$ time a segmentation on which* (1) *is attained, thus solving the minimum segmentation problem.*

3 Minimum Segmentation Problem

Given a bound L and a set of recombinants $\mathcal{R} = \{R_1, \ldots, R_m\}$ each of which has length n, Ukkonen [21] proposed a dynamic programming algorithm that solves the minimum segmentation problem in $O(mn^2)$ time based on the following recurrence relation:

$$M(k) = \begin{cases} +\infty & \text{if } k < L, \\ |\mathcal{R}[1,k]| & \text{if } L \le k < 2L, \\ \min\limits_{0 \le j \le k-L} \max\{M(j), |\mathcal{R}[j+1,k]|\} & \text{if } k \ge 2L. \end{cases} \tag{2}$$

It is obvious that $M(n)$ is equal to the solution (1); the segmentation itself can be reconstructed by "backtracking" in a standard way (see [21]). We build on the same approach.

For a given $k \in [1,n]$, denote by $j_{k,1}, \ldots, j_{k,r_k}$ the sequence of all positions $j \in [1, k-L]$ in which the value of $|\mathcal{R}[j,k]|$ changes, i.e., $1 \le j_{k,1} < \cdots < j_{k,r_k} \le k-L$ and $|\mathcal{R}[j_{k,h}, k]| \neq |\mathcal{R}[j_{k,h}+1, k]|$ for $h \in [1, r_k]$. We complement this sequence with $j_{k,0} = 0$ and $j_{k,r_k+1} = k-L+1$, so that $j_{k,0}, \ldots, j_{k,r_k+1}$ can be interpreted as a splitting of the range $[0, k-L]$ into segments in which the value $|\mathcal{R}[j+1,k]|$ stays the same: namely, for $h \in [0, r_k]$, one has $|\mathcal{R}[j+1,k]| = |\mathcal{R}[j_{k,h+1}, k]|$ provided $j_{k,h} \le j < j_{k,h+1}$. Hence, $\min\limits_{j_{k,h} \le j < j_{k,h+1}} \max\{M(j), |\mathcal{R}[j+1,k]|\} = \max\{|\mathcal{R}[j_{k,h+1}, k]|, \min\limits_{j_{k,h} \le j < j_{k,h+1}} M(j)\}$ and, therefore, (2) can be rewritten as follows:

$$M(k) = \begin{cases} +\infty & \text{if } k < L, \\ |\mathcal{R}[1,k]| & \text{if } L \le k < 2L, \\ \min\limits_{0 \le h \le r_k} \max\{|\mathcal{R}[j_{k,h+1}, k]|, \min\limits_{j_{k,h} \le j < j_{k,h+1}} M(j)\} & \text{if } k \ge 2L. \end{cases} \tag{3}$$

Our crucial observation is that, for $k \in [1,n]$ and $j \in [1,k]$, one has $|\mathcal{R}[j+1,k]| \le |\mathcal{R}[j,k]| \le m$. Therefore, $m \ge |\mathcal{R}[j_{k,1}, k]| > \cdots > |\mathcal{R}[j_{k,r_k+1}, k]| \ge 1$ and $r_k < m$. Hence, $M(k)$ can be computed in $O(m)$ time using (3), provided one has the following components:

(i) the numbers $|\mathcal{R}[j_{k,h+1}, k]|$, for $h \in [0, r_k]$;

(ii) the values $\min\{M(j) \colon j_{k,h} \le j < j_{k,h+1}\}$, for $h \in [0, r_k]$.

In the remaining part of the section, we describe a streaming algorithm that reads the strings $\{R_1, \ldots, R_m\}$ "columnwise" from left to right and computes the components (i) and (ii) immediately after reading each "column" $\{R_1[k], \ldots, R_m[k]\}$, for $k \in [1,n]$, and all in $O(mn)$ total time and $O(m+L)$ space.

To reconstruct a segmentation corresponding to the found solution $M(n)$, we build along with the values $M(k)$ an array of size n whose kth element, for each $k \in [1,n]$, stores 0 if $M(k) = |\mathcal{R}[1,k]|$, and stores a number $j \in [1, k-L]$ such that $M(k) = \max\{M(j), |\mathcal{R}[j+1,k]|\}$

$R_1 = tttccat$
$R_2 = accatta$
$R_3 = actacct$
$R_4 = actccat$
$R_5 = cttacct$
$R_6 = atcacat$

i	1	2	3	4	5	6
$a_k[i]$	2	6	4	1	3	5
$d_k[i]$	8	8	5	3	7	3

i	1	2	3	4	5	6	7
$\lvert\mathcal{R}[i,k]\rvert$	6	6	4	4	3	3	2

$a_k[1] = 2$
$a_k[2] = 6$
$a_k[3] = 4$
$a_k[4] = 1$
$a_k[5] = 3$
$a_k[6] = 5$

a	c	c	a	t	t	a
a	t	c	a			
a	c					
t	t	t	c	c	a	t
a	c	t	a	c	c	
c	t					

Figure 1 The pBWT for a set of recombinants $\mathcal{R} = \{R_1, \ldots, R_6\}$ with $k = 7$ and the corresponding trie containing the reversed strings $R_1[1,k], \ldots, R_6[1,k]$ in lexicographic order.

otherwise; then, the segmentation can be reconstructed from the array in an obvious way in $O(n)$ time. In order to maintain the array, our algorithm computes, for each $k \in [1,n]$, along with the values $\min\{M(j) \colon j_{k,h} \le j < j_{k,h+1}\}$, for $h \in [0, r_k]$, positions j on which these minima are attained (see below). Further details are straightforward and, thence, omitted.

3.1 Positional Burrows–Wheeler Transform

Let us fix $k \in [1,n]$. Throughout this subsection, the string $R_i[k]R_i[k-1]\cdots R_i[1]$, which is the reversal of $R_i[1,k]$, is denoted by $R'_{i,k}$, for $i \in [1,m]$. Given a set of recombinants $\mathcal{R} = \{R_1, \ldots, R_m\}$ each of which has length n, a *positional Burrows–Wheeler transform (pBWT)*, as defined by Durbin [3], is a pair of integer arrays $a_k[1,m]$ and $d_k[1,m]$ such that:

1. $a_k[1,m]$ is a permutation of $[1,m]$ such that $R'_{a_k[1],k} \le \cdots \le R'_{a_k[m],k}$ lexicographically;
2. $d_k[i]$, for $i \in [1,m]$, is an integer such that $R_{a_k[i]}[d_k[i],k]$ is the longest common suffix of $R_{a_k[i]}[1,k]$ and $R_{a_k[i-1]}[1,k]$, and $d_k[i] = k+1$ if either this suffix is empty or $i = 1$.

▶ **Example 2.** Consider the following example, where $m = 6$, $k = 7$, and $\Sigma = \{a, c, t\}$. It is easy to see that the pBWT implicitly encodes the trie depicted in the right part of Figure 1, and such interpretation drives the intuition behind this structure: The trie represents the *reversed* sequences $R_1[1,k], \ldots, R_6[1,k]$ (i.e., read from right to left) in lexicographic order. Leaves (values a_k) store the corresponding input indices. The branches correspond to values d_k (the distance from the root subtracted from $k+1$). Our main algorithm in this paper makes implicitly a sweep-line over the trie stopping at the branching positions.

Durbin [3] showed that a_k and d_k can be computed from a_{k-1} and d_{k-1} in $O(m)$ time on the binary alphabet. Mäkinen and Norri [12] further generalized the construction for integer alphabets of size $O(m)$, as in our case. For the sake of completeness, we describe in this subsection the solution from [12] (see Figure 2a), which serves then as a basis for our main algorithm. We also present a modification of this solution (see Figure 2b), which, albeit seems to be slightly inferior in theory (we could prove only $O(m \log |\Sigma|)$ time upper bound), showed better performance in practice and thus, as we believe, is interesting by itself.

▶ **Lemma 3.** *The arrays $a_k[1,m]$ and $d_k[1,m]$ can be computed from $a_{k-1}[1,m]$ and $d_{k-1}[1,m]$ in $O(m)$ time, assuming the input alphabet is $[0, |\Sigma|-1]$ with $|\Sigma| = O(m)$.*

Proof. Given a_{k-1} and d_{k-1}, we are to show that the algorithm from Figure 2a correctly computes a_k and d_k. Since, for any $i, j \in [1,m]$, we have $R'_{i,k} \le R'_{j,k}$ iff either $R_i[k] < R_j[k]$, or $R_i[k] = R_j[k]$ and $R'_{i,k-1} \le R'_{j,k-1}$ lexicographically, it is easy to see that the array a_k can be deduced from a_{k-1} by radix sorting the sequence of pairs $\{(R_{a_{k-1}[i]}[k], R'_{a_{k-1}[i],k-1})\}_{i=1}^{m}$. Further, since, by definition of a_{k-1}, the second components of the pairs are already in a sorted order, it remains to sort the first components by the counting sort. Accordingly, in

zero initialize $C[0, |\Sigma|]$ and $P[0, |\Sigma| - 1]$
for $i \leftarrow 1$ **to** m **do**
  $C[R_i[k] + 1] \leftarrow C[R_i[k] + 1] + 1$;
for $i \leftarrow 1$ **to** $|\Sigma| - 1$ **do** $C[i] \leftarrow C[i] + C[i - 1]$;
for $i \leftarrow 1$ **to** m **do**
  $b \leftarrow R_{a_{k-1}[i]}[k]$;
  $C[b] \leftarrow C[b] + 1$;
  $a_k[C[b]] \leftarrow a_{k-1}[i]$;
  if $P[b] = 0$ **then** $d_k[C[b]] \leftarrow k + 1$;
  else $d_k[C[b]] \leftarrow \max\{d_{k-1}[\ell] \colon P[b] < \ell \leq i\}$;
  $P[b] \leftarrow i$;

(a) The basic pBWT algorithm computing a_k and d_k from a_{k-1} and d_{k-1}.

zero initialize $C[0, |\Sigma|]$ and $P[0, |\Sigma| - 1]$
for $i \leftarrow 1$ **to** m **do**
  $C[R_i[k] + 1] \leftarrow C[R_i[k] + 1] + 1$;
for $i \leftarrow 1$ **to** $|\Sigma| - 1$ **do** $C[i] \leftarrow C[i] + C[i - 1]$;
for $i \leftarrow 1$ **to** m **do**
  $b \leftarrow R_{a_{k-1}[i]}[k]$;
  $C[b] \leftarrow C[b] + 1$;
  $a_k[C[b]] \leftarrow a_{k-1}[i]$;
  $a_{k-1}[i] \leftarrow i + 1$; ▷ erase $a_{k-1}[i]$
  if $P[b] = 0$ **then** $d_k[C[b]] \leftarrow k + 1$;
  else $d_k[C[b]] \leftarrow \mathsf{maxd}(P[b] + 1, i)$;
  $P[b] \leftarrow i$;
function $\mathsf{maxd}(j,\ i)$
  if $j \neq i$ **then**
    $d_{k-1}[j] \leftarrow \max\{d_{k-1}[j], \mathsf{maxd}(a_{k-1}[j], i)\}$;
    $a_{k-1}[j] \leftarrow i + 1$;
  return $d_{k-1}[j]$;

(b) The algorithm with simple RMQ; a_{k-1} and d_{k-1} are used as auxiliary arrays (and corrupted).

Figure 2 The computation of a_k and d_k from a_{k-1} and d_{k-1} in the pBWT.

Figure 2a, the first loop counts occurrences of letters in the sequence $\{R_i[k]\}_{i=1}^m$ using an auxiliary array $C[0, |\Sigma|]$; as is standard in the counting sort, the second loop modifies the array C so that, for each letter $b \in [0, |\Sigma|-1]$, $C[b] + 1$ is the first index of the "bucket" that will contain all $a_{k-1}[i]$ such that $R_{a_{k-1}[i]}[k] = b$; finally, the third loop fills the buckets incrementing the indices $C[b] \leftarrow C[b] + 1$, for $b = R_{a_{k-1}[i]}[k]$, and performing the assignments $a_k[C[b]] \leftarrow a_{k-1}[i]$, for $i = 1, \ldots, m$. Thus, the array a_k is computed correctly. All is done in $O(m + |\Sigma|)$ time, which is $O(m)$ since the input alphabet is $[0, |\Sigma|-1]$ and $|\Sigma| = O(m)$.

The last three lines of the algorithm are responsible for computing d_k. Denote the length of the longest common prefix of any strings s_1 and s_2 by $\mathtt{LCP}(s_1, s_2)$. The computation of d_k relies on the following well-known fact: given a sequence of strings $s_1, \ldots, s_r$ such that $s_1 \leq \cdots \leq s_r$ lexicographically, one has $\mathtt{LCP}(s_1, s_r) = \min\{\mathtt{LCP}(s_{i-1}, s_i) \colon 1 < i \leq r\}$. Suppose that the last loop of the algorithm, which iterates through all i from 1 to m, assigns $a_k[i'] \leftarrow a_{k-1}[i]$, for a given $i \in [1, m]$ and some $i' = C[b]$. Let j be the maximum integer such that $j < i$ and $R_{a_{k-1}[j]}[k] = R_{a_{k-1}[i]}[k]$ (if any). The definition of a_k implies that $a_k[i'-1] = a_{k-1}[j]$ if such j exists. Hence, $\mathtt{LCP}(R'_{a_k[i'-1],k}, R'_{a_k[i'],k}) = 1 + \min\{\mathtt{LCP}(R'_{a_{k-1}[\ell-1],k-1}, R'_{a_{k-1}[\ell],k-1}) \colon j < \ell \leq i\}$ if such number j does exist, and $\mathtt{LCP}(R'_{a_k[i'-1],k}, R'_{a_k[i'],k}) = 0$ otherwise. Therefore, since $d_k[i']$ equals $k+1-\mathtt{LCP}(R'_{a_k[i'],k}, R'_{a_k[i'-1],k})$, we have either $d_k[i'] = \max\{d_{k-1}[\ell] \colon j < \ell \leq i\}$ or $d_k[i'] = k+1$ according to whether the required j exists. To find j, we simply maintain an auxiliary array $P[0, |\Sigma|-1]$ such that on the ith loop iteration, for any letter $b \in [0, |\Sigma|-1]$, $P[b]$ stores the position of the last seen b in the sequence $R_{a_{k-1}[1]}[k], R_{a_{k-1}[2]}[k], \ldots, R_{a_{k-1}[i-1]}[k]$, or $P[b] = 0$ if b occurs for the first time. Thus, d_k is computed correctly.

In order to calculate the maximums $\max\{d_{k-1}[\ell] \colon P[b] \leq \ell \leq i\}$ in $O(1)$ time, we build a range maximum query (RMQ) data structure on the array $d_{k-1}[1, m]$ in $O(m)$ time (e.g., see [6]). Therefore, the running time of the algorithm from Figure 2a is $O(m)$. ◀

In practice the bottleneck of the algorithm is the RMQ data structure, which, although answers queries in $O(1)$ time, has a sensible constant under the big-O in the construction time. We could naively compute the maximums by scanning the ranges $d_{k-1}[P[b]+1, i]$ from

left to right but such algorithm works in quadratic time since same ranges of d_{k-1} might be processed many times in the worst case. Our key idea is to store the work done by a simple scanning algorithm to reuse it in future queries. We store this information right in the arrays a_{k-1} and d_{k-1} rewriting them; in particular, since a_{k-1} is accessed sequentially from left to right in the last loop, the range $a_{k-1}[1, i]$ is free to use after the ith iteration.

More precisely, after the ith iteration of the last loop, the subarrays $a_{k-1}[1, i]$ and $d_{k-1}[1, i]$ are modified so that the following invariant holds: for any $j \in [1, i]$, $j < a_{k-1}[j] \leq i + 1$ and $d_{k-1}[j] = \max\{d'_{k-1}[\ell] \colon j \leq \ell < a_{k-1}[j]\}$, where d'_{k-1} denotes the original array d_{k-1} before modifications; note that the invariant holds if one simply puts $a_{k-1}[j] = j + 1$ without altering $d_{k-1}[j]$. Then, to compute $\max\{d'_{k-1}[\ell] \colon j \leq \ell \leq i\}$, we do not have to scan all elements but can "jump" through the chain $j, a_{k-1}[j], a_{k-1}[a_{k-1}[j]], \ldots, i$ and use maximums precomputed in $d_{k-1}[j], d_{k-1}[a_{k-1}[j]], d_{k-1}[a_{k-1}[a_{k-1}[j]]], \ldots, d_{k-1}[i]$; after this, we redirect the "jump pointers" in a_{k-1} to $i + 1$ and update the maximums in d_{k-1} accordingly. This idea is implemented in Figure 2b. Notice the new line $a_{k-1}[i] \leftarrow i + 1$ in the main loop (it is commented), which erases $a_{k-1}[i]$ and makes it a part of the "jump table". The correctness of the algorithm is clear. But it is not immediate even that the algorithm works in $O(m \log m)$ time. The next lemma states that the bound is actually even better, $O(m \log |\Sigma|)$. The proof of the lemma is given in Appendix.

▶ **Lemma 4.** *The algorithm from Figure 2b computes the arrays $a_k[1, m]$ and $d_k[1, m]$ from $a_{k-1}[1, m]$ and $d_{k-1}[1, m]$ in $O(m \log |\Sigma|)$ time, assuming the input alphabet is $[0, |\Sigma|-1]$ with $|\Sigma| = O(m)$.*

3.2 Modification of the pBWT

We are to modify the basic pBWT construction algorithm in order to compute the sequence $j_{k,1}, \ldots, j_{k,r_k}$ of all positions $j \in [1, k - L]$ in which $|\mathcal{R}[j, k]| \neq |\mathcal{R}[j + 1, k]|$, and to calculate the numbers $|\mathcal{R}[j_{k,h+1}, k]|$ and $\min\{M(j) \colon j_{k,h} \leq j < j_{k,h+1}\}$, for $h \in [0, r_k]$ (assuming $j_{k,0} = 0$ and $j_{k,r_k+1} = k - L + 1$); see the beginning of the section. As it follows from (3), these numbers are sufficient to calculate $M(k)$, as defined in (2) and (3), in $O(m)$ time. The following lemma reveals relations between the sequence $j_{k,1}, \ldots, j_{k,r_k}$ and the array d_k.

▶ **Lemma 5.** *Consider recombinants $\mathcal{R} = \{R_1, \ldots, R_m\}$, each having length n. For $k \in [1, n]$ and $j \in [1, k - 1]$, one has $|\mathcal{R}[j, k]| \neq |\mathcal{R}[j + 1, k]|$ iff $j = d_k[i] - 1$ for some $i \in [1, m]$.*

Proof. Suppose that $|\mathcal{R}[j, k]| \neq |\mathcal{R}[j + 1, k]|$. It is easy to see that $|\mathcal{R}[j, k]| > |\mathcal{R}[j + 1, k]|$, which implies that there are two indices h and h' such that $R_h[j + 1, k] = R_{h'}[j + 1, k]$ and $R_h[j] \neq R_{h'}[j]$. Denote by $a_k^{-1}[h]$ the number x such that $a_k[x] = h$. Without loss of generality, assume that $a_k^{-1}[h] < a_k^{-1}[h']$. Then, there exists $i \in [a_k^{-1}[h] + 1, a_k^{-1}[h']]$ such that $R_{a_k[i-1]}[j + 1, k] = R_{a_k[i]}[j + 1, k]$ and $R_{a_k[i-1]}[j] \neq R_{a_k[i]}[j]$. Hence, $d_k[i] = j + 1$.

Suppose now that $j \in [1, k - 1]$ and $j = d_k[i] - 1$, for some $i \in [1, m]$. Since $j < k$ and $d_k[1] = k + 1$, we have $i > 1$. Then, by definition of d_k, $R_{a_k[i-1]}[j + 1, k] = R_{a_k[i]}[j + 1, k]$ and $R_{a_k[i-1]}[j] \neq R_{a_k[i]}[j]$, i.e., $R_{a_k[i]}[j + 1, k]$ can be "extended" to the left in two different ways, thus producing two distinct strings in the set $\mathcal{R}[j, k]$. Therefore, $|\mathcal{R}[j, k]| > |\mathcal{R}[j + 1, k]|$. ◀

Denote by r the number of distinct integers in the array d_k. Clearly, r may vary from 1 to m. For integer ℓ, define $M'(\ell) = M(\ell)$ if $1 \leq \ell \leq k - L$, and $M'(\ell) = +\infty$ otherwise (M' is introduced for purely technical reasons). Our modified algorithm does not store d_k but stores the following four arrays (but we still often refer to d_k for the sake of analysis):

- $s_k[1, r]$ contains all distinct elements from $d_k[1, m]$ in the increasing sorted order;
- $e_k[1, m]$: for $j \in [1, r]$, $e_k[j]$ is equal to the unique index such that $s_k[e_k[j]] = d_k[j]$;

- $t_k[1,r]$: for $j \in [1,r]$, $t_k[j]$ is equal to the number of times $s_k[j]$ occurs in $d_k[1,m]$;
- $u_k[1,r]$: for $j \in [1,r]$, $u_k[j] = \min\{M'(\ell)\colon s_k[j-1]-1 \le \ell < s_k[j]-1\}$, assuming $s_k[0] = 1$.

▶ **Example 6.** In Example 2, where $m = 6$, $k = 7$, and $\Sigma = \{a, c, t\}$, we have $r = 4$, $s_k = [3, 5, 7, 8]$, $t_k = [2, 1, 1, 2]$, $e_k = [4, 4, 2, 1, 3, 1]$. It is easy to see that the array s_k marks positions of the branching nodes in the trie from Figure 1 in the increasing order (in the special case $s_k[1] = 1$, $s_k[1]$ does not mark any such node). The arrays s_k and e_k together emulate d_k. The array t_k will be used below to calculate some numbers $|\mathcal{R}[j,k]|$ required to compute $M(k)$. Further, suppose that $L = 3$, so that $k - L = 4$. Then, $u_k[1] = M(1)$, $u_k[2] = \min\{M(2), M(3)\}$, $u_k[3] = \min\{M(4), M'(5)\} = M(4)$ since $M'(5) = +\infty$, and $u_k[4] = M'(6) = +\infty$. The use of u_k is discussed in the sequel.

For convenience, let us recall Equation (3) defined in the beginning of this section:

$$M(k) = \begin{cases} +\infty & \text{if } k < L, \\ |\mathcal{R}[1,k]| & \text{if } L \le k < 2L, \\ \min\limits_{0 \le h \le r_k} \max\{|\mathcal{R}[j_{k,h+1}, k]|, \min\limits_{j_{k,h} \le j < j_{k,h+1}} M(j)\} & \text{if } k \ge 2L, \end{cases} \quad \text{(3 revisited)}$$

where $j_{k,0} = 0$, $j_{k,r_k+1} = k-L+1$, and $j_{k,1}, \ldots, j_{k,r_k}$ is the increasing sequence of all positions $j \in [1, k-L]$ in which $|\mathcal{R}[j,k]| \ne |\mathcal{R}[j+1,k]|$. In order to compute $M(k)$, one has to find the minima $\min\limits_{j_{k,h} \le j < j_{k,h+1}} M(j)$ and calculate $|\mathcal{R}[j_{k,h+1}, k]|$. As it follows from Lemma 5 and the definition of s_k, all positions $j \in [1, k-1]$ in which $|\mathcal{R}[j,k]| \ne |\mathcal{R}[j+1,k]|$ are represented by the numbers $s_k[i] - 1$ such that $1 < s_k[i] \le k$ (in the increasing order); hence, the sequence $j_{k,1}, \ldots, j_{k,r_k}$ corresponds to either $s_k[1] - 1, \ldots, s_k[r_k] - 1$ or $s_k[2] - 1, \ldots, s_k[r_k + 1] - 1$, depending on whether $s_k[1] \ne 1$. Then, the minima $\min\limits_{j_{k,h} \le j < j_{k,h+1}} M(j)$ are stored in the corresponding elements of u_k (assuming $s_k[0] = 1$): $u_k[i] = \min\{M'(\ell)\colon s_k[i-1]-1 \le \ell < s_k[i]-1\} = \min\{M(\ell)\colon s_k[i-1]-1 \le \ell < \min\{s_k[i]-1, k-L+1\}\} = \min\limits_{j_{k,h} \le j < j_{k,h+1}} M(j)$, provided $s_k[i-1] - 1 = j_{k,h}$. It is clear that $u_k[i] \ne +\infty$ only if the segment $[s_k[i-1] - 1, s_k[i] - 2]$ intersects the range $[1, k-L]$ and, thus, corresponds to a segment $[j_{k,h}, j_{k,h+1} - 1]$, for $h \in [0, r_k]$. Therefore, since $M'(\ell) = +\infty$ for $\ell < 1$ and $\ell > k - L$ and, thus, such values $M'(\ell)$ do not affect, in a sense, the minima stored in u_k, one can rewrite (3) as follows:

$$M(k) = \begin{cases} +\infty & \text{if } k < L, \\ |\mathcal{R}[1,k]| & \text{if } L \le k < 2L, \\ \min\limits_{1 \le j \le |u_k|} \max\{|\mathcal{R}[s_k[j] - 1, k]|, u_k[j]\} & \text{if } k \ge 2L. \end{cases} \quad (4)$$

It remains to compute the numbers $|\mathcal{R}[s_k[j] - 1, k]|$, for $j \in [1, |s_k|]$.

▶ **Lemma 7.** *Consider a set of recombinants $\mathcal{R} = \{R_1, \ldots, R_m\}$, each of which has length n. For $k \in [1, n]$ and $j \in [1, |s_k|]$, one has $|\mathcal{R}[s_k[j] - 1, k]| = t_k[j] + t_k[j+1] + \cdots + t_k[|t_k|]$.*

Proof. Denote $\ell = k - s_k[j] + 1$, so that $\mathcal{R}[s_k[j]-1, k] = \mathcal{R}[k-\ell, k]$. Suppose that $\ell = 0$. Note that $R_{a_k[1]}[k] \le \cdots \le R_{a_k[m]}[k]$. Since $d_k[i] = k + 1$ iff either $i = 1$ or $R_{a_k[i-1]}[k] \ne R_{a_k[i]}[k]$, it is easy to see that $|\mathcal{R}[k,k]|$, the number of distinct letters $R_i[k]$, is equal to the number of time $k + 1 = s_k[|s_k|]$ occurs in d_k, i.e., $t_k[|t_k|]$.

Suppose that $\ell > 0$. It suffices to show that $|\mathcal{R}[k-\ell, k]| - |\mathcal{R}[k-\ell+1, k]| = t_k[j]$. For $i \in [1, m]$, denote by R'_i the string $R_i[k]R_i[k-1]\cdots R_i[k-\ell]$. Fix $w \in \mathcal{R}[k-\ell+1, k]$. Since $R'_{a_k[1]} \le \cdots \le R'_{a_k[m]}$ lexicographically, there are numbers h and h' such that $R_{a_k[i]}[k-\ell+1, k] = w$ iff $i \in [h, h']$. Further, we have $R_{a_k[h]}[k-\ell] \le R_{a_k[h+1]}[k-\ell] \le \cdots \le R_{a_k[h']}[k-\ell]$.

Algorithm 1 Computing e_k, s_k, t_k, u_k, a_k from e_{k-1}, s_{k-1}, t_{k-1}, u_{k-1}, a_{k-1}.

1: copy s_{k-1} into s_k and add the element $k+1$ to the end of s_k, thus incrementing $|s_k|$;
2: copy u_{k-1} into u_k and add the element $M'(k-1)$ to the end of u_k, thus incrementing $|u_k|$;
3: zero initialize $C[0,|\Sigma|]$, $P[0,|\Sigma|-1]$, and $t_k[1,|s_k|]$;
4: **for** $i \leftarrow 1$ **to** m **do** $C[R_i[k]+1] \leftarrow C[R_i[k]+1]+1$;
5: **for** $i \leftarrow 1$ **to** $|\Sigma|-1$ **do** $C[i] \leftarrow C[i]+C[i-1]$;
6: **for** $i \leftarrow 1$ **to** m **do**
7: $\quad b \leftarrow R_{a_{k-1}[i]}[k]$;
8: $\quad C[b] \leftarrow C[b]+1$;
9: $\quad a_k[C[b]] \leftarrow a_{k-1}[i]$;
10: $\quad$ **if** $P[b]=0$ **then** $e_k[C[b]] \leftarrow |s_k|$;
11: $\quad$ **else** $e_k[C[b]] \leftarrow \max\{e_{k-1}[\ell] \colon P[b] < \ell \le i\}$
12: $\quad P[b] \leftarrow i$;
13: **for** $i \leftarrow 1$ **to** m **do** $t_k[e_k[i]] \leftarrow t_k[e_k[i]]+1$;
14: $j \leftarrow 1$;
15: add a new "dummy" element $+\infty$ to the end of u_k and u_{k-1};
16: **for** $i \leftarrow 1$ **to** $|s_k|$ **do**
17: $\quad u_k[j] \leftarrow \min\{u_k[j], u_{k-1}[i]\}$;
18: $\quad$ **if** $t_k[i] \neq 0$ **then**
19: $\quad\quad tmp[i] \leftarrow j$;
20: $\quad\quad s_k[j] \leftarrow s_k[i]$, $t_k[j] \leftarrow t_k[i]$, $u_k[j+1] \leftarrow u_{k-1}[i+1]$;
21: $\quad\quad$ **if** $s_k[j]-1 > k-L$ **and** $(j=1$ **or** $s_k[j-1]-1 \le k-L)$ **then** $u_k[j] \leftarrow \min\{u_k[j], M(k-L)\}$;
22: $\quad\quad j \leftarrow j+1$;
23: shrink s_k, t_k, and u_k to $j-1$ elements, so that $|s_k| = |t_k| = |u_k| = j-1$;
24: **for** $i \leftarrow 1$ **to** m **do** $e_k[i] \leftarrow tmp[e_k[i]]$;

Thus, by definition of d_k, for $i \in [h+1, h']$, we have $R_{a_k[i-1]}[k-\ell] \neq R_{a_k[i]}[k-\ell]$ iff $d_k[i] = k-\ell+1 = s_k[j]$. Note that $d_k[h] > s_k[j]$. Therefore, the number of strings $R_i[k-\ell,k]$ from $\mathcal{R}[k-\ell,k]$ having suffix w is equal to one plus the number of integers $s_k[j]$ in the range $d_k[h,h']$, which implies $|\mathcal{R}[k-\ell,k]| - |\mathcal{R}[k-\ell+1,k]| = t_k[j]$. ◀

By (4) and Lemma 7, one can calculate $M(k)$ in $O(m)$ time using the arrays t_k and u_k. It remains to describe how we maintain a_k, e_k, s_k, t_k, u_k.

▶ **Lemma 8.** *The arrays a_k, e_k, s_k, t_k, u_k can be computed from $a_{k-1}, e_{k-1}, s_{k-1}, t_{k-1}, u_{k-1}$ and from the numbers $M(k-L)$ and $M(k-1)$ in $O(m)$ time, assuming the input alphabet is $[0, |\Sigma|-1]$ with $|\Sigma| = O(m)$.*

Proof. Let us analyze Algorithm 1 that computes a_k, e_k, s_k, t_k, u_k. By definition, $d_{k-1}[i] = s_{k-1}[e_{k-1}[i]]$ for $i \in [1,m]$. The first line of the algorithm initializes s_k so that $d_{k-1}[i] = s_k[e_{k-1}[i]]$, for $i \in [1,m]$, and $s_k[|s_k|] = k+1$. Since after this initialization s_k, obviously, is in the sorted order, one has, for $i,j \in [1,m]$, $e_{k-1}[i] \le e_{k-1}[j]$ iff $d_{k-1}[i] \le d_{k-1}[j]$ and, therefore, for $\ell \in [i,j]$, one has $d_{k-1}[\ell] = \max\{d_{k-1}[\ell'] \colon i \le \ell' \le j\}$ iff $e_{k-1}[\ell] = \max\{e_{k-1}[\ell'] \colon i \le \ell' \le j\}$. Based on this observation, we fill e_k in lines 3–12 so that $d_k[i] = s_k[e_k[i]]$, for $i \in [1,m]$, using exactly the same algorithm as in Figure 2a, where d_k is computed, but instead of the assignment $d_k[C[b]] \leftarrow k+1$, we have $e_k[C[b]] \leftarrow |s_k|$ since $s_k[|s_k|] = k+1$. Here we also compute a_k in the same way as in Figure 2a.

The loop in line 13 fills t_k so that, for $i \in [1,|s_k|]$, $t_k[i]$ is the number of occurrences of the integer i in e_k (t_k was zero initialized in line 3). Since, for $i \in [1,m]$, we have $d_k[i] = s_k[e_k[i]]$ at this point, $t_k[i]$ is also the number of occurrences of the integer $s_k[i]$ in $d_k[1,m]$.

By definition, s_k must contain only elements from d_k, but this is not necessarily the case in line 14. In order to fix s_k and t_k, we simply have to remove all elements $s_k[i]$ for which $t_k[i] = 0$, moving all remaining elements of s_k and non-zero elements of t_k to the left accordingly. Suppose that, for some h and i, we have $e_k[h] = i$ and the number $s_k[i]$ is moved to $s_k[j]$, for some $j < i$, as we fix s_k. Then, $e_k[h]$ must become j. We utilize an additional temporary array $tmp[1, |s_k|]$ to fix e_k. The loop in lines 16–22 fixes s_k and t_k in an obvious way; once $s_k[i]$ is moved to $s_k[j]$ during this process, we assign $tmp[i] = j$. Then, s_k, t_k, u_k (u_k is discussed below) are resized in line 23, and the loop in line 24 fixes e_k using tmp.

Recall that $[s_k[j-1]-1, s_k[j]-2]$, for $j \in [1, |s_k|]$, is a system of disjoint segments covering $[0, k-1]$ (assuming $s_k[0] = 1$). It is now easy to see that this system is obtained from the system $[s_{k-1}[j-1]-1, s_{k-1}[j]-2]$, with $j \in [1, |s_{k-1}|]$ (assuming $s_{k-1}[0] = 1$), by adding the new segment $[k-1, k-1]$ and joining some segments together. The second line of the algorithm copies u_{k-1} into u_k and adds $M'(k-1)$ to the end of u_k, so that, for $j \in [1, |u_{k-1}|]$, $u_k[j]$ is equal to the minimum of $M'(\ell)$ for all ℓ from the segment $[s_{k-1}[j-1]-1, s_{k-1}[j]-2]$ and $u_k[|u_{k-1}|+1] = M'(k-1)$ is the minimum in the segment $[k-1, k-1]$. (This is not completely correct since M' has changed as k was increased; namely, $M'(k-L)$ was equal to $+\infty$ but now is equal to $M(k-L)$.) As we join segments removing some elements from s_k in the loop 16–22, the array u_k must be fixed accordingly: if $[s_k[j-1]-1, s_k[j]-2]$ is obtained by joining $[s_{k-1}[h-1]-1, s_{k-1}[h]-2]$, for $j' \le h \le j''$, then $u_k[j] = \min\{u_{k-1}[h] \colon j' \le h \le j''\}$. We perform such fixes in line 17, accumulating the latter minimum. We start accumulating a new minimum in line 20, assigning $u_k[j+1] \leftarrow u_{k-1}[i+1]$. If at this point the ready minimum accumulated in $u_k[j]$ corresponds to a segment containing the position $k-L$, we have to fix u_k taking into account the new value $M'(k-L) = M(k-L)$; we do this in line 21. To avoid accessing out of range elements in u_k and u_{k-1} in line 20, we add a "dummy" element in, respectively, u_k and u_{k-1} in line 15. ◀

Besides all the arrays of length m, the algorithm from Lemma 8 also requires access to $M(k-L)$ and, possibly, to $M(k-1)$. During the computation of $M(k)$ for $k \in [1, n]$, we maintain the last L calculated numbers $M(k-1), M(k-2), \ldots, M(k-L)$ in a circular array, so that the overall required space is $O(m+L)$; when k is incremented, the array is modified in $O(1)$ time in an obvious way. Thus, Lemma 8 implies Theorem 1

If, as in our case, one does not need s_k, t_k, u_k for all k, the arrays s_k, t_k, u_k can be modified in-place, i.e., s_k, t_k, u_k can be considered as aliases for s_{k-1}, t_{k-1}, u_{k-1}, and yet the algorithm remains correct. Thus, we really need only 7 arrays in total: a_k, a_{k-1}, e_k, e_{k-1}, s, t, u, where s, t, u serve as s_k, t_k, u_k and the array tmp can be organized in place of a_{k-1} or e_{k-1}. It is easy to maintain along with each value $u_k[j]$ a corresponding position ℓ such that $u_k[j] = M'(\ell)$; these positions can be used then to restore the found segmentation of $\mathcal{R}$ using backtracking (see the beginning of the section). To compute e_k, instead of using an RMQ data structure, one can adapt in an obvious way the algorithm from Figure 2b rewriting the arrays a_{k-1} and e_{k-1} during the computation, which is faster in practice but theoretically takes $O(m \log \sigma)$ time by Lemma 4. We do not discuss further details as they are straightforward.

4 Implementation

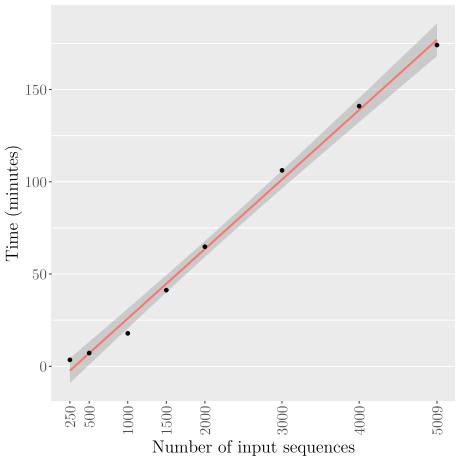
We implemented the segmentation algorithm using Lemma 4 to build the data structures, and thus the overall time complexity of the implemented approach is $O(mn \log \sigma)$. The implementation outputs for each segment the distinct founder sequence fragments, and associates to each fragment the set of haplotypes containing that fragment as a substring

at that location (these are easily deduced given the segmentation and the positional BWT structures). As discussed earlier, to obtain full founder sequences, one needs to concatenate these fragments in some order. If each segment has exactly the same amount of fragments, there is a way of finding an optimal concatenation minimizing the number of crossovers [21]: For any two consecutive segments, form a bipartite graph with fragments as nodes and edges corresponding to plausible concatenations. The cost of an edge is the symmetric difference of the sets associated with the fragments. Minimum cost perfect matching then gives an optimal concatenation order. Each consecutive pair can be solved independently, and founder sequences can be extracted through the paths formed by matches edges. However, not all segments may contain the maximum amount of distinct fragments; we conjecture the ordering problem becomes hard in this case (due to the local matchings being no longer independent). There are many possible heuristic ways to alleviate this issue, among which we opted to preprocess the segments to have the same amount of fragments; we duplicated the fragments greedily according to the sizes of the associated sets. Then we used the matching approach described above. In the upcoming extended version of this paper, we plan to study an alternative approach that provides an approximation guarantee.

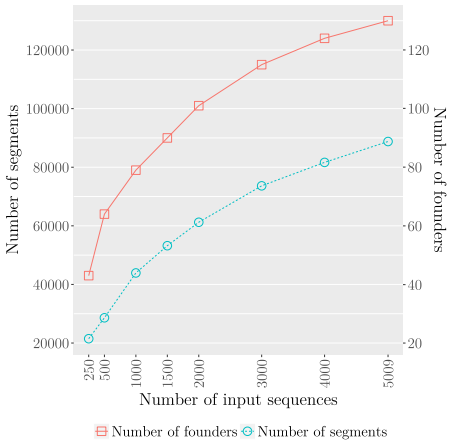
To test the implementation of our algorithm, we generated a multiple alignment of haplotype sequences from chromosome 6 variants from the phase 3 data of the 1000 Genomes Project [19]. This resulted in 5009 haplotype sequences of equal length (including the reference sequence) of approximately 171 million characters. We then discarded columns of identical characters, which reduced each sequence to approximately 5.38 million characters. We used an increasing number of these sequences as an input to our tool to verify its usability. The tests were run on a Ubuntu Linux 16.04 server. The server had 96 Intel Xeon E7-4830 v3 CPUs running at 2.10GHz and 1.4 TB of memory. Results on varying input sizes are shown in Fig. 3a. From this experiment it is conceivable that processing of thousands of complete human genomes takes only few CPU days. Figure 3b plots the number of founders against the number of segments. These look very promising, as using 130 founders instead of 5009 haplotypes as the input to pan-genome indexing of [22] will result into significant saving of resources; this solves the space bottleneck, and the preprocessing of founder reconstruction also saves time in the heavy indexing steps. The average segment length of some 1926 bases (171 million divided by 88778 segments) is also likely to be high enough not to break too badly the contiguity required for successful read alignment. To better see the contiguity of the resulting founder set, we mapped the haplotypes to the founders minimizing the number of jumps (simple greedy algorithm is optimal [15]). The result is shown in Fig. 3c. Since identical columns do not cause any jumps, we can now divide 171 million with 18274 jumps. This gives average distance between two jumps being 9358 bases. The heuristic part of optimizing the concatenation of founder blocks yields almost 5-fold improvement in the contiguity, compared to the worst case of each segment boundary yielding a discontinuity for each input sequence (which should be close to what a random concatenation order would produce).

Our intention was to compare our tool to an implementation of Ukkonen's algorithm [15]. However, initial testing with four input sequences showed that the latter implementation is not practical with a data set of this size.

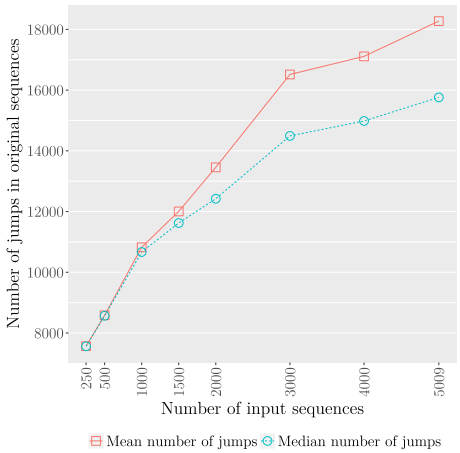
Our implementation is open source and available at the URL `https://github.com/tsnorri/founder-sequences`.



(a) The running time of our implementation plotted against the number of input sequences with $L = 10$. The data points have been fitted with a least-squares linear model, and the grey band shows the 95% confidence interval.



(b) The founder and segment counts as produced by our implementation plotted against the number of input sequences with $L = 10$.



(c) The average and median numbers of jumps, i.e. counts of positions where one needs to change from one founder sequence to another to read an original sequence, plotted against the number of input sequences with $L = 10$.

Figure 3 Evaluation of founder reconstruction on a pan-genomic setting.

5 Discussion

With 5009 haplotypes reducing down to 130 founders with the average distance of two crossovers being 9358 bases, one can expect short read alignment and variant calling to become practical on such pan-genomic setting. We are investigating this on our tool *PanVC* [22], where one can simply replace its input multiple alignment with the one made of the founder sequences. With graph-based approaches, slightly more effort is required: Input variations are encoded with respect to the reference, so one first needs to convert variants into a multiple alignment, apply the founder reconstruction algorithm, and finally convert

the multiple alignment of founder sequences into a directed acyclic graph. PanVC toolbox provides the required conversions. Alternatively, one can construct the pan-genome graph using other methods, and map the founder sequences afterwards to the paths of the graph: If original haplotype sequences are already spelled as paths, each founder sequence is a concatenation of existing subpaths, and can hence be mapped to a continuous path without alignment (possibly requiring adding a few missing edges).

Finally, it will be interesting to see how much the contiguity of the founder sequences can still be improved with different strategies for the concatenation order and with different formulations of the segmentation problem. For the former, we are investigating an approach with approximation guarantee. For the latter, we consider a variant with the number of founder sequenced fixed.

References

1 Computational Pan-Genomics Consortium et al. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, page bbw089, 2016.

2 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*, 47:682–688, 2015.

3 Richard Durbin. Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). *Bioinformatics*, 30(9):1266–1272, 2014.

4 Exome Aggregation Consortium. Analysis of protein-coding genetic variation in 60,706 humans. *Nature*, 536(7616):285–291, 2016.

5 Héctor Ferrada, Travis Gagie, Tommi Hirvola, and Simon J. Puglisi. Hybrid indexes for repetitive datasets. *Philosophical Transactions of the Royal Society A*, 372, 2014.

6 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *CPM 2006*, volume 4009 of *LNCS*, pages 36–48. Springer, 2006. doi:10.1007/11780441_5.

7 Travis Gagie and Simon J. Puglisi. Searching and indexing genomic databases via kernelization. *Frontiers in Bioengineering and Biotechnology*, 3(12), 2015.

8 Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Michael F Lin, Benedict Paten, and Richard Durbin. Sequence variation aware genome references and read mapping with the variation graph toolkit. *bioRxiv*, 2017. doi:10.1101/234856.

9 Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):361–370, 2013.

10 Sorina Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. A natural encoding of genetic variation in a Burrows-Wheeler transform to enable mapping and genome inference. In *Algorithms in Bioinformatics - 16th International Workshop, WABI 2016, Aarhus, Denmark, August 22-24, 2016. Proceedings*, volume 9838 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2016.

11 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.

12 Veli Mäkinen and Tuukka Norri. Applying the positional Burrows–Wheeler transform to all-pairs hamming distance. *Submitted manuscript*, 2018.

13 Tom O Mokveld, Jasper Linthorst, Zaid Al-Ars, and Marcel Reinders. Chop: Haplotype-aware path indexing in population graphs. *bioRxiv*, 2018. doi:10.1101/305268.

14 Gonzalo Navarro. Indexing highly repetitive collections. In *Proc. 23rd International Workshop on Combinatorial Algorithms (IWOCA)*, LNCS 7643, pages 274–279, 2012.

15 Pasi Rastas and Esko Ukkonen. Haplotype inference via hierarchical genotype parsing. In *Algorithms in Bioinformatics, 7th International Workshop, WABI 2007, Philadelphia, PA, USA, September 8-9, 2007, Proceedings*, pages 85–97, 2007.

16 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:R98, 2009.

17 Jouni Sirén, Erik Garrison, Adam M. Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. *arXiv preprint arXiv:1805.03834*, 2018.

18 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

19 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526(7571):68–74, sep 2015.

20 The UK10K Consortium. The UK10K project identifies rare variants in health and disease. *Nature*, 526(7571):82–90, 2015.

21 Esko Ukkonen. Finding founder sequences from a set of recombinants. In *Algorithms in Bioinformatics, Second International Workshop, WABI 2002, Rome, Italy, September 17-21, 2002, Proceedings*, pages 277–286, 2002.

22 Daniel Valenzuela, Tuukka Norri, Välimäki Niko, Esa Pitkänen, and Veli Mäkinen. Towards pan-genome read alignment to improve variation calling. *BMC Genomics*, 19(Suppl 2):87, 2018. doi:10.1186/s12864-018-4465-8.

23 Sebastian Wandelt, Johannes Starlinger, Marc Bux, and Ulf Leser. Rcsi: Scalable similarity search in thousand(s) of genomes. *PVLDB*, 6(13):1534–1545, 2013.

A Proof of Lemma 4

Proof. Fix $i \in [1, m]$. The ith iteration of the last loop in the algorithm computes the maximum in a range $d'_{k-1}[i', i]$, where d'_{k-1} is the original array d_{k-1} before modifications and $i' = P[b] + 1$ for some b and P. Let $\ell_i = i - i'$. Denote $\tilde{\ell} = \frac{1}{m}\sum_{i=1}^{m} \ell_i$, the "average query length". We are to prove that the running time of the algorithm is $O(m \log \tilde{\ell})$, which implies the result since $m\tilde{\ell} = \sum_{i=1}^{m} \ell_i$ and, obviously, $\sum_{i=1}^{m} \ell_i \le \sigma m$.

We say that a position j is *touched* if the function maxd is called with its first argument equal to j. Clearly, it suffices to prove that the total number of touches is $O(m \log \tilde{\ell})$. While processing the query maxd$(i-\ell_i, i)$, we may have touched many positions. Denote the sequence of all such position, for the given i, by $i_1, \ldots, i_r$; in other words, at the time of the query maxd$(i-\ell_i, i)$, we have $i_1 = i - \ell_i$, $i_j = a_{k-1}[i_{j-1}]$ for $j \in [2, r]$, and $i_r = i$. Obviously, $i_1 < \cdots < i_r$. We say that, for $j \in [1, r-1]$, the touch of i_j in the query maxd$(i-\ell_i, i)$ is *scaling* if there exists an integer r such that $i - i_j > 2^r$ and $i - i_{j+1} \le 2^r$ (see Figure 4). We count separately the total number of scaling and non-scaling touches in all i.

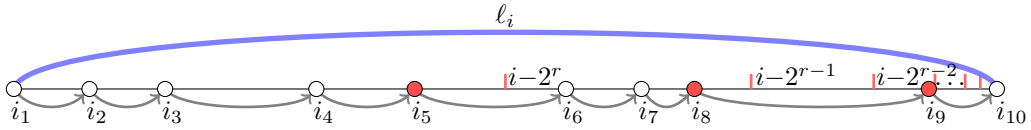


Figure 4 RMQ query on a range $[i - \ell_i, i]$; scaling touches are red.

For position j, denote by $p(j)$ the number of non-scaling touches of j. We are to prove that $P = \sum_{j=1}^{m} p(j) \le 2m \log \tilde{\ell}$. Let $q_h(j)$ denote the value of $a_{k-1}[j] - j$ in the hth non-scaling touch of j, for $h \in [1, p(j)]$. Suppose that this hth touch happens during the processing of a query maxd$(i-\ell_i, i)$. By the definition, $j + q_h(j)$ follows j in the sequence of touched positions. Since the touch of j is non-scaling, we have $i - j > j + q_h(j) > 2^r$, where r is the largest integer such that $i - j > 2^r$, and hence, $q_h(j) < 2^r$. Since maxd$(i - \ell_i, i)$ assigns $a_{k-1}[j] \leftarrow i + 1$, we have $a_{k-1}[j] - j > i - j > 2^r$ after the query. In other words, we had $a_{k-1}[j] - j = q_h(j) < 2^r$ before the query and have $a_{k-1}[j] - j > 2^r$ after. This immediately implies that $q_h(j) \ge 2^{h-1}$, for $h \in [1, p(j)]$, and, therefore, every position can be touched in the non-scaling way at most $O(\log m)$ times, implying $P = O(m \log m)$. But we can deduce a stronger bound. Since the sum of all values $j - a_{k-1}[j]$ for all positions j touched in a query maxd$(i - \ell_i, i)$ is equal to ℓ_i, it is obvious that $\sum_{j=1}^{m} \sum_{h=1}^{p(j)} q_h(j) \le \sum_{i=1}^{m} \ell_i = m\tilde{\ell}$. On the other hand, we have $\sum_{j=1}^{m} \sum_{h=1}^{p(j)} q_h(j) \ge \sum_{j=1}^{m} \sum_{h=1}^{p(j)} 2^{h-1} = \sum_{j=1}^{m} 2^{p(j)} - m$. The well-known property of the convexity of the exponent is that the sum $\sum_{j=1}^{m} 2^{p(j)}$ is minimized whenever all $p(j)$ are equal and maximal, i.e., $\sum_{j=1}^{m} 2^{p(j)} \ge \sum_{j=1}^{m} 2^{P/m}$. Hence, once $P > 2m \log \tilde{\ell}$, we obtain $\sum_{j=1}^{m} \sum_{h=1}^{p(j)} q_h(j) \ge \sum_{j=1}^{m} 2^{P/m} - m > m\tilde{\ell}^2 - m$, which is larger than $m\tilde{\ell}$ for $\tilde{\ell} \ge 2$ (the case $\tilde{\ell} < 2$ is trivial), contradicting $\sum_{j=1}^{m} \sum_{h=1}^{p(j)} q_h(j) \le m\tilde{\ell}$. Thus, $P = \sum_{j=1}^{m} p(j) \le 2m \log \tilde{\ell}$.

It remains to consider scaling touches. The definition implies that each query maxd$(i-\ell_i, i)$ performs at most $\log \ell_i$ scaling touches. Thus, it suffices to upperbound $\sum_{i=1}^{m} \log \ell_i$. Since the function log is concave, the sum $\sum_{i=1}^{m} \log \ell_i$ is maximized whenever all ℓ_i are equal and maximal, i.e., $\sum_{i=1}^{m} \log \ell_i \le \sum_{i=1}^{m} \log(\frac{1}{m}\sum_{j=1}^{m} \ell_j) = m \log \tilde{\ell}$, hence the result follows. ◀

Multiple-Choice Knapsack for Assigning Partial Atomic Charges in Drug-Like Molecules

Martin S. Engler
Life Sciences and Health Group, Centrum Wiskunde & Informatica,
Amsterdam, The Netherlands
martin.engler@cwi.nl

Bertrand Caron
School of Chemistry & Molecular Biosciences,
The University of Queensland, St Lucia, Australia
https://orcid.org/0000-0003-2305-1452

Lourens Veen
Netherlands eScience Center,
Amsterdam, The Netherlands

Daan P. Geerke
AIMMS Division of Molecular and Computational Toxicology,
Vrije Universiteit Amsterdam, The Netherlands
https://orcid.org/0000-0002-5262-6166

Alan E. Mark
School of Chemistry & Molecular Biosciences,
The University of Queensland, St Lucia, Australia
https://orcid.org/0000-0001-5880-4798

Gunnar W. Klau
Algorithmic Bioinformatics, Heinrich Heine University Düsseldorf, Germany
gunnar.klau@hhu.de
https://orcid.org/0000-0002-6340-0090

Abstract

A key factor in computational drug design is the consistency and reliability with which intermolecular interactions between a wide variety of molecules can be described. Here we present a procedure to efficiently, reliably and automatically assign partial atomic charges to atoms based on known distributions. We formally introduce the molecular charge assignment problem, where the task is to select a charge from a set of candidate charges for every atom of a given query molecule. Charges are accompanied by a score that depends on their observed frequency in similar neighbourhoods (chemical environments) in a database of previously parameterised molecules. The aim is to assign the charges such that the total charge equals a known target charge within a margin of error while maximizing the sum of the charge scores. We show that the problem is a variant of the well-studied multiple-choice knapsack problem and thus weakly $\mathcal{NP}$-complete. We propose solutions based on Integer Linear Programming and a pseudo-polynomial time Dynamic Programming algorithm. We show that the results obtained for novel molecules not included in the database are comparable to the ones obtained performing explicit charge calculations while decreasing the time to determine partial charges for a molecule by several orders of magnitude, that is, from hours or even days to below a second.

Our software is openly available at `https://github.com/enitram/charge_assign`.

2012 ACM Subject Classification Applied computing → Chemistry

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen; Article No. 16; pp. 16:1–16:13

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

Keywords and phrases Multiple-choice knapsack, integer linear programming, pseudo-polynomial dynamic programming, partial charge assignment, molecular dynamics simulations

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.16

Funding This research was partially supported by the Netherlands eScience Center (NLeSC), grant number 027.015.G06.

Acknowledgements We thank all members of the NLeSC-ASDI project *Enhancing Protein-Drug Binding Prediction* for valuable discussions. We thank Ulrich Pferschy from the University of Graz for providing insight and expertise on the multiple-choice knapsack problem.

1 Introduction

Molecule-based computational modelling and simulation studies play a central role in modern drug design and development. In particular, molecular dynamics (MD) simulations and free energy calculations are increasingly being used to screen potential ligand molecules in terms of their interactions with proposed target molecules (e.g. cell surface receptors or enzymes involved in metabolism) [11, 18]. They are also used to model structural changes in the target molecule associated with the binding of a given drug in order to understand the mechanism of action. The accuracy and utility of such modelling studies depends directly on the fidelity with which intermolecular interactions can be represented [1, 15]. While ideally one might wish to represent such interactions on the level of quantum mechanics, the size and complexity of protein/ligand complexes necessitates the use of classical dynamics in conjunction with empirical potentials. These so-called force fields are parameterised to reproduce the interactions between atoms in a system of interest (e.g. protein, membrane, drug) and involve bonds, angles, dihedrals, van der Waals and coulombic interactions.

Of particular importance is the assignment of partial atomic charges to describe the latter interactions. Partial atomic or point charges are used to represent the electrostatic potential around a molecule and the coulombic interactions between these point charges dominate the calculation of inter-molecular interactions. The difficulty is that the effective partial charge on an atom needed to represent the electrostatic potential surrounding a molecule is heavily dependent on the local environment in which an atom is found. For small molecules (< 40 atoms) partial atomic charges can be generally inferred *de novo* from quantum-mechanical computations [19]. However, when using e.g. commonly applied Density Functional Theory (DFT) such calculations scale cubic in the number of valence electrons [3], increasing the computational costs significantly. In addition, as molecules become larger the accuracy with which charges can be assigned decreases.

The standard approach to address this problem is to manually assign charges to atoms based on their similarity to atoms (or groups) in a set of reference molecules containing equivalent chemical moieties. The challenge in making such assignments is twofold: 1) the charges assigned to equivalent chemical groups in alternative reference molecules may vary making the choice of a reference molecule difficult and 2) the charges assigned to neighbouring atoms must be consistent. In particular, the total charge on the molecule must be integer. In the recent years, a number of machine learning approaches emerged that infer charges based on a set of reference molecules [4, 13, 16]. However, these approaches often struggle to deal with the ambiguity of similar groups that have different charges in different molecules and the requirement that the overall charge must be integer.

In this paper, we consider the problem of – given a large set of reference molecules with known charge distributions – how to efficiently, automatically and optimally assign partial atomic charges which are consistent with both the neighbouring atoms and the total charge. As a reference we have used molecules parameterised using the Automated Topology Builder (ATB) and repository [15]. The ATB contains a large number of molecules (< 50 atoms) for which partial charges have been assigned *de novo*. In previous work, we have contributed to improving the reliability of this repository by ensuring the consistency and utility of the partial charges assigned to atoms by identifying atoms that could be used to form *charge groups*, which can be collectively assigned integer formal charges $(\ldots, -1, 0, 1, \ldots)$ [5]. We have also developed methods to match molecular substructures, taking into account that the partial charge of an atom is heavily dependent on its neighbours and the nature of its local chemical environment [7]. This made it possible to study the distribution of charges within local molecular environments for all molecules in the ATB ($\approx$ 200,000 molecules; 7,800,000 atoms and 5,600,000 bonds) and to find, given a query molecule, all possible matching fragments (sub-graphs).

Here we build on this previous work and our ability to match sub-fragments of a query molecule against the available database, to consider how the information contained in already parameterised molecules can be used best to infer the charges within a novel molecule. The most direct approach would be to simply use the mean partial charge on individual atoms identified as equivalent using a given similarity criterion. However, quantum mechanics dictates that the total charge on a molecule must be integer. Simply attributing to each atom the value of the mean partial charge from the known distribution fails as it results in the accumulation of errors and a total charge deviating from the required value.

Instead, we have considered solutions to the *molecular charge assignment problem*, which allow charges that deviate from the mean to be selected while their sum is constrained to lie close to a target total charge. Among the possible set of solutions we prefer those that maximize a score that depends on the observed frequencies of the chosen charges. We show that the problem is similar to a *multiple-choice knapsack problem* (MCKP) [6, 14]. We introduce ϵ-MCKP, a variant of the standard MCKP with an error margin ϵ. We provide an Integer Linear Programming (ILP) formulation of ϵ-MCKP and adapt the MCKP pseudo-polynomial Dynamic Programming (DP) algorithm to ϵ-MCKP. Finally, we demonstrate the utility of the ϵ-MCKP approach for solving the charge assignment problem by comparing the charges proposed based on this approach to those obtained directly using the ATB. We find that the charges computed with our novel approach are comparable to the ones obtained using explicit charge calculations while decreasing the time to determine partial charges for a molecule by several orders of magnitude, that is, from hours or even days to below a second.

Our code is publicly available at `https://github.com/enitram/charge_assign` under the Apache 2.0 open source license.

2 Assigning charges

We consider molecules as graphs. Let $G = (V, E, t)$ be a *molecular graph*, where vertices V correspond to atoms, edges E correspond to bonds and $t : V \to \Sigma$ colors vertices with atom types. A straightforward alphabet of atom types Σ would be the chemical elements. In this work, we used ATB-assigned GROMOS atom types which provide a more detailed classification of some chemical elements (N, C, O, S) depending on their hybridization (number of bonded nodes), and therefore provides a more detailed description of the local environment.

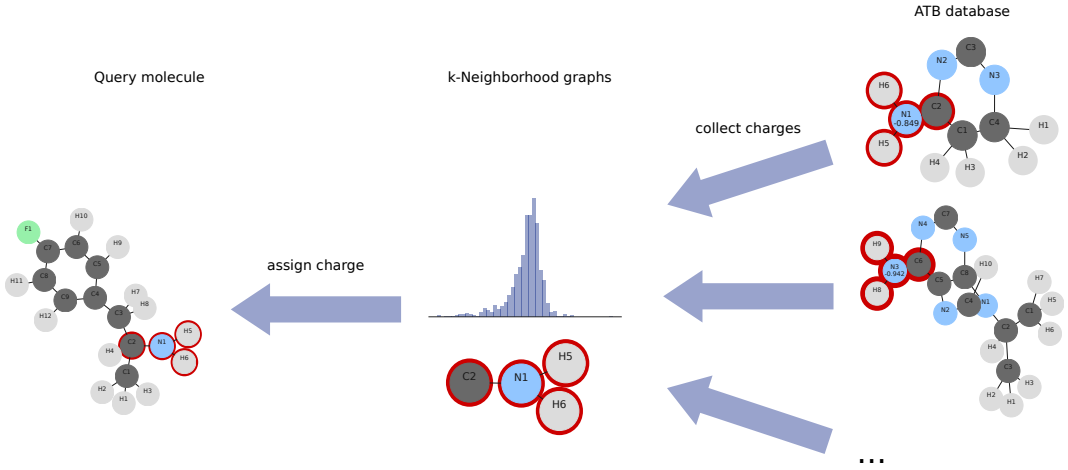


Figure 1 Given a query molecule, our method assigns atomic partial charges based on matching isomorphic subgraphs (red) with a known partial charge distribution collected from the ATB database of parameterised molecules.

The partial charge of an atom is heavily dependent on its bonded neighbours and the nature of its local environment. Formally, we define the k-neighbourhood as:

▶ **Definition 1** (k-neighbourhood)**.** Let $N(v) = \{u \mid (u, v) \in E\}$ be the neighbourhood of an atom v. We define the *k-neighbourhood* recursively as $N_k(v) = N_{k-1}(v) \cup \bigcup_{u \in N_{k-1}(v)} N(u)$, with $N_0(v) = v$.

Informally, the k-neighbourhood of an atom v is the set of all atoms for which a path of length $\leq k$ to v exists. Let $G[N_k(v)]$ be the subgraph induced by the k-neighbourhood of v.

To collect all possible partial charge values, we consider all k-neighbourhoods in the set of previously parameterised molecules. For this we iterate over all atoms v of all molecular graphs in the ATB and construct a list of subgraphs $G[N_k(v)]$ with associated partial charges of the corresponding atom v. We construct a database with an entry for each isomorphism class in the subgraph list. For each isomorphism class we collect the partial charges of its subgraphs and condense the values to a histogram. Since the point charges assigned by the ATB are rounded to three digits after the decimal point, we round the partial charge values accordingly.

Given a query molecule with a known target total charge, the challenge is to assign the most representative partial charge to each atom while staying close to the target total charge (Fig. 1). For that purpose, we iterate over all atoms of the query molecular graph and generate the subgraphs $G[N_k(v)]$. We match each subgraph to its isomorphism class in our database of k-neighbourhood subgraphs. If there is no match, we iteratively retry with $G[N_{k-1}(v)]$ until $k = 0$. Now each atom in our query molecule has a histogram of possible partial charges. The task is now to assign the charges such that we maximize the frequencies of the assigned charges while the sum of assigned partial charges equals the target charge with some error margin.

3 Problem Formulation and Complexity

We map each atom i to a set of items j with weights $w_{i,j}$ corresponding to partial charges and profits $p_{i,j}$ corresponding to their frequency-based scores. The target total charge corresponds to capacity c. Note that the charge assignment problem is now similar to a multiple-choice knapsack problem (MCKP). The decision version of MCKP is defined as:

▶ **Problem 1** (MCKP). *Given a decision variable $K \geq 0$, capacity $c \geq 0$, m sets $N_1, \ldots, N_m$ of items $j \in N_i$ with profit $p_{i,j} \geq 0$ and weight $w_{i,j} \geq 0$, select exactly one item from each set, such that the sum of weights of the selected items does not exceed c and the sum of profits of the selected items is equal or larger than K.*

MCKP is known to be weakly $\mathcal{NP}$-complete [6, 9, 14]. However, although the problem of assigning charges is similar to MCKP, there are two differences. First, weights and capacity can be negative numbers. Second, the sum of weights of selected items must hit the capacity with some error margin, resulting in an upper and lower capacity limit. We define a variant of MCKP, which is equivalent to the charge assignment problem as:

▶ **Problem 2** (ϵ-MCKP). *Given a decision variable $K \geq 0$, capacity $-\infty \leq c \leq \infty$, error $\epsilon \geq 0$, m sets $N_1, \ldots, N_m$ of items $j \in N_i$ with profit $p_{i,j} \geq 0$ and weight $-\infty \leq w_{i,j} \leq \infty$, select exactly one item from each set, such that the sum of weights of the selected items is in the range $[c - \epsilon, c + \epsilon]$ and the sum of profits of the selected items is equal or larger than K.*

▶ **Theorem 2.** *ϵ-MCKP is weakly $\mathcal{NP}$-complete.*

Proof. Showing that ϵ-MCKP is in $\mathcal{NP}$ is straightforward. Given an instance of ϵ-MCKP and a candidate solution $\hat{S}$, we can easily check in polynomial time whether $c - \epsilon \leq \sum_{w_{i,j} \in \hat{S}} w_{i,j} \leq c + \epsilon$ and $\sum_{p_{i,j} \in \hat{S}} p_{i,j} \geq K$ as well as if $\hat{S}$ contains exactly one item from each set $N_1, \ldots, N_m$. We show that ϵ-MCKP is weakly $\mathcal{NP}$-hard as follows: We reduce MCKP $\leq_{\mathrm{P}}$ ϵ-MCKP. Given an instance of the standard MCKP with capacity c, we transform it to an ϵ-MCKP instance with capacity $c' = \frac{1}{2}c$ and $\epsilon = \frac{1}{2}c$. Then, $c' - \epsilon = 0$ and $c' + \epsilon = c$, making both instances equivalent. ◀

Both problems obviously can be transformed into optimization problems by omitting the decision variable K and maximizing the sum of profits. The definition of ϵ-MCKP allows us to solve the charge assignment problem.

4 Solving ϵ-MCKP

In this section we present two algorithmic strategies to solve ϵ-MCKP: the first is based on an integer linear programming (ILP) formulation, which can be solved by general ILP solvers, while the second is a purely combinatorial dynamic programming (DP) algorithm.

Formulating ϵ-MCKP as an ILP is straightforward. Let $x_{i,j}$ be a binary variable with value 1 if and only if item j in set N_i is selected. We formulate the problem as:

$$\max \quad \sum_{i=1}^{m} \sum_{j \in N_i} x_{i,j} p_{i,j} \tag{1a}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \sum_{j \in N_i} x_{i,j} w_{i,j} \geq c - \epsilon \tag{1b}$$

$$\sum_{i=1}^{m} \sum_{j \in N_i} x_{i,j} w_{i,j} \leq c + \epsilon \tag{1c}$$

$$\sum_{j \in N_i} x_{i,j} = 1 \qquad \text{for } 1 \leq i \leq m \tag{1d}$$

$$x_{i,j} \in \{0, 1\} \qquad \text{for } 1 \leq i \leq m, j \in N_i \tag{1e}$$

The second algorithm is an adaption of the pseudo-polynomial DP of the standard MCKP to ϵ-MCKP. The standard MCKP assumes numbers to be non-negative integers. If a given

ϵ-MCKP instance does not comply with the non-negativity and integrality constraints, we transform the instance as follows:

First, we convert floating point weights $w_{i,j}$, capacity c and error ϵ to integers by multiplying with an appropriate factor. Since point charges in this work are rounded to three digits after the decimal point, a factor of 10^3 is sufficient. Second, we transform the weights $w_{i,j}$ and capacity c to non-negative numbers. For every set N_j with $j = 1, \ldots, m$, we determine the minimum weight $w_i^* = \min_{j \in N_i} w_{i,j}$. We define the new weights as $\tilde{w}_{i,j} = w_{i,j} - w_i^*$. Then, the weights are guaranteed to be non-negative. As we have to select one item per set, we can define the new capacity as $\tilde{c} = c - \sum_{i=1}^{m} w_i^*$.

Therefore, we assume in the following (without loss of generality) that weights $w_{i,j}$, capacity c and error ϵ are non-negative integers. Let P be a two-dimensional DP-table of size $m \times (c + \epsilon)$. $P[k, d]$ holds the maximum profit that we can achieve with sets 0 to k and a sum of weights of exactly d:

$$P[k,d] = \max \left\{ \sum_{i=0}^{k} \sum_{j \in N_i} x_{i,j} p_{i,j} : \sum_{i=0}^{k} \sum_{j \in N_i} x_{i,j} w_{i,j} = d, \sum_{j \in N_k} x_{i,j} = 1 \text{ for all } 0 \leq i \leq k \right\} \quad (2)$$

We compute P recursively. Let $P[k, d]$ be defined as:

$$P[k,d] = \max \begin{cases} P[k-1, d - w_{k,j}] + p_{k,j} & \text{for } j \in N_i \text{ and } d - w_{k,j} \geq 0 \\ -\infty \end{cases} \quad (3)$$

$P[k, d]$ is calculated by considering all items of the current set N_k and computing the maximum profit that can be achieved when adding those profits to possible previous solutions with $k - 1$ sets and sum of weights $d - w_{k,j}$. The profit is $-\infty$ if there is no possible solution for $P[k, d]$. Contrary to the standard MCKP DP we initialize P as:

$$P[0,d] = \begin{cases} 0 & \text{if } d = 0 \\ -\infty & \text{otherwise} \end{cases} \quad (4)$$

This ensures that only solutions in which the sum of selected weights equals exactly d are possible. We find the maximum profit p^* by:

$$p^* = \max \{ P[m,d] : \max\{c - \epsilon, 0\} \leq d \leq c + \epsilon \} \quad (5)$$

The DP can be easily implemented using one dimension, as the recursion only looks back one step in the dimension k (the number of sets we currently consider). The space requirement of the DP algorithm is $O(c + \epsilon)$. The running time complexity is $O(n(c + \epsilon))$, with n being the total number of items.

5 Score

We modeled the charge assignment problem as ϵ-MCKP, where atoms i are sets of items j with weight $w_{i,j}$ corresponding to partial charges and profits $p_{i,j}$ corresponding to their scores. In this section we propose a frequency-based score for the ϵ-MCKP profit maximization.

Figure 2 shows the distribution of charges over all 3-neighbourhood graphs in a snapshot of the ATB of roughly 160,000 molecules centered at the sample mean of each 3-neighbourhood graph. At a first glance, it may seem to be normally distributed, but the Q-Q-plot on the right hand side of Figure 2 reveals that the distribution is heavy-tailed. Therefore, using measures that assume normally distributed data such as the z-score is not advisable. We also

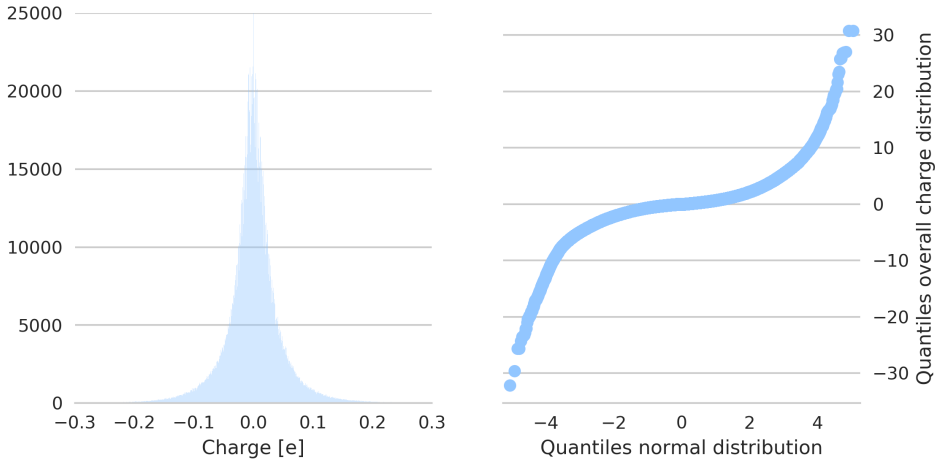


Figure 2 Distribution of charges over all 3-neighbourhood graphs centered at the sample mean of each 3-neighbourhood graph (left) and Q-Q-plot with the quantiles of the charge distribution over all 3-neighbourhood graphs on the y-axis and the quantiles of a fitted normal distribution on the x-axis.

refrain from simply using the logarithm of the frequencies as our score, since the deviation of the sample mean of the observed charges should also be taken into account. Additionally, the logarithm will result in negative profits.

We propose a simple score using squared distances. Let $f_{i,j}$ be the observed frequency of partial charge $w_{i,j}$ and $\hat{\mu}_i$ the sample mean of all observed charges of atom i. We define the score $p_{i,j}$ as

$$p_{i,j} = \frac{f_{i,j}}{1 + (w_{i,j} - \hat{\mu}_i)^2}. \tag{6}$$

The score reflects both the observed frequency of a partial charge and its distance to the observed mean of all partial charges of an atom. If the charge equals the mean $w_{i,j} = \hat{\mu}_i$, then the score equals its observed frequency $p_{i,j} = f_{i,j}$. The larger the distance of a charge to the mean is, the smaller the score will be. This serves as a tie-breaker, such that if two charges have the same observed frequency and are within the capacity limits, ϵ-MCKP will prefer the charge closer to the observed mean.

6 Results and Discussion

To evaluate our method, we conducted a leave-one-out-analysis using a snapshot of the ATB database containing roughly 160,000 molecules. We focus on this set of previously computed molecules, since the computational effort of large-scale quantum-mechanical calculations is significant. We created a database of k-neighbourhood subgraphs associated with partial charge histograms with variable bin widths and a fixed $k = 3$. Bin widths were determined according to the Friedman-Diaconis rule [8]. Then, we temporarily removed all charge values associated with its 3-neighbourhood subgraphs for each molecule and computed the atomic partial charges using the new, smaller histogram database. We compared the assigned values to the original atomic partial charges in the ATB database.

All computations were performed on a compute cluster with 16 3.2 GHz Xeon CPUs and 512GB RAM. The ILP was solved using COIN-OR [17]. We recorded the running times of the ILP and DP algorithm, see Fig. 3. As expected, the running time of the DP scales

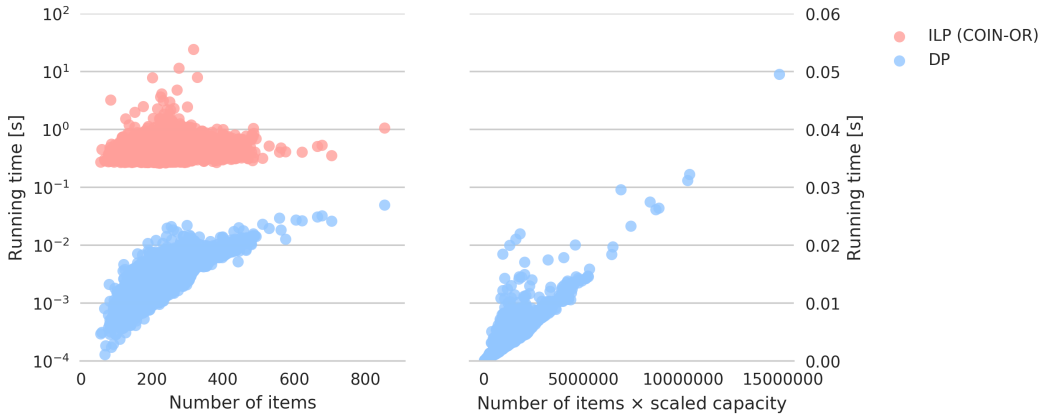


Figure 3 Running times of the ILP solved with COIN-OR and the DP dependent on the number of items n, showing that the DP is significantly faster than the ILP (left). The running time of the DP actually depends on the number of items times the scaled capacity $10^3 \cdot n \cdot (\tilde{c} + \epsilon)$ (right).

linearly with $10^3 \cdot n \cdot (\tilde{c} + \epsilon)$, where n is the number of items, 10^3 is the blowup factor and $\tilde{c}$ is the capacity c transformed to non-negativity with $\tilde{c} = c - \sum_{i=1}^{m} w_i^*$ and $w_i^* = \min_{j \in N_i} w_{i,j}$. The running times of the ILP show more variation and a marginal positive correlation to the number of items n, which equals the number of variables in the ILP. The DP was always significantly faster than the ILP in the leave-one-out-evaluation.

In the leave-one-out evaluation, we compared the naive approach of estimating the atomic partial charges by simply taking the mean and our method of solving an ϵ-MCKP instance, see Fig. 4. As expected, while the naive method on average is able to find charges with a slightly lower distance to the original partial charges, it often results in a total charge far away from the target total charge (with errors more than 1e in many cases). Our method on the other hand is able to assign charge values which are only slightly worse than the ones computed by the naive method while achieving a total charge close to the target total charge. As can be seen in Fig. 4 the deviation of the total charge from the target charge using the ϵ-MCKP approach is so small it is barely visible on the scale used.

As an example of the charges assigned by our method, Fig. 5 shows the two molecules with the atomic partial charges that are on average closest to and farthest from the original ATB charges. The computed charges for the molecule with the closest distance fit well to the original ATB charges. ϵ-MCKP assigns identical charges to atoms H1, H2 and H3. The 3-neighbourhood graphs of all three atoms have the same isomorphism class. This is an advantage of our ϵ-MCKP approach, since quantum-mechanical *de novo* charge assignment does not guarantee that similar charges are assigned to equivalent atoms (although in this case the ATB charges are also identical). For the molecule with the farthest distance there are some large distances of more than 1e. However, we observe that the large distances are caused by the original ATB charges being on the outer edges of the charge distributions, while ϵ-MCKP on the other hand picks charges close to the largest mode of the distribution, see bottom side of Fig. 5. Note that Fig. 5 shows the distributions used in the leave-one-out evaluation without the original ATB charges of the depicted molecule. Additionally, the charge distributions of atoms with large charge distances have been computed with a low number of observed charges, resulting in multimodal distributions with several large peaks. We expect this effect to disappear when more data is available in the constantly growing ATB repository.

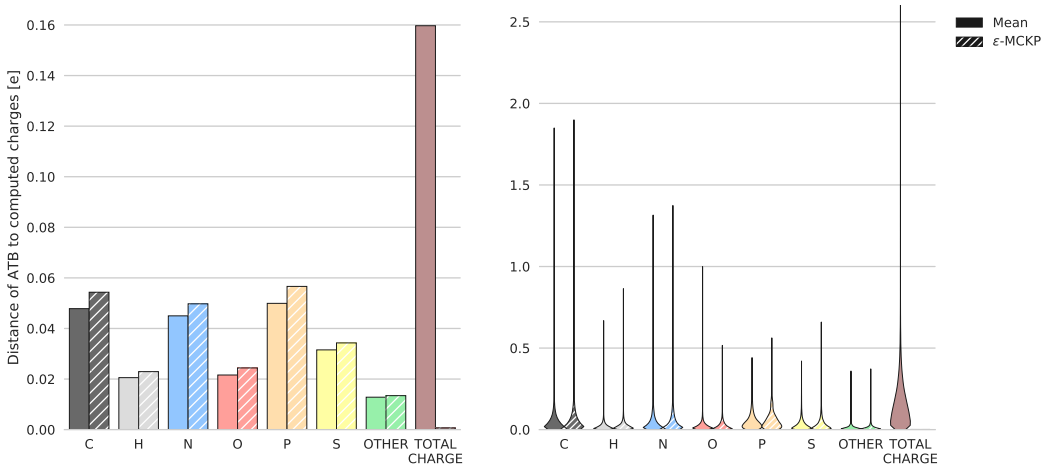


Figure 4 Results of the leave-one-out experiment with $k = 3$ showing mean distances in elementary charge units (left) and violin plots of all distances (right) of original charges found in the ATB to charges calculated by selecting the mean and solving ϵ-MCKP. The distances are categorized by chemical elements. For ϵ-MCKP, the computed total charges are virtually the same as the target total charges from the ATB, resulting in mean distances of almost zero.

Fig. 6 shows the chemical structure of a more complex example (ATB ID 25338). For this molecule, the *de novo* electrostatic-potential based charge assignment using quantum-mechanical computations required $\sim$ 140 days using on one core while solving our ϵ-MCKP approach was finished in $\sim$ 0.12 (ILP) and $\sim$ 0.06 (DP) seconds.

Most of the outer atoms – especially the hydrogens – showed a narrow unimodal distribution of ATB charges and ϵ-MCKP picked charges close to the original ATB charge. The more buried atoms showed a higher variability. For some atoms, we observed a similar behavior as in Fig. 5, that ϵ-MCKP selects a charge closer to the distribution mean than the original ATB charge was. However, for several atoms we observed the limit of our data-driven approach. If only a few charge values are available for a certain k-neighbourhood, then the distributions are multimodal with very similar or equal peak heights, reflecting the variability of the quantum-mechanically derived charges. Then, ϵ-MCKP may freely choose between co-optimal solutions. On the other hand, if only exactly one charge value is available, ϵ-MCKP has to choose this value. While the probability of this occurring will decrease with the addition of more data, in this case (with the current dataset) it would be advisable to use a smaller k. With choosing an appropriate k, the user may balance the specificity of large k-neighbourhoods against the robustness of small k-neighbourhoods.

In general, charges on the outer atoms of a molecule can be assigned quite well while charges of the inner atoms deviate more from the ATB charges. This may be explained by the higher variability of the inner atoms in the ATB dataset, an artifact of the *de novo* electrostatic-potential based charge assignment [2].

7 Conclusions

The ability to accurately calculate the electrostatic interactions between a ligand and its receptor is a key component of computer-aided drug development. In this paper, we have investigated the problem of automatically assigning partial charges. The charge assignment problem is similar to the multiple-choice knapsack problem. We introduced a variant tailored

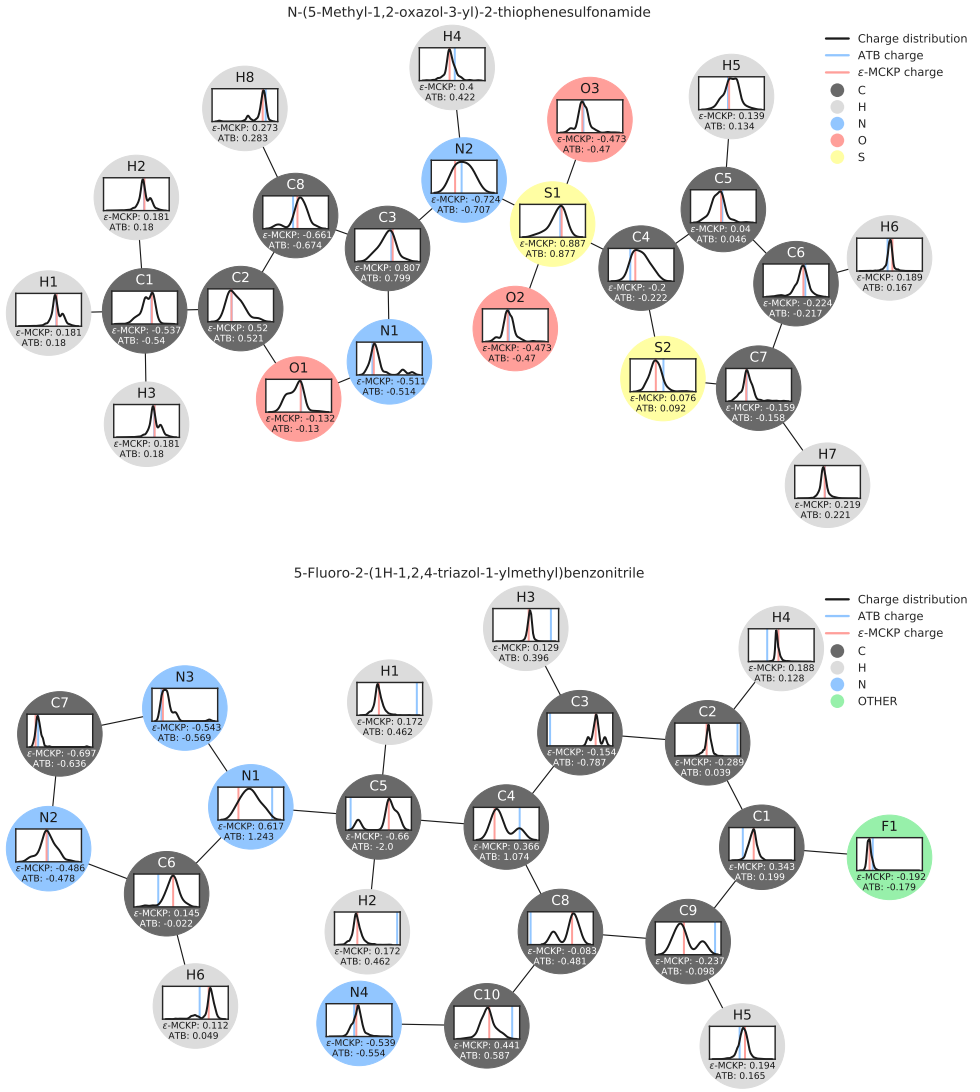


Figure 5 Best (top) and worst (bottom) molecules in the leave-one-out evaluation ranked by average distance of the original ATB charges to the charges computed by ϵ-MCKP. Atoms (nodes) are color-coded by their chemical element (red for oxygen, blue for nitrogen, black for carbon, yellow for sulfur and grey for hydrogen). Atoms are overlayed with the kernel-density estimate of the histograms of the leave-one-out charges of their respective 3-neighbourhoods. The original ATB charge and the computed charges are shown by blue and red vertical lines in the histograms.

to the charge assignment problem, the ϵ-multiple-choice knapsack problem (ϵ-MCKP). Like most knapsack problems, ϵ-MCKP is weakly $\mathcal{NP}$-complete. We presented two algorithmic solutions to ϵ-MCKP, an integer linear programming (ILP) formulation and a dynamic programming (DP) algorithm.

We conducted a leave-one-out evaluation on a snapshot of the ATB database. The computed atomic partial charges were close to the original ATB charges and the total charge virtually the same as the target total charge, suggesting that our method provides consistent parameters for MD simulations, docking studies and other related applications.

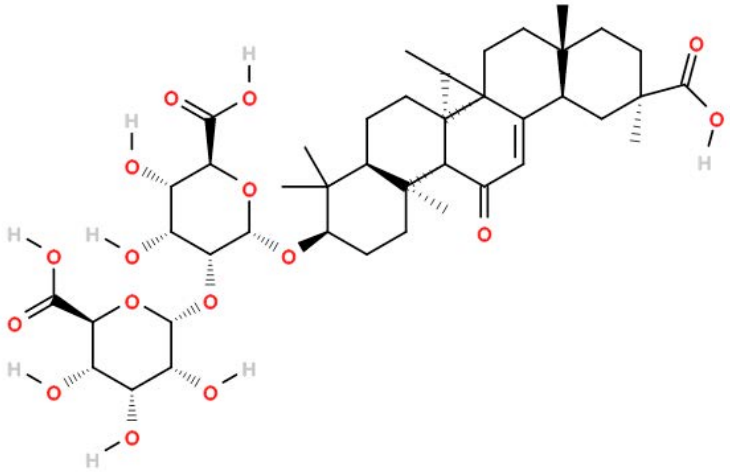


Figure 6 The chemical structure of ATB ID 25338 containing 120 atoms. Note that aliphatic hydrogens are not shown.

One additional advantage of our approach is that equivalent nodes in the graph will be assigned similar charges and the charge distribution will therefore mirror the symmetry of the molecular graph.

The DP algorithm performed faster than the ILP on a set of 160,000 molecules contained within the ATB. On average, both implementations required only a fraction of a second to assign charges to molecules containing 50-100 atoms, while quantum-mechanical computations required many days. This is important when screening large molecular databases. For instance, ChEMBL [10], a manually curated chemical database of bioactive molecules with drug-like properties, contains in excess of 1.6 million compounds. The majority of these have more than 50 atoms making quantum-mechanical computations difficult. Other computational drug design databases are larger again [20]. For example, ZINC, a free database of commercially-available compounds, contains more than 35 million compounds [12].

Our method builds on a repository of previously computed molecular parameters and assigns consistent partial atomic charges in a swift manner to facilitate MD simulations and related applications in drug design.

References

1 Robert Abel, Lingle Wang, David L. Mobley, and Richard A. Friesner. A critical review of validation, blind testing, and real-world use of alchemical protein-ligand binding free energy calculations. *Current Topics in Medicinal Chemistry*, 17(23):2577–2585, 2017. doi:10.2174/1568026617666170414142131.

2 Christopher I. Bayly, Piotr Cieplak, Wendy Cornell, and Peter A. Kollman. A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges:

the RESP model. *The Journal of Physical Chemistry*, 97(40):10269–10280, 1993. doi:10.1021/j100142a004.

3 F. Matthias Bickelhaupt and Evert Jan Baerends. Kohn-Sham Density Functional Theory: Predicting and Understanding Chemistry. In Kenny B. Lipkowitz and Donald B. Boyd, editors, *Reviews in Computational Chemistry*, pages 1–86. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2007. doi:10.1002/9780470125922.ch1.

4 Patrick Bleiziffer, Kay Schaller, and Sereina Riniker. Machine Learning of Partial Charges Derived from High-Quality Quantum-Mechanical Calculations. *Journal of Chemical Information and Modeling*, 58(3):579–590, 2018. doi:10.1021/acs.jcim.7b00663.

5 Stefan Canzar, Mohammed El-Kebir, René Pool, Khaled Elbassioni, Alpeshkumar K. Malde, Alan E. Mark, Daan P. Geerke, Leen Stougie, and Gunnar W. Klau. Charge Group Partitioning in Biomolecular Simulation. *Journal of Computational Biology*, 20(3):188–198, 2013. doi:10.1089/cmb.2012.0239.

6 Krzysztof Dudziński and Stanisław Walukiewicz. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28(1):3–21, 1987. doi:10.1016/0377-2217(87)90165-2.

7 Martin S. Engler, Mohammed El-Kebir, Jelmer Mulder, Alan E. Mark, Daan P. Geerke, and Gunnar W. Klau. Enumerating common molecular substructures. *PeerJ Preprints*, 5:e3250v1, 2017. doi:10.7287/peerj.preprints.3250v1.

8 David Freedman and Persi Diaconis. On the histogram as a density estimator: L_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 57(4):453–476, 1981. doi:10.1007/BF01025868.

9 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

10 Anna Gaulton, Louisa J. Bellis, A. Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and John P. Overington. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40(D1):D1100–D1107, 2012. doi:10.1093/nar/gkr777.

11 Alexander Hillisch, Nikolaus Heinrich, and Hanno Wild. Computational chemistry in the pharmaceutical industry: From childhood to adolescence. *ChemMedChem*, 10(12):1958–1962, 2015. doi:10.1002/cmdc.201500346.

12 John J. Irwin and Brian K. Shoichet. ZINC - a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005. doi:10.1021/ci049714+.

13 Maxim V. Ivanov, Marat R. Talipov, and Qadir K. Timerghazin. Genetic Algorithm Optimization of Point Charges in Force Field Development: Challenges and Insights. *The Journal of Physical Chemistry A*, 119(8):1422–1434, 2015. doi:10.1021/acs.jpca.5b00218.

14 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer Berlin, Berlin, 1. edition, 2004.

15 Alpeshkumar K. Malde, Le Zuo, Matthew Breeze, Martin Stroet, David Poger, Pramod C. Nair, Chris Oostenbrink, and Alan E. Mark. An Automated Force Field Topology Builder (ATB) and Repository: Version 1.0. *Journal of Chemical Theory and Computation*, 7(12):4026–4037, 2011. doi:10.1021/ct200196m.

16 Brajesh K. Rai and Gregory A. Bakken. Fast and accurate generation of ab initio quality atomic charges using nonparametric statistical regression. *Journal of Computational Chemistry*, 34(19):1661–1671, 2013. doi:10.1002/jcc.23308.

17 Matthew J. Saltzman. COIN-OR: An open-source library for optimization. In Søren S. Nielsen, editor, *Programming Languages and Systems in Computational Economics and Finance*, pages 3–32. Springer, Boston, MA, 2002. doi:10.1007/978-1-4615-1049-9_1.

18 Bradley Sherborne, Veerabahu Shanmugasundaram, Alan C. Cheng, Clara D. Christ, Renee L. DesJarlais, Jose S. Duca, Richard A. Lewis, Deborah A. Loughney, Eric S. Manas, Georgia B. McGaughey, Catherine E. Peishoff, and Herman van Vlijmen. Collaborating to improve the use of free-energy and other quantitative methods in drug discovery. *Journal of Computer-Aided Molecular Design*, 30(12):1139–1141, 2016. doi:10.1007/s10822-016-9996-y.

19 U. Chandra Singh and Peter A. Kollman. An approach to computing electrostatic charges for molecules. *Journal of Computational Chemistry*, 5(2):129–145, 1984. doi:10.1002/jcc.540050204.

20 Johannes H. Voigt, Bruno Bienfait, Shaomeng Wang, and Marc C. Nicklaus. Comparison of the NCI Open Database with Seven Large Chemical Structural Databases. *Journal of Chemical Information and Computer Sciences*, 41(3):702–712, 2001. doi:10.1021/ci000150t.

ℓ_1-Penalised Ordinal Polytomous Regression Estimators with Application to Gene Expression Studies

Stéphane Chrétien
National Physical Laboratory, Hampton Road, Teddington, United Kingdom
stephane.chretien@npl.co.uk

Christophe Guyeux
Computer Science Department, FEMTO-ST Institute, UMR 6174 CNRS, Université de Bourgogne Franche-Comté, 16 route de Gray, 25030 Besançon, France
christophe.guyeux@univ-fcomte.fr

Serge Moulin
Computer Science Department, FEMTO-ST Institute, UMR 6174 CNRS, Université de Bourgogne Franche-Comté, 16 route de Gray, 25030 Besançon, France
serge.moulin@univ-fcomte.fr

Abstract

Qualitative but ordered random variables, such as severity of a pathology, are of paramount importance in biostatistics and medicine. Understanding the conditional distribution of such qualitative variables as a function of other explanatory variables can be performed using a specific regression model known as ordinal polytomous regression. Variable selection in the ordinal polytomous regression model is a computationally difficult combinatorial optimisation problem which is however crucial when practitioners need to understand which covariates are physically related to the output and which covariates are not. One easy way to circumvent the computational hardness of variable selection is to introduce a penalised maximum likelihood estimator based on some well chosen non-smooth penalisation function such as, e.g., the ℓ_1-norm. In the case of the Gaussian linear model, the ℓ_1-penalised least-squares estimator, also known as LASSO estimator, has attracted a lot of attention in the last decade, both from the theoretical and algorithmic viewpoints. However, even in the Gaussian linear model, accurate calibration of the relaxation parameter, *i.e.*, the relative weight of the penalisation term in the estimation cost function is still considered a difficult problem that has to be addressed with caution. In the present paper, we apply ℓ_1-penalisation to the ordinal polytomous regression model and compare several hyper-parameter calibration strategies. Our main contributions are: (a) a useful and simple ℓ_1 penalised estimator for ordinal polytomous regression and a thorough description of how to apply Nesterov's accelerated gradient and the online Frank-Wolfe methods to the problem of computing this estimator, (b) a new hyper-parameter calibration method for the proposed model, based on the QUT idea of Giacobino et al. and (c) a code which can be freely used that implements the proposed estimation procedure.

2012 ACM Subject Classification Mathematics of computing → Regression analysis

Keywords and phrases LASSO, ordinal polytomous regression, Quantile Universal Threshold, Frank-Wolfe algorithm, Nesterov algorithm

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.17

Funding Computations have been performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

1 Introduction

Ordinal polytomous variables are of paramount importance in bioinformatics where practitioners may have to tackle qualitative but ordered data such as, e.g., the severity of a certain type of cancer [23], [24], [12], [13], [14], etc. Understanding how such variables can be explained by other variables such as, e.g., gene expressions, can help the research community investigate the influence of certain genes in the pathology under study. Oftentimes, only a small number of genes are relevant to the statistical modelling and variable selection needs to be performed in order to detect which of them should be ignored and which of them should not. The ordinal polytomous regression model is an adaptation of the classical regression model which is extremely well suited for this type of problem, and the goal of the present paper is to propose efficient approaches to the estimation and variable selection problems for this specific model.

1.1 When the number of covariates exceeds the number of observations: the blessing of sparsity

One important additional problem in standard gene expression studies is that the number of observations (e.g. patients) is often much smaller than the number of covariates (e.g. genes). In such cases, the problem cannot be expected to be solvable without some additional structure because the number of unknowns is larger than the number of observations. The main structural assumption which is usually made in such cases is that some sparsity property holds. In the example of gene expression analysis, it is usually considered natural to assume that only a small number of genes have significant influence on the output under study. Therefore, only a small number of regression coefficients should be nonzero in the estimator, although we cannot know before hand which are the ones which should be selected. Selecting the right variables in regression is often called "support recovery". Various approaches to variable selection have been proposed in the statistical literature. In practical applications, the most extensively used selection methods are the forward selection and the AIC/BIC information criteria based approaches [2], [22] [19]. Such methods however, can hardly be applied in situations where the number of covariates, e.g. genes, is large and one usually resorts to convex optimisation based strategies such as the LASSO [23] and its generalisations to nonlinear models [24], [25].

1.2 Previous work on variable selection via ℓ_1-norm penalisation

Convex optimisation based variable selection approaches are often based on penalised log-likelihood estimation, where the penalisation term is the ℓ_1-norm. In the linear model, it was discovered in [7] that under certain specific properties of the design matrix, known as the Restricted Isometry Property, the ℓ_1-norm penalised least ℓ_∞ estimator, aka the Dantzig estimator, would recover the location of the non-zero components exactly. This type of result, was then proven for the ℓ_1-penalised least ℓ_2 estimator, aka the LASSO estimator under weaker assumptions, including incoherence of the design matrix in [8]. The work [6] provided interesting alternative views on the statistical properties of the LASSO and Dantzig estimators which are still extensively used in the current literature on this topic.

Even when neither the Restricted Isometry Property nor the incoherence assumptions are satisfied, the mere computational tractability of ℓ_1-penalisation based estimators makes them the method of choice when the problem size is large.

1.3 The problem of hyper-parameter calibration

The main advantage of ℓ_1-based penalisation is to reduce the estimation problem to a convex optimisation one if the hyper-parameter, *i.e.* the relative weight associated with the ℓ_1-penalisation term, is calibrated to an appropriate value. In practice however, finding the right value for this hyper-parameter is often a complicated issue.

Most theoretical works come up with a formula for the hyper-parameter, see e.g. [8]. Such types of results are very important because they prove existence of a value of the hyper-parameter that will allow exact support recovery of the sparse regression vector under appropriate, e.g. incoherence assumptions of the design matrix. The theoretical value often gives the right order of dependencies with respect to the dimension of the problem, the standard deviation of the noise, and other important structural parameters, and is therefore a good indicator of how well conditioned the problem is, at least in theory.

In practice, however, the noise level is not known beforehand and therefore, hyper-parameter calibration cannot be performed without joint variance estimation. Reference [10] presents efficient methods for solving this joint estimation/calibration problem and present preliminary computational experiments showing practical relevance of the overall approach. The square-root LASSO [5] is another interesting alternative but is sometimes reported to have slightly worse performance in practice.

The usually preferred practical approach to hyper-parameter calibration is Cross Validation [3]. The Cross-Validation approach is very intuitive and had nice theoretical properties when the number of covariates is smaller than the number of observations. A drawback of Cross-Validation is the computational burden of re-sampling and computing the LASSO estimator a large number of times. Moreover, Cross-Validation is oriented towards prediction performance rather than accurate support selection. An alternative approach devised in [11], based on the Hedge algorithm of [16] and the stochastic Frank-Wolfe algorithm, was shown to outperform Cross Validation in terms of computational time for the linear model as well.

Recently, [17] devised a very efficient method called Quantile Universal Thresholding for hyper-parameter calibration in the linear model with a view towards efficient variable selection. Extensive numerical experiments provided in [17] show that Quantile Universal Thresholding outperforms Cross-Validation, although Cross-Validation has to be performed when the noise variance is unknown. Fortunately enough, recent work on fast variance estimation, as described e.g. in [18] or based on [11], should however allow to overcome the burden of using Cross-Validation as a subroutine in the Quantile Universal Thresholding procedure of [17].

1.4 Contributions of the paper

The main contributions of the present paper are threefold. The first is to present a ℓ_1-penalised maximum likelihood estimator for the ordered polytomous model and present efficient methods for computing this estimator. The second contribution is an efficient hyper-parameter calibration procedure based on recent work [17]. The last contribution is a freely available software implementation which can be downloaded online [1].

2 Methods

2.1 The model and the penalised estimator

2.1.1 The standard polytomous regression model

In the ordinal polytomous regression model, the independent qualitative output variables Y_i, $i = 1, \ldots, n$ with Q modalities $m_1, \ldots, m_Q$, are assumed to result from the quantification of a latent continuous variable $Y_i^* = X_i^t \beta^0 + \epsilon_i$, $i = 1, \ldots, n$, where X_i is a p-dimensional vector of covariates and where the residual ϵ_i has logistic cumulative distribution function $\Phi(y) = \frac{\exp(y)}{1+\exp(y)}$. More precisely, setting $-\infty = \gamma_0^0 < \cdots < \gamma_{Q-1}^0 < \gamma_Q^0 = +\infty$, we have $Y_i = m_q$ if and only if $Y_i^* \in]\gamma_{q-1}, \gamma_q]$. For $q = 1, \ldots, Q$, let us denote I_q the subset of {1...n} such that $i \in I_q$ if and only if $Y_i = m_q$. Let us denote by γ the vector $\gamma = (\gamma_1, \ldots, \gamma_{q-1})$. The conditional likelihood given $X_1, \ldots, X_n$ for this model is:

$$L_{Y|X}(\beta, \gamma) = \prod_{q=1}^{Q} \prod_{i \in I_q} \left(\Phi\left(X_i^t \beta - \gamma_{q-1}\right) - \Phi\left(X_i^t \beta - \gamma_q\right) \right). \tag{1}$$

where X is the $n \times p$ matrix such that X_i is its i^{th} row for all i in 1, ..., n.

The conditional log-likelihood is given by

$$l_{Y|X}(\beta, \gamma) = \sum_{i=1}^{n} \sum_{q=1}^{Q} 1_{\{Y_i = m_q\}} \log\left(\Phi\left(X_i^t \beta - \gamma_{q-1}\right) - \Phi\left(X_i^t \beta - \gamma_q\right) \right),$$

The parameters of this model are usually estimated using the maximum likelihood principle, *i.e.*, by finding the vector $(\hat{\beta}, \hat{\gamma})$ that maximizes $l_{Y|X}$. Maximization of the log-likelihood is made easy by the well known fact that the conditional log-likelihood function is concave.

The problem with this approach is that it cannot work when p is larger than n because, in this case, the Hessian matrix is easily shown to be singular. The situation where p is larger than n is however frequent in gene expression analysis as in many other problems, and one needs an estimator which can perform variable selection in such settings with low computational complexity. The next section introduces such an estimator based on ℓ_1 penalisation.

2.1.2 The penalised maximum likelihood estimator

One estimator of choice for the type of problem we just described (*i.e.* ordinal polytomous regression) is the ℓ_1-penalised maximum likelihood estimator given by

$$(\hat{\beta}, \hat{\gamma}) \quad \in \quad \operatorname{argmax}_{(b,c) \in \mathbb{R}^p \times \mathbb{R}^{Q-1}} \; l_{Y|X}(b, c) - \lambda \, \|b\|_1, \tag{2}$$

where λ is a relaxation parameter. This estimator corresponds exactly to the LASSO in the case where the log-likelihood is the one of the linear model. The main motivation for introducing this estimator is Theorem 1.2 in [8] about the LASSO. This theorem states that for a sufficiently sparse β in the linear model $Y = X\beta + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, the risk of the LASSO estimator is near optimal, *i.e.* is comparable to the risk obtained with an oracle estimator which would know the support of β ahead of time. Moreover, support recovery is proved to hold with large probability for a vast majority of possible supports.

The assumptions in this theorem are the following:

1. X has low coherence, *i.e.* the maximum scalar product of two columns of X is less than $A_0 / \log(p)$;

2. the support and sign pattern of β have uniform distribution;
3. the nonzero components of β have magnitude above the noise level times a log factor.

It is therefore natural to expect that an appropriate translation of this result to the case of (ordinal or not) polytomous regression model will hold as well. In the sequel, we will present simulation based results on the penalised conditional likelihood estimator from the view point of variable selection.

2.2 Algorithms

2.2.1 Nesterov's algorithm

In [21, 20, 4], Nesterov introduced a new approach to convex minimization with possibly non-differentiable functions. Nesterov's method consists of smoothing the non-differentiable function and then applying a refined first order scheme to the problem. The main interest of this approach is that at iteration k, a bound of $O(1/k^2)$ on the error is guaranteed, whereas standard gradient methods only guarantee $O(1/k)$. Let us now describe a simple version of this method.

The first step is to smooth the ℓ_1-norm function. Notice that, for the vector β, $\|\beta\|_1$ can be written

$$\|\beta\|_1 = \max_{\|u\|_\infty \leq 1} u^t \beta, \tag{3}$$

and the maximizer in this expression is simply $\text{sign}(\beta)$, where $\text{sign}(\beta)$ is the vector with the component-wise signs of β. A possible simple smoothing of the ℓ_1-norm is given by

$$\ell_{1,\mu}(\beta) = \max_{\|u\|_\infty \leq 1} u^t \beta - \frac{\mu}{2}\|u\|_2^2. \tag{4}$$

Notice that the maximizer u^*_β in (4) exists due to continuity and coercivity, and is unique due to the strict convexity of $\|\cdot\|_2^2$. The main interesting feature of this smoothing is the following proposition.

▶ **Proposition 2.1.** *The function $\ell_{1,\mu}$ is differentiable with Lipschitz gradient. Moreover, the gradient is given by*

$$\nabla \ell_{1,\mu} = u^*_\beta \tag{5}$$

where u^*_β is the unique maximizer in (4) and the Lipschitz constant of the gradient is $L_1 = 1/\mu$.

Proof. See [20, Theorem 1]. ◀

With this result in hand, we can present Nesterov's accelerated gradient algorithm for smooth optimisation in Algorithm 1 below. In order to implement the algorithm, one needs to know the Lipschitz constant of the gradient of minus the log-likelihood, which is unknown, and the Lipschitz constant of the smoothed ℓ_1-norm penalty, which is $1/\mu$. In practice, the Lipschitz constant of the gradient of minus the log-likelihood can be estimated by random sampling and computing ratio between the norm of the difference between gradients at sampled points and the norm of the difference of these sample points.

Algorithm 1 Nesterov's algorithm for penalised log-likelihood estimation.

Input An initial point $\theta^{(0)} = (\beta^{(0)}, \gamma^{(0)})$, e.g. $\theta^{(0)} = 0$, the relaxation coefficient λ, the Lipschitz constants L_0 (resp. L_1) of the gradient of - the log-likelihood (resp. of $\ell_{1,\mu}$) and the maximum number of iterations $N \in \mathbb{N}_*$

for $k = 0...N-1$ **do**

Compute $g^{(k)} = \nabla\left(-l(\theta^{(k)}) + \lambda \ell_{1,\mu}(\beta^{(k)})\right)$

Compute $\theta^{(k,1)}$:

$$\theta^{(k,1)} = \operatorname{argmin}_{\tau \in \mathbb{R}^{p+Q-1}} \langle g^{(k)}, \tau - \theta^{(k)} \rangle + \frac{L_0 + L_1}{2} \|\tau - \theta^{(k)}\|_2^2.$$

Compute $\theta^{(k,2)}$:

$$\theta^{(k,2)} = \operatorname{argmin}_{\tau \in \mathbb{R}^{p+Q-1}} \Big(\sum_{0 \le k' \le k} \frac{1}{2(k'+1)} g^{(k')}, \tau - \theta^{(k')} \rangle \Big) + \frac{L_0 + L_1}{2} \|\tau - \theta^{(0)}\|_2^2.$$

Update $\theta^{(k+1)}$:

$$\theta^{(k+1)} = \frac{k+1}{k+3}\, \theta^{(k,1)} + \frac{2}{k+3}\, \theta^{(k,2)}.$$

end for

Output $\widehat{\theta}^{(N)}$.

2.2.2 The Frank-Wolfe algorithm

The Frank–Wolfe (FW) algorithm, proposed by Marguerite Frank and Philip Wolfe in 1956 [15], is another convex optimisation algorithm. The difference between the FW algorithm and the Nesterov one is that FW applies to constrained optimisation.

The main trick that is needed to implement the Frank-Wolfe algorithm is to reformulate the penalised problem

$$(\hat{\beta}, \hat{\gamma}) \in \operatorname{argmax}_{(b,c) \in \mathbb{R}^p \times \mathbb{R}^{Q-1}} \; l_{Y|X}(b,c) - \lambda \, \|b\|_1. \tag{6}$$

as a constrained optimisation problem

$$(\hat{\beta}, \hat{\gamma}) \in \operatorname{argmax}_{(b,c) \in \mathbb{R}^p \times \mathbb{R}^{Q-1}} \; l_{Y|X}(b,c) \text{ with } \|b\|_1 \le r \tag{7}$$

for an appropriate value of r. In this new formulation, the problem of choosing λ is translated into the problem of choosing r.

Generally speaking, each iteration of the FW algorithm consists of finding

$$s^k = \operatorname{argmin}_{\mathbf{s} \in \mathcal{D}} \mathbf{s}^{\mathrm{T}} \nabla \mathrm{f}(\mathbf{x}_\mathrm{k}),$$

then upgrade $\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{2}{k+2}(\mathbf{s}_k - \mathbf{x}_k)$, where f is the function to minimize, k is the current iteration, and $\mathcal{D}$ is the set on which we want to optimize f. In the case where $\mathcal{D}$ is the hypercube defined by $\|\beta\|_1 \le r$ (as in our case), determining s_k is simple, since it is the point:

- of coordinate r for the component such that $\nabla_\beta \, l_{\tilde{Y}|X}(\beta, \gamma)$ is minimal,
- and zero for all the other components.

However, the logistic regression is a special case in which constraints have to be put on β, but not on γ. Practically speaking, the choice has been to alternate iterations of the Frank-Wolfe algorithm (to optimize β with γ fixed) with a simple gradient descent (to optimize γ with β fixed).

Algorithm 2 Find a λ that cancels all β components following a dichotomy approach.

V: number of non-zero coefficients of β, as a function of λ.
δ: desired accuracy, set by the user (default value: $\delta = 0.01$).
$\lambda_{max} = 1$
while $\mathrm{V}(\lambda_{max}) \neq 0$ **do**
  $\lambda_{max} = \lambda_{max} \times 2$
end while
$\lambda_{min} = \frac{\lambda_{max}}{2}$
if $\lambda_{max} = 1$ **then**
  $\lambda_{min=0}$
end if
while $\lambda_{max} - \lambda_{min} \geq \delta$ **do**
  $\lambda_{mean} = \frac{\lambda_{max} - \lambda_{min}}{2}$
  if $V(\lambda_{mean}) = 0$ **then**
    $\lambda_{max} = \lambda_{mean}$
  else
    $\lambda_{min} = \lambda_{mean}$
  end if
end while
Output $\lambda_{\#} = \lambda_{mean}$.

2.3 Hyperparameter calibration

2.3.1 Selection of the parameter by AIC

The first implemented method to select the λ parameter is to use the Akaike information criterion (AIC) [2]. This AIC is a compromise between the likelihood of the model and the number of non-zero parameters. More precisely, $AIC = -2l_{Y|X}(\beta, \gamma) + 2\|\beta\|_0$, and the goal is to find a set of parameters that minimizes this value. The method of choosing lambda processes in three steps. In the first one, the objective to is determine a penalty $\lambda_{\#}$ that is large enough to cancel all β components. This objective is realized by using Algorithm 2.

One can then apply, e.g. Nesterov's or the stochastic Frank-Wolfe algorithm with different values of the hyperparameter. One possible set of values is $\lambda_0 = 0$, $\lambda_1 = \frac{\lambda_{\#}}{50}$, $\lambda_2 = \frac{2\times\lambda_{\#}}{50}$, ..., $\lambda_{49} = \frac{49\times\lambda_{\#}}{50}$. The AIC value is then computed for each obtained model and, at the end of the day, the model with smallest AIC is finally selected.

2.3.2 BIC Selection

λ is chosen in the same manner than for the AIC method, except for the fact that the value to optimize is, this time, $BIC = -2\ l_{Y|X}(\beta, \gamma) + \log(n)\|\beta\|_0$.

2.3.3 Adapting the Quantile Universal Threshold selection to ordinal polytomous regression

Quantile Universal Threshold (QUT) [17] is a simulation-based method. Its objective is to be sure that, if the vector Y to be predicted has no link with the matrix of predictive variables X, then the vector β of the regression coefficients will be the null vector with probability $1 - \alpha$, where α is set by the user (α is set to 5% in Section 3).

The working principle of QUT is as follows.

Algorithm 3 QUT : successive evaluations of gamma knowing lambda, and of lambda knowing gamma.

$\lambda = \sqrt{2} \times \log(2 \times \max(\mathrm{p}, 1)) \times \max(0.01, \mathrm{std}(\mathrm{Y}))$ (initialization of λ)
for i = 1 ... 3 **do**
  Choose γ based on the current λ with Nesterov.
  Choose λ based on the current γ with QUT.
end for
Output λ.

- Randomly pick a large number of vectors of the same size than Y. For instance, in the case study of Section 3, 100 vectors $\tilde{Y}_1 ... \tilde{Y}_{100}$ are picked as permutations of the original Y vector. That is to say, $\tilde{Y}_i$ has the same number of subjects in each category as the initial vector Y.
- For each random vector $\tilde{Y}_i$, find a λ_i large enough such that, when the ℓ_1-penalised maximum likelihood estimator described in Section 2.1.2 is optimized, β is the null vector.
- The obtained λ_i are sorted, and then we select the value such that a proportion $1 - \alpha$ of the λ_i is below this threshold.

To speed up the second step of this process, the following property is used: if $\lambda_\# = \|\nabla_\beta \, l_{\tilde{Y}|X}(\beta = \vec{0}, \gamma)\|_\infty$, then the optimisation of the ℓ_1-penalised maximum likelihood estimator with a penalty of $\lambda_\#$ returns $\beta = \vec{0}$. Please note that $\lambda_\#$ is not necessarily the smallest possible penalisation such as $\beta = \vec{0}$.
γ is required in order to compute $\lambda_\#$. However γ is not known, and λ is needed to calculate it. So, a loop has been implemented as in Algorithm 3.

Thanks to the shortcut $\lambda_\# = \|\nabla_\beta \, l_{\tilde{Y}|X}(\beta = \vec{0}, \gamma)\|_\infty$, the computation time to obtain λ is greatly reduced, leading to the fastest determination of λ (see Section 3), as it requires only a few the optimisation of the ℓ_1-penalised maximum likelihood estimator. Note that a version of the QUT whose second step is performed by dichotomy, as in Algorithm 2, has been implemented too, but it underperforms the other methods in terms of computation time.

2.3.4 Selection of the r parameter by Online Frank-Wolfe algorithm

The method follows the procedure described in [11] with small necessary adjustments in order to accommodate for the specific constraints associated with our estimator. We refer the reader to the associated longer report [9] for complete details.

3 Simulation results

3.1 Description of the experiments

We now assess the practical performance of the proposed methods. For this purpose, we performed various numerical experiments on simulated data. The simulation and testing procedure works as follows.

1. The number of subjects n, the number of variables p, the number of influential variables s, and the underlying threshold vector γ_0 are set (Section 3.2 contains the authors' choices).

2. The vector of underlying parameters β_0 is randomly picked. This vector is of size p such that $p - s$ of its components are null, while the other s components follow a Gaussian law $\mathcal{N}(0, 1)$
3. The matrix X of explanatory variables is then drawn. This is a matrix of size $n \times p$, in which each component follows a law $\mathcal{N}(0, 1)$.
4. The noise vector ϵ of Y^* is drawn. It is of size n, where each component follows a logistic(1,1) law.
5. $Y^* = X\beta_0 + \epsilon$ is computed, and then Y based on Y^* and γ_0.
6. Steps 2, 3, 4, and 5 above allows the construction of a database. They are repeated 50 times, leading to 50 different databases.
7. Each of these 50 databases is divided into a learning sample ($\frac{2}{3}$ of the subjects) and a testing one (the other third).
8. Each of the regression methods listed in Section 3.2 is finally applied to the 50 learning samples. The performances of the models are measured on the 50 corresponding test samples based on the criteria defined in Section 3.2.

3.2 Results

The methods we decided to compare are the following.

- λ parameter selection by AIC as in Section 2.3.1.
- λ parameter selection by BIC as in Section 2.3.2.
- λ parameter selection according to Quantile Universal Threshold, as presented in Section 2.3.3.
- The use of the Frank-Wolfe algorithm, to solve the constrained optimization with selection of the r parameter using Online Frank-Wolfe, as defined in Sections 2.2.2 and 2.3.4. This model is named "OFW" in Tables 1 and 2.
- The absence of variable selection. That is to say, the model obtained when the likelihood is maximized without penalty. This model is simply named "$\lambda = 0$" in Tables 1 and 2.
- The model that predicts, for each subject in the test sample, the largest category of the learning sample. It is named "null model" in Tables 1 and 2, as this is the best possibility if no explanatory variable is taken into account.

$\lambda = 0$ and the null model are only performed to check if the first four methods work well. Indeed, when dealing with the logistic regression, it is important to check if the predictive model is better than simply placing all patients in the majority category. Moreover, when working on variables selection, it can be useful to check if the obtained model is better than the one with no selection.

Two experiments have been performed. In the first one, $n > p$, there are 50 variables, the learning sample has been constituted by 200 subjects, while the test sample has 100 subjects (see Table 1). In the other experiment, $p > n$, the learning sample has 100 subjects, the test one has 50 subjects, and there are 200 variables (Table 2). In both cases, the number of significant variables was set to $s = 5$.

We considered $Q = 3$ categories for Y, and we set $\gamma_0 \in [0, 3]$, as unbalanced categories were wanted to complicate the regression problem. With this choice of γ_0, Y_i is in the first category for all $Y_i^* \leq 0$, *i.e.*, for half of the simulated subjects. The Nesterov algorithm runs for 200 iterations, while the Franck-Wolfe one iterates 200 times.

For each method, four performance criteria are studied.

Table 1 Monte Carlo simulations with $n_{\text{learning}} = 200, p = 50, n_{\text{test}} = 100$.

choice of λ (or r)	correctly ranked	average likelihood	prediction error	CRW	Time	λ (or r)	Nb of variables
BIC	66.7	0.48	0.35	59.4	2464.9	14.8	3.9
QUT $\|\|\|_\infty$	65.4	0.48	0.36	57.9	53.0	22.0	2.6
AIC	65.3	0.47	0.36	58.6	2458.9	10.2	7.1
OFW	63.3	0.46	0.39	55.2	199.0	7.2	39.6
$\lambda = 0$	60.0	0.44	0.43	55.3	20.1	0.0	50.0
null model	49.0	-	-	33.3	0	-	0

- The percentage of subjects in the test sample which are correctly ranked by the model fitted on the learning sample. This percentage is named "correctly ranked" in Tables 1 and 2.
- The average likelihood. That is, the geometric mean of the probabilities that the model fitted to the learning sample assigns the actual categories of subjects in the test sample. This is what we called "average likelihood" in Tables 1 and 2.
- The average prediction error. That is to say, the average gap between the predicted category and the actual category, named "prediction error" in Tables 1 and 2.
- The percentage of correctly ranked subjects, weighted by the size of the categories. More precisely, we calculate $100 \times \sum_{i=1}^{n} 1_{\text{prediction is right}} \frac{Q \times p}{\#I_{Y_i}}$, where $\#I_{Y_i}$ is the number of subjects in the same category than Y_i. This criterion attaches greater importance to the proper classification of subjects that are in a poorly represented category. It is referenced as "CRW" in Tables 1 and 2.

The "average likelihood" and "correctly-ranked weighted" criteria are relevant when classes are very unbalanced (like 98 %, 1 %, and 1 %), which can really occur in practice. In the case study, the "correctly ranked" criterion has been considered first, as this is probably the most natural criterion for not too unbalanced categories like the ones used during our simulations. Tables 1 and 2 are sorted according to this criterion.

Table 1 summarizes the results in the case where the number of subjects in the training sample is 200, the number of subjects in the testing one is 100, and the number of explanatory variables is 50. Table 2, summarizes the results in the case where the number of subjects in the training sample is 100, the number of subjects in the testing one is 50, and the number of explanatory variables is 200.

To finish describing Tables 1 and 2, let us note that "nb of variables" represents the number of variables that the model considers as influential.

Wilcoxon tests have also been performed in order to determine if the differences between the methods are statistically significant. Tables 3 and 4 show the results of these Wilcoxon tests. In the $n_{\text{learning}} = 200, p = 50$ case, the difference between correctly ranked subjects for BIC (66,7%) and QUT (65,4%) is significant with a p-value of $8,03 \times 10^{-3}$, even if this difference is only equal to 1,3%. Conversely, in the $n_{\text{learning}} = 100, p = 200$ case, the difference between QUT, BIC, OFW is not significant. This case may require more simulated data if we want to separate these methods correctly. Finally, in any cases, QUT, BIC, OFW, and AIC are significantly better than λ =0 and the null model.

Table 2 Monte Carlo simulations with $n_{\text{learning}} = 100, p = 200, n_{\text{test}} = 50$.

choice of λ (or r)	correctly ranked	average likelihood	prediction error	CRW	Time	λ (or r)	Nb of variables
QUT $\|\|\|_\infty$	61.0	0.43	0.42	52.4	33.5	16.9	1.3
BIC	60.7	0.44	0.42	54.0	982.9	11.9	3.5
OFW	59.8	0.43	0.42	50.2	72.4	6.4	54.1
AIC	55.4	0.42	0.48	50.1	995.8	8.9	9.2
null model	48.1	-	-	33.3	0	-	0
$\lambda = 0$	36.7	0.22	0.77	40.0	12.8	0.0	200.0

Table 3 Paired Wilcoxon tests associated to Monte Carlo simulations with $n_{\text{learning}} = 200, p = 50, n_{\text{test}} = 100$.

	QUT$\|\|\|_\infty$	AIC	OFW	$\lambda = 0$	null model
BIC	8.03×10^{-3}	3.69×10^{-3}	7.83×10^{-7}	1.71×10^{-9}	7.38×10^{-10}
QUT$\|\|\|_\infty$	-	7.03×10^{-1}	3.36×10^{-3}	9.54×10^{-8}	7.83×10^{-10}
AIC	-	-	8.99×10^{-4}	2.02×10^{-8}	7.32×10^{-10}
OFW	-	-	-	1.58×10^{-6}	7.44×10^{-10}
$\lambda = 0$	-	-	-	-	1.95×10^{-9}

Table 4 Paired Wilcoxon tests associated to Monte Carlo simulations with $n_{\text{learning}} = 100, p = 200, n_{\text{test}} = 50$.

	BIC	OFW	AIC	null model	$\lambda = 0$
QUT	6.74×10^{-1}	3.77×10^{-1}	2.93×10^{-4}	$8,66 \times 10^{-9}$	9.13×10^{-10}
BIC	-	4.86×10^{-1}	5.47×10^{-4}	$1,62 \times 10^{-8}$	1.32×10^{-9}
OFW	-	-	2.66×10^{-5}	$1,17 \times 10^{-8}$	1.31×10^{-9}
AIC	-	-	-	$1,87 \times 10^{-5}$	3.33×10^{-9}
null model	-	-	-	-	$6,95 \times 10^{-7}$

4 Discussion

First of all, the four variable selection methods work better than $\lambda = 0$ and the null model. This shows that the algorithms work correctly, and that variable selection is useful. The absence of variable selection is particularly harmful in the case where $p > n$, see Table 2. It makes sense because, in this case, the optimisation of the unpenalised likelihood allows an infinite number of solutions. This $p > n$ case is very common in practice.

In the experiment shown in Table 1, the BIC works a bit better than the other methods, while in the experiment summarized in Table 2, QUT, BIC, and OFW are very close. In terms of computation time, QUT is the most interesting approach. Indeed, as explained in Section 2.3.3, this method allows to choose λ by executing the regression only a few times.

5 Conclusion

The present paper proposed a new estimator for sparse ordinal polytomous regression in a high dimensional setting together with a strategy for hyper-parameter calibration based on previous results from [17]. Performance of the method was assessed via extensive numerical experiments. The forthcoming report [9] will include further implementation details, and improvements, and additional numerical results on large real datasets.

References

1 Our python module. Accessed: 2018-05-11. URL: `https://github.com/SergeMOULIN/l1-penalised-ordinal-polytomous-regression-estimators`.

2 Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.

3 Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

4 Stephen Becker, Jérôme Bobin, and Emmanuel J Candès. Nesta: A fast and accurate first-order method for sparse recovery. *SIAM Journal on Imaging Sciences*, 4(1):1–39, 2011.

5 Alexandre Belloni, Victor Chernozhukov, and Lie Wang. Square-root lasso: pivotal recovery of sparse signals via conic programming. *Biometrika*, 98(4):791–806, 2011.

6 Peter J Bickel, Ya'acov Ritov, Alexandre B Tsybakov, et al. Simultaneous analysis of lasso and dantzig selector. *The Annals of Statistics*, 37(4):1705–1732, 2009.

7 Emmanuel Candes, Terence Tao, et al. The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, 2007.

8 Emmanuel J Candès, Yaniv Plan, et al. Near-ideal model selection by $\ell 1$ minimization. *The Annals of Statistics*, 37(5A):2145–2177, 2009.

9 Stephane Chretien, Guyeux Christophe, and Serge Moulin. l1-penalised ordinal polytomous regression estimators. *arXiv preprint to be submitted*, 2018.

10 Stéphane Chrétien and Sébastien Darses. Sparse recovery with unknown variance: a lasso-type approach. *IEEE Transactions on Information Theory*, 60(7):3970–3988, 2014.

11 Stephane Chretien, Alex Gibberd, and Sandipan Roy. Hedging hyperparameter selection for basis pursuit. *arXiv preprint arXiv:1805.01870*, 2018.

12 Stephane Chretien, Christophe Guyeux, Michael Boyer-Guittaut, Regis Delage-Mouroux, and Francoise Descotes. Investigating gene expression array with outliers and missing data in bladder cancer. In *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pages 994–998. IEEE, 2015.

13 Stéphane Chrétien, Christophe Guyeux, Michael Boyer-Guittaut, Régis Delage-Mouroux, and Françoise Descôtes. Using the lasso for gene selection in bladder cancer data. *arXiv preprint arXiv:1504.05004*, 2015.

14 Stéphane Chrétien, Christophe Guyeux, Bastien Conesa, Régis Delage-Mouroux, Michèle Jouvenot, Philippe Huetz, and Françoise Descôtes. A bregman-proximal point algorithm for robust non-negative matrix factorization with possible missing values and outliers-application to gene expression analysis. *BMC bioinformatics*, 17(8):284, 2016.

15 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics (NRL)*, 3(1-2):95–110, 1956.

16 Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

17 Caroline Giacobino, Sylvain Sardy, Jairo Diaz-Rodriguez, and Nick Hengartner. Quantile universal threshold for model selection. *arXiv preprint arXiv:1511.05433*, 2015.

18 Christopher Kennedy and Rachel Ward. Greedy variance estimation for the lasso. *arXiv preprint arXiv:1803.10878*, 2018.

19 Alan Miller. *Subset selection in regression*. CRC Press, 2002.

20 Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152, 2005.

21 Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). *Soviet Mathematics Doklady*, pages 372–376, 1983.

22 Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

23 Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

24 Robert Tibshirani. The lasso method for variable selection in the cox model. *Statistics in medicine*, 16(4):385–395, 1997.

25 Sara A Van de Geer et al. High-dimensional generalized linear models and the lasso. *The Annals of Statistics*, 36(2):614–645, 2008.

Differentially Mutated Subnetworks Discovery

Morteza Chalabi Hajkarim
Biotech Research and Innovation Centre, University of Copenhagen, Denmark
morteza.hajkarim@bric.ku.dk

Eli Upfal
Department of Computer Science, Brown University, Providence, RI, USA
eli@cs.brown.edu

Fabio Vandin[1]
Department of Information Engineering, University of Padova, Italy
fabio.vandin@unipd.it

Abstract

We study the problem of identifying differentially mutated subnetworks of a large gene-gene interaction network, that is, subnetworks that display a significant difference in mutation frequency in two sets of cancer samples. We formally define the associated computational problem and show that the problem is NP-hard. We propose a novel and efficient algorithm, called DAMOKLE to identify differentially mutated subnetworks given genome-wide mutation data for two sets of cancer samples. We prove that DAMOKLE identifies subnetworks with a statistically significant difference in mutation frequency when the data comes from a reasonable generative model, provided enough samples are available. We test DAMOKLE on simulated and real data, showing that DAMOKLE does indeed find subnetworks with significant differences in mutation frequency and that it provides novel insights not obtained by standard methods.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Applied computing → Biological networks, Applied computing → Computational genomics

Keywords and phrases Cancer genomics, network analysis, combinatorial algorithm

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.18

Related Version A full version of the paper is available at `http://www.dei.unipd.it/~vandinfa/DAMOKLE_long.pdf`.

Funding This work is supported, in part, by University of Padova project SID2017 and *STARS: Algorithms for Inferential Data Mining*, and by NSF grant IIS-1247581.

Acknowledgements The results presented in this manuscript are in whole or part based upon data generated by the TCGA Research Network: `http://cancergenome.nih.gov/`.

1 Introduction

The analysis of molecular measurements from large collections of cancer samples has revolutionized our understanding of the processes leading to a tumour through somatic mutations, changes of the DNA appearing during the lifetime of an individual [10]. One of the most important aspects of cancer revealed by recent large cancer studies is *inter-tumour genetic*

[1] corresponding author

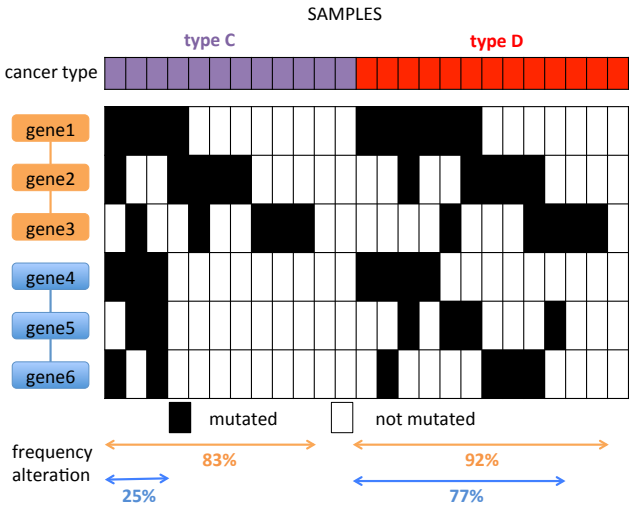


Figure 1 Identification of subnetworks with significant difference in mutation frequency in two set of samples $\mathcal{C}, \mathcal{D}$. The blue subnetwork is significantly more mutated in $\mathcal{D}$ than in $\mathcal{C}$, but it is not be detected by methods that look for the most significantly mutated subnetworks in $\mathcal{C}$ or in $\mathcal{D}$ or in $\mathcal{C} \cup \mathcal{D}$, since the orange subnetwork is in each case mutated at much higher frequency.

heterogeneity: each tumour presents hundreds-thousands mutations and no two tumours harbour the same set of DNA mutations [23].

One of the fundamental problems in the analysis of somatic mutations is the identification of the handful of *driver mutations* (i.e., mutations related to the disease) of each tumour, detecting them among the thousands or tens of thousands that are present in each tumour genome [32]. Inter-tumour heterogeneity renders the identification of driver mutations, or of driver genes (genes containing driver mutations), extremely difficult, since only few genes are mutated in a relatively large fraction of samples while most genes are mutated in a low fraction of samples in a cancer cohort [28].

Recently, several analyses (e.g, [18, 12]) have shown that interaction networks provide useful information to discover driver genes by identifying groups of interacting genes, called *pathways*, in which each gene is mutated at relatively low frequency while the entire group has one or more mutations in a significantly large fraction of all samples. Several network-based methods have been developed to identify groups of interacting genes mutated in a significant fraction of tumours of a given type and have been shown to improve the detection of driver genes compared to methods that analyze genes in isolation [18, 26, 13, 7].

The availability of molecular measurements in a large number of samples for different cancer types have also allowed *comparative* analyses of mutations in cancer [11, 14, 18]. Such analyses usually analyze large cohorts of different cancer types as a whole employing methods to find genes or subnetworks mutated in a significant fraction of tumours in *one* cohort, and also analyze each cancer type individually, with the goal to identify:

i) pathways that are common to various cancer types;
ii) pathways that are specific to a given cancer type.

For example, [18] analyzed 12 cancer types and identified subnetworks (e.g., a TP53 subnetwork) mutated in most cancer types as well as subnetworks (e.g., a MHC subnetwork) enriched for mutations in one cancer type. In addition, comparative analyses may also be used for the identification of mutations of clinical relevance [35]. For example: comparing mutations in a patients that responded to a given therapy with mutations in patients (of

the same cancer type) that did not respond to the same therapy may identify genes and subnetworks associated with response to therapy; comparing mutations in patients whose tumours metastasized with mutations in patients whose tumours did not metastasize may identify mutations associated with the insurgence of metastases.

Pathways that are significantly mutated only in a specific cancer type may not be identified by analyzing one cancer type at the time or all samples together (Figure 1), but, interestingly, to the best of our knowledge no method has been designed to *directly* identify sets of interacting genes that are significantly more mutated in a set of samples compared to another. The task of finding such sets is more complex than the identification of subnetworks significantly mutated in a set of samples, since subnetworks that have a significant difference in mutations in two sets may display relatively modest frequency of mutation in both set of samples, whose difference can be assessed as significant only by the joint analysis of both sets of samples.

Related Work. Several methods have been designed to analyze different aspects of somatic mutations in a large cohort of cancer samples in the context of networks. Some methods analyze mutations in the context of known pathways to identify the ones significantly enriched in mutations (e.g., [30]). Other methods combine mutations and large interaction networks to identify cancer subnetworks [29, 18, 5]. Networks and somatic mutations have also been used to prioritarize mutated genes in cancer [26, 13, 16, 24, 4] and for patients stratification [12, 17]. Some of these methods have been used for the identification of common mutation patterns or subnetworks in several cancer types [18, 11], but to the best of our knowledge no method has been designed to identify mutated subnetworks with a significant difference in two cohorts of cancer samples.

Few methods studied the problem of identifying subnetworks with significant differences in two sets of cancer samples using data other than mutations. [8] studied the problem of identifying optimally discriminative subnetworks of a large interaction network using gene expression data. [20] developed a procedure to identify statistically significant changes in the topology of biological networks. Such methods cannot be readily applied to find subnetworks with significant difference in mutation frequency in two sets of samples. Other related work use gene expression to characterize different cancer types: [33] defined a pathway-based score that clusters samples by cancer type, while [15] defined pathway-based features used for classification in various settings.

Our Contribution. In this work we study the problem of finding subnetworks with frequency of mutation that is significantly different in two sets of samples. In particular, our contributions are fourfold. First, we propose a combinatorial formulation for the problem of finding subnetworks significantly more mutated in one set of samples than in another and prove that such problem is NP-hard. Second, we propose DifferentiAlly Mutated subnetwOrKs anaLysis in cancEr (DAMOKLE), a simple and efficient algorithm for the identification of subnetworks with a significant difference of mutation in two sets of samples, and analyze DAMOKLE proving that it identifies subnetworks significantly more mutated in one of two sets of samples under reasonable assumptions for the data. Third, we test DAMOKLE on simulated data, verifying experimental that DAMOKLE correctly identifies subnetworks significantly more mutated in a set of samples when enough samples are provided in input. Fourth, we test DAMOKLE on large cancer datasets comprising two cancer types, and show that DAMOKLE identifies subnetworks significantly associated with one of the two types which cannot be identified by state-of-the-art methods designed for the analysis of one set of samples.

2 Methods and Algorithms

This section presents the problem we study, the algorithm we propose for its solution, and the analysis of our algorithm. In particular, Section 2.1 formalizes the computational problem we consider; Section 2.2 presents DifferentiAlly Mutated subnetwOrKs anaLysis in cancEr (DAMOKLE), our algorithm for the solution of the computational problem; Section 2.3 describes the analysis of DAMOKLE under a reasonable generative model for mutations; Section 2.4 presents a formal analysis of the statistical significance of subnetworks obtained by DAMOKLE; and Section 2.5 describes two permutation test to assess the significance of the results of DAMOKLE for limited sample sizes.

2.1 Computational Problem

We are given measurements on mutations in m genes $\mathcal{G} = \{1, \ldots, m\}$ on two sets $\mathcal{C} = \{c_1, \ldots, c_{n_C}\}, \mathcal{D} = \{d_1, \ldots, d_{n_D}\}$ of samples. Such measurements are represented by two matrices C and D, of dimension $m \times n_C$ and $m \times n_D$, respectively, where n_C (resp., n_D) is the number of samples in $\mathcal{C}$ (resp., $\mathcal{D}$). $C(i,j) = 1$ (resp., $D(i,j) = 1$) if gene i is mutated in the j-th sample of $\mathcal{C}$ (resp., $\mathcal{D}$) and $C(i,j) = 0$ (resp., $D(i,j) = 0$) otherwise. We are also given an (undirected) graph $G = (V, E)$, where vertices $V = \{1, \ldots, m\}$ are genes and $(i,j) \in E$ if gene i interacts with gene j (e.g., the corresponding proteins interact).

Given a set of genes $S \subset \mathcal{G}$, we define the indicator function $c_S(c_i)$ with $c_S(c_i) = 1$ if at least one of the genes of S is mutated in sample c_i, and $c_S(c_i) = 0$ otherwise. We define $c_S(d_i)$ analogously. We define the *coverage* $c_S(\mathcal{C})$ *of* S *in* $\mathcal{C}$ as the fraction of samples in $\mathcal{C}$ for which at least one of the genes in S is mutated in the sample, that is $c_S(\mathcal{C}) = \frac{\sum_{i=1}^{n_C} c_S(c_i)}{n_C}$ and, analogously, define the *coverage* $c_S(\mathcal{D})$ *of* S *in* $\mathcal{D}$ as $c_S(\mathcal{D}) = \frac{\sum_{i=1}^{n_D} c_S(d_i)}{n_D}$.

We are interested in identifying sets of genes S, with $|S| \leq k$, corresponding to connected subgraphs in G and displaying a *significant* difference in coverage between $\mathcal{C}$ and $\mathcal{D}$, i.e., with a high value of $|c_S(\mathcal{C}) - c_S(\mathcal{D})|$. We define the *differential coverage* $dc_S(\mathcal{C}, \mathcal{D})$ as $dc_S(\mathcal{C}, \mathcal{D}) = c_S(\mathcal{C}) - c_S(\mathcal{D})$.

In particular, we study the following computational problem.

The Differentially Mutated Subnetworks Discovery problem: *G*iven a value θ with $\theta \in [0, 1]$, find all connected subgraphs S of G of size $\leq k$ such that $dc_S(\mathcal{C}, \mathcal{D}) \geq \theta$.

Note that by finding sets that maximize $dc_S(\mathcal{C}, \mathcal{D})$ we identify sets with significantly more mutations in $\mathcal{C}$ than in $\mathcal{D}$, while to identify sets with significantly more mutations in $\mathcal{D}$ than in $\mathcal{C}$ we need to find sets maximizing $dc_S(\mathcal{D}, \mathcal{C})$. In addition, note that a subgraph S in the solution may contain genes that are not mutated in $\mathcal{C} \cup \mathcal{D}$ but that are needed for the connectivity of S.

We have the following. (Omitted proofs can be found in the long version of the paper available at: `http://www.dei.unipd.it/~vandinfa/DAMOKLE_long.pdf`.)

▶ **Theorem 1.** *The Differentially Mutated Subnetworks Discovery problem is NP-hard.*

Proof. The proof is by reduction from the connected maximum coverage problem [29]. In the connected maximum coverage problem we are given a graph G defined on a set $V = \{v_1, \ldots, v_n\}$ of n vertices, a family $\mathcal{P} = \{P_1, \ldots, P_n\}$ of subsets of a universe I (i.e., $P_i \in 2^I$), with P_i being the subset of I *covered* by $v_i \in V$ and value k, and we want to find the subgraph $C^* = \{v_{i_1}, \ldots, v_{i_k}\}$ with k nodes of G that maximizes $|\cup_{j=1}^{k} P_{i_j}|$.

Given an instance of the connected maximum coverage problem, we define an instance of the Differentially Mutated Subnetworks Discovery problem as follows: the set $\mathcal{G}$ of genes

Algorithm 1: DAMOKLE.

Input: mutation matrices C, D; gene-gene interaction graph $G = (V, E)$; integer $k > 0$; $\theta \in [0, 1]$
Output: maximal connected subgraphs with $dc_S(\mathcal{C}, \mathcal{D}) \geq \theta$

```
solutions ← ∅;
foreach {u, v} ∈ E do
    if dc_{u,v}(C, D) ≥ θ/(k − 1) then
        solutions ← solutions ∪ GetSolutions(E,{u, v});
    end
end
return solutions;
```

corresponds to the set V of vertices of G in the connected maximum coverage problem, and the graph G is the same as in the instance of the maximum coverage instance; the set $\mathcal{C}$ is given by the set I and the matrix C is defined as $C_{i,j} = 1$ if $i \in P_j$, while $\mathcal{D} = \emptyset$.

Note that for any subgraph S of G, the differential coverage $dc_D(\mathcal{C}, \mathcal{D}) = c_S(\mathcal{C}) - c_S(\mathcal{D}) = c_S(\mathcal{C})$ and $c_S(\mathcal{C}) = |\cup_{g \in S} P_g|/|I|$. Since $|I|$ is the same for all solutions, the optimal solution of the Differentially Mutated Subnetworks Discovery instance corresponds to the optimal solution to the connected maximum coverage instance, and viceversa. ◀

2.2 Algorithm

We now describe DifferentiAlly Mutated subnetwOrKs anaLysis in cancEr (DAMOKLE), an algorithm to solve the Differentially Mutated Subnetworks Discovery problem. DAMOKLE takes in input mutation matrices C and D for two sets $\mathcal{C}$, $\mathcal{D}$ of samples, a (gene-gene) interaction graph G, and integer k, and a real value $\theta \in [0, 1]$, and returns subnetworks S of G with $\leq k$ vertices and differential coverage $dc_S(\mathcal{C}, \mathcal{D}) \geq \theta$. Subnetworks reported by DAMOKLE are also *maximal* (no edge can be added to S while maintaining $|S| \leq k$ *and* $dc_S(\mathcal{C}, \mathcal{D}) \geq \theta$). DAMOKLE is described in Algorithm 1. DAMOKLE starts by considering each edge $e = \{u, v\} \in E$ of G with differential coverage $dc_{\{u,v\}}(\mathcal{C}, \mathcal{D}) \geq \theta/(k-1)$, and for each such e identifies subnetworks including e to be reported in output using Algorithm 2.

GetSolutions, described in Algorithm 2, is a recursive algorithm that, give a current subgraph S, identifies all maximal connected subgraphs $S', |S'| \leq k$, containing S and with $dc_{S'}(\mathcal{C}, \mathcal{D}) \geq \theta$. This is obtained by expanding S one edge at the time and stopping when the number of vertices in the current solution is k or when the addition of no vertex leads to an increase in differential coverage $dc_S(\mathcal{C}, \mathcal{D})$ for the current solution S. In Algorithm 2, $N(S)$ refers to the set of edges with exactly one vertex in the set S.

The motivation for design choices of DAMOKLE are provided in the next section.

2.3 Analysis of DAMOKLE

The design and analysis of DAMOKLE are based on the following generative model for the underlying biological process.

Model

For each gene $i \in \mathcal{G} = \{1, 2, ..., m\}$ there is an a-priori probability p_i of observing a mutation in gene i. Let $H \subset \mathcal{G}$ be the connected subnetwork of up to k genes that is differentially mutated in samples of $\mathcal{C}$ w.r.t. samples of $\mathcal{D}$. Mutations in our samples are taken from

Algorithm 2: GetSolutions.

Input: set E of edges of the graph; current subgraph (solution) S
Output: maximal connected subgraphs containing S with $dc_S(\mathcal{C},\mathcal{D}) \geq \theta$

```
nextEdges ← ∅;
foreach e ∈ N(S) do
    if dc_{S∪{e}}(C,D) ≥ dc_S(C,D) then nextEdges ← nextEdges ∪{e};
end
if |nextEdges| = 0 OR |S| = k then
    if dc_S(C,D) ≥ θ then return S;
end
newSols ← ∅;
foreach e ∈ nextEdges do newSols ← newSols ∪ GetSolutions(E,S ∪ {e}) ;
return newSols;
```

two related distributions. In the "control" distribution F a mutation in gene i is observed with probability p_i independent of other genes' mutations. The second distribution F_H is analogous to the distribution F but we condition on the event $E(H)$ ="at least one gene in H is mutated in the sample".

For genes not in H, all mutations come from distribution F. For genes in H, in a perfect experiment with no noise we would assume that samples in $\mathcal{C}$ are taken from F_H and samples from $\mathcal{D}$ are taken from F. However, to model realistic, noisy data we assume that with some probability q the "true" signal for a sample is lost, that is the sample from $\mathcal{C}$ is taken from F. In particular, samples in $\mathcal{C}$ are taken with probability $1-q$ from F_H and with probability q from F.

Let p be the probability that H has at least one mutation in samples from the control model F, $p = 1 - \prod_{j\in H}(1-p_j) \approx \sum_{j\in H} p_j$. Clearly, we are only interested in sets $H \subset \mathcal{G}$ with $p \ll 1$.

If we focus on individual genes, the probability gene i is mutated in a sample from $\mathcal{D}$ is p_i, while the probability that it is mutated in a sample from $\mathcal{C}$ is $\frac{(1-q)p_i}{1-\prod_{j\in H}(1-p_j)} + qp_i$. Such a gap may be hard to detect with a small number of samples. On the other hand, the probability of $E(H)$ (i.e., of at least one mutation in the set H) in a sample from $\mathcal{C}$ is $(1-q) + q(1-\prod_{j\in H}(1-p_j)) = 1-q+qp$, while the probability of $E(H)$ in a sample from $\mathcal{D}$ is $1-\prod_{j\in H}(1-p_j) = p$ which is a more significant gap, when $p \ll 1$.

The efficiency of DAMOKLE is based on two fundamental results. First we show that it is sufficient to start the search only in edges with relatively high discrepancy.

▶ **Proposition 1.** *If $dc_S(\mathcal{C},\mathcal{D}) \geq \theta$, then, in the above generating model, with high probability (asymptotic in n_C and n_D) there exist an edge $e \in S$ such that $dc_{\{e\}}(\mathcal{C},\mathcal{D}) \geq (\theta-\epsilon)/(k-1)$, for any $\epsilon > 0$.*

The second result motivates the choice, in Algorithm 2, of adding only edges that increase the score of the current solution (and to stop if there is no such edge).

▶ **Proposition 2.** *If subgraph S can be partitioned as $S = S' \cup \{j\} \cup S''$, and $dc_{S'\cup\{j\}}(\mathcal{C},\mathcal{D}) < dc_{S'}(\mathcal{C},\mathcal{D}) - pp_j$, then with high probability (asymptotic in $n_\mathcal{D}$) $dc_{S\setminus\{j\}}(\mathcal{C},\mathcal{D}) > dc_S(\mathcal{C},\mathcal{D})$.*

2.4 Statistical Significance of the Results

To compute a threshold that guarantees statistical confidence of our finding, we first compute a bound on the gap in a non significant set.

▶ **Theorem 2.** *Assume that S is not a significant set, i.e., $\mathcal{C}$ and $\mathcal{D}$ have the same distribution on S, then*

$$Prob(dc_S(\mathcal{C},\mathcal{D}) > \epsilon) \leq 2e^{-2\epsilon^2 n_\mathcal{C} n_\mathcal{D}/(n_\mathcal{C}+n_\mathcal{D})}.$$

Let N_k be the set of subnetworks under consideration, or the set of all connected components of size $\leq k$. We use Theorem 2 to obtain guarantees on the statistical significance of the results of DAMOKLE in terms of the Family-Wise Error Rate (FWER) or of the False Discovery Rate (FDR) as follows:

- FWER: if we want to find just the subnetwork with significant maximum differential coverage, to bound the FWER of our method by α we use the maximum ϵ such that $N_k 2e^{-2\epsilon^2 n_\mathcal{C} n_\mathcal{D}/(n_\mathcal{C}+n_\mathcal{D})} \leq \alpha$.
- FDR: if we want to find several significant subnetworks with high differential coverage, to bound the FDR by α we use the maximum ϵ such that $N_k 2e^{-2\epsilon^2 n_\mathcal{C} n_\mathcal{D}/(n_\mathcal{C}+n_\mathcal{D})}/n(\alpha) \leq \alpha$, where $n(\alpha)$ is the number of sets with differential coverage $\geq \epsilon$.

2.5 Permutation Testing

While Theorem 2 shows how to obtain guarantees on the statistical significance of the results of DAMOKLE by appropriately setting θ, in practice, due to relatively small sample sizes and to inevitable looseness in the theoretical guarantees, a permutation testing approach may be more effective in estimating the statistical significance of the results of DAMOKLE and provide more power for the identification of differentially mutated subnetworks.

We consider two permutation tests to assess the association of mutations in the subnetwork with the highest differential coverage found by DAMOKLE. The first test assesses whether the observed differential coverage can be obtained under the independence of mutations in genes by considering the null distribution in which each gene is mutated in a random subset (of the same cardinality as observed in the data) of all samples, independently of all other events. The second test assesses whether, under the observed marginal distributions for mutations in sets of genes, the observed differential coverage of a subnetwork can be obtained under the independence between mutations and samples' memberships (i.e., being a sample of $\mathcal{C}$ or a sample of $\mathcal{D}$), by randomly permuting the samples memberships.

Let $dc_S(\mathcal{C},\mathcal{D})$ be the differential coverage observed on real data for the solution S with highest differential coverage found by DAMOKLE (for some input parameters). For both tests we estimate the p-value as follow:

1. generate N (permuted) datasets from the null distribution;
2. run DAMOKLE (with the same input parameters used on real data) on each of the N permuted datasets;
3. let x be the number of permuted datasets in which DAMOKLE reports a solution with differential coverage $\geq dc_S(\mathcal{C},\mathcal{D})$: then the p-value of S is $(x+1)/(N+1)$.

3 Results

We implemented DAMOKLE in Python and tested it on simulated and on cancer data. Our experiments have been conducted on a Linux machine with 16 cores and 256 GB of RAM. All experiments required less than 10 MB of RAM and at most one day (for the

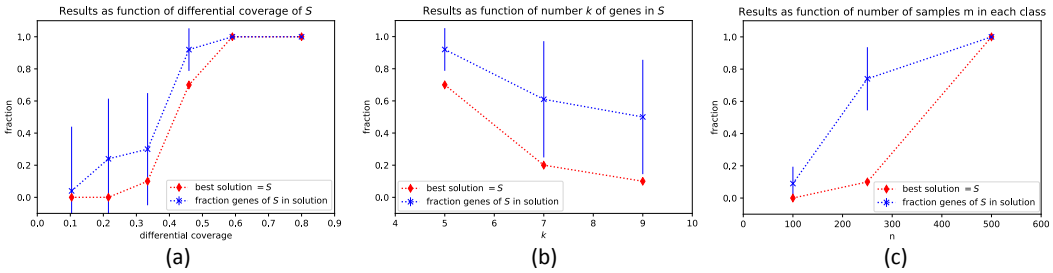


Figure 2 (a) Performance of DAMOKLE as a function of the differential coverage $dc_S(\mathcal{C}, \mathcal{D})$ of subnetwork S. The figure shows (red) the fraction of times, out of 10 experiments, that the best solution corresponds to S and (blue) the fraction of genes in S that are reported in the best solution by DAMOKLE. For the latter, error bars show the standard deviation on the 10 experiments. $n = 100$ and $k = 5$ for all experiments. (b) Performance of DAMOKLE as a function of the number k of genes in subnetwork S. $n = 100$ and $dc_S(\mathcal{C}, \mathcal{D}) = 0.46$ for all experiments. (c) Performance of DAMOKLE as a function of the number n of samples in $\mathcal{C}, \mathcal{D}$. $k = 10$ and $dc_S(\mathcal{C}, \mathcal{D}) = 0.46$ for all experiments.

largest simulated datasets). For all our experiments we used as interaction graph G the HINT+HI2012 network[2] [18], a combination of the HINT network [9] and the HI-2012 [34] set of interactions. In all cases we considered only the subnetwork with the highest differential coverage among the ones returned by DAMOKLE. We first present the results on simulated data (Section 3.1) and then present the results on cancer data (Section 3.2).

3.1 Simulated data

We tested DAMOKLE on simulated data generated as follows. We simulate data assuming there is a subnetwork S of k genes with differential coverage $dc_S(\mathcal{C}, \mathcal{D}) = c$. In our simulations we set $|\mathcal{C}| = |\mathcal{D}| = n$. For each sample in $\mathcal{D}$, each gene g in G (including S) is mutated with probability p_g, independently of all other events. For samples in $\mathcal{C}$, we first mutated each gene g with probability p_g independently of all other events. We then considered the samples of $\mathcal{C}$ without mutations in S, and for each such sample we mutated, with probability c, one gene of S, chosen uniformly at random. In this way c is the *expectation* of the differential coverage $dc_S(\mathcal{C}, \mathcal{D})$. For genes in $G \setminus S$ we used mutation probabilities p_g estimated from oesophageal cancer data [22]. We considered only value of $n \geq 100$, consistent with sample sizes in most recent cancer sequencing studies[3].

The goal of our investigation using simulated data is to evaluate the impact of various parameters on ability of DAMOKLE to recover S or part of it. To evaluate the impact of such parameters, for each combination of parameters in our experiments we generated 10 simulated datasets and run DAMOKLE on each dataset with $\theta = 0.01$, recording

1. the fraction of times that DAMOKLE reported S as the solution with the highest differential coverage, and
2. the fraction of genes of S that are in the solution with highest differential coverage found by DAMOKLE.

We first investigated the impact of the differential coverage $c = dc_S(\mathcal{C}, \mathcal{D})$. We analyzed simulated datasets with $n = 100$ samples in each class, where $k = 5$ genes are part of the subnetwork S, for values of $c = 0.1, 0.22, 0.33, 0.46, 0.6, 0.8,$. We run DAMOKLE on each

[2] `http://compbio-research.cs.brown.edu/pancancer/hotnet2/`
[3] `https://dcc.icgc.org/`

dataset with $k = 5$. The results are shown in Figure 2(a). For low values of the differential coverage c, with $n = 100$ samples DAMOKLE never reports S as the best solution found and only a small fraction of the genes in S are part of the solution reported by DAMOKLE. However, as soon as the differential coverage is ≥ 0.45, even with $n = 100$ samples in each class DAMOKLE identifies the entire planted solution S most of the times, and even when the best solution does not entirely corresponds to S, more than 80% of the genes of S are reported in the best solution. For values of $c \geq 0.6$, DAMOKLE *always* reports the whole subnetwork S as the best solution. Given that many recent large cancer sequencing studies consider at least 200 samples, DAMOKLE will be useful to identify differentially mutated subnetworks in such studies.

We then tested the performance of DAMOKLE as a function of the number of genes k in S. We tested the ability of DAMOKLE to identify a subnetwork S with differential coverage $dc_S(\mathcal{C}, \mathcal{D}) = 0.46$ in a dataset with $n = 100$ samples in both $\mathcal{C}$ and $\mathcal{D}$, when the number k of genes in S varies as $k = 5, 7, 9$. The results are shown in Figure 2(b). As expected, when the number of genes in S increases, the fraction of times S is the best solution as well as the fraction of genes reported in the best solution by S decreases, and for $k = 9$ the best solution found by DAMOKLE corresponds to S only 10% of the times. However, even for $k = 9$, on average most of the genes of S are reported in the best solution by DAMOKLE. Therefore DAMOKLE can be used to identify relatively large subnetworks mutated in a significantly different number of samples even when the number of samples is relatively low.

Finally, we tested the performance of DAMOKLE as the number of samples n in each set $\mathcal{C}, \mathcal{D}$ increases. In particular, we tested the ability of DAMOKLE to identify a relatively large subnetwork S of $k = 10$ genes with differential coverage $dc_S(\mathcal{C}, \mathcal{D}) = 0.46$ as the number of samples n increases. We analyzed simulated datasets for $n = 100, 250, 500$. The results are shown in Figure 2. For $n = 100$, when $k = 10$, DAMOKLE never reports S as the best solution and only a small fraction of all genes in S are reported in the solution. However, for $n = 250$, while DAMOKLE still reports S as the best solution only 10% of the times, on average 70% of the genes of S are reported in the best solution. More interestingly, already for $n = 500$, DAMOKLE *always* reports S as the best solution. These results show that DAMOKLE can reliably identify relatively large differentially mutated subnetworks from currently available datasets of large cancer sequencing studies.

3.2 Cancer data

We use DAMOKLE to analyze somatic mutations from The Cancer Genome Atlas. We first compared two similar cancer types and two very different cancer types to test whether DAMOKLE behaves as expected on these types. We then analyzed two pairs of cancer types where differences in alterations are unclear. In all cases we run DAMOKLE with $\theta = 0.1$ and obtained p-values with the permutation tests described in Section 2.5.

Lung Cancer. We used DAMOKLE to analyze 188 samples of lung squamous cell carcinoma (LUSC) and 183 samples of lung adenocarcinoma (LUAD). We only considered single nucleotide variants (SNVs)[4] and use $k = 5$. DAMOKLE did not report any significant subnetwork, in agreement with previous work showing that these two cancer types have known differences in gene expression [27] but are much more similar with respect to SNVs [3].

[4] `http://cbio.mskcc.org/cancergenomics/pancan_tcga/`

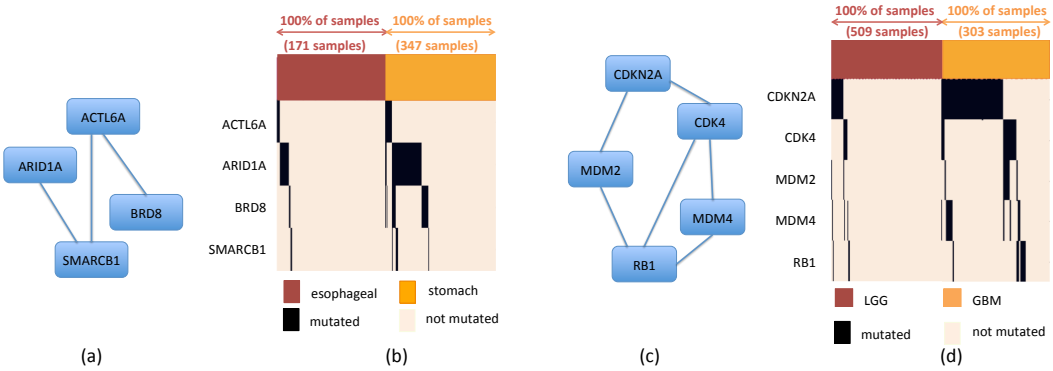


Figure 3 Results of DAMOKLE analysis of esophagus tumours and stomach tumours and of diffuse gliomas. (a) Subnetwork S with significant ($p < 0.02$) differential coverage in esophagus tumours vs stomach tumours (interactions from HINT+HI2012 network). (b) Fractions of samples with mutations in genes of S in esophagus tumours and in stomach tumours. c) Subnetwork S with significant ($p<0.01$) differential coverage in LGG samples vs GBM samples (interactions from HINT+HI2012 network). (d) Fractions of samples with mutations in genes of S in LGG samples and GBM samples.

Colorectal vs Ovarian Cancer. We used DAMOKLE to analyze 456 samples of colorectal adenocarcinoma (COADREAD) and 496 samples of ovarian serous cystadenocarcinoma (OV) using only SNVs[5]. For $k = 5$, DAMOKLE identifies the significant ($p < 0.01$ according to both tests in Section 2.5) subnetwork APC, CTNNB1, FBXO30, SMAD4, SYNE1 with differential coverage 0.81 in COADREAD w.r.t. OV. APC, CTNNB1, and SMAD4 are members of the WNT signaling and TFG-β signaling pathways, known to be involved in COADREAD [21]. The high differential coverage of the subnetwork is in accordance with COADREAD being altered mostly by SNVs and OV being altered mostly by copy number aberrations (CNAs) [6].

Esophagus-Stomach Cancer. We analyzed SNVs and CNAs in 171 samples of esophagus cancer and in 347 samples of stomach cancer [22].[6] The number of mutations in the two sets is not significantly different (t-test $p = 0.16$). We first considered single genes, identifying TP53 with high (> 0.5) differential coverage between the two cancer types. Alterations in TP53 have then be removed for the subsequent DAMOKLE analysis. We run DAMOKLE with $k = 4$ with $\mathcal{C}$ being the set of stomach tumours and $\mathcal{D}$ being the set of esophagus tumours. DAMOKLE identifies the significant ($p < 0.01$ for both tests in Section 2.5) subnetwork $S = \{$ACTL6A,ARID1A, BRD8, SMARCB1$\}$ with differential coverage 0.26 (Figure 3a-b). Such subnetwork is not reported as differentially mutated in the TCGA publication comparing the two cancer types [22]. BRD8 is only the top-16 gene by differential coverage, while ACTL6 and SMARCB1 are not among the top-2000 genes by differential coverage. ACTL6A, ARID1A, and SMARCB1 are all members of the chromatin organization machinery, recently associated with cancer [25, 19]. We compared the results obtained by DAMOKLE with the results obtained by HotNet2 [18], a method to identify significantly mutated subnetworks, using the same mutation data and the same interaction network as input: none of the genes in S appeared in significant subnetworks reported by HotNet2.

[5] `http://cbio.mskcc.org/cancergenomics/pancan_tcga/`
[6] `http://www.cbioportal.org/study?id=stes_tcga_pub#summary`

Diffuse Gliomas. We analyzed single nucleotide variants (SNVs) and copy number aberrations (CNAs) in 509 samples of lower grade glioma (LGG) and in 303 samples of glioblastoma multiforme (GBM).[7] We considered nonsilent SNVs, short indels, and CNAs. We removed from the analysis genes with < 6 mutations in both classes. By single gene analysis we identified IDH1 with high (> 0.5) differential coverage, and removed alterations in such gene for the DAMOKLE analysis. We run DAMOKLE with $k = 5$ with $\mathcal{C}$ being the set of GBM samples and $\mathcal{D}$ being the set of LGG samples. The number of mutations in $\mathcal{C}$ and in D is not significantly different (t-test $p = 0.1$). DAMOKLE identifies the significant ($p < 0.01$ for both tests in Section 2.5) subnetwork $S = \{$CDKN2A, CDK4, MDM2, MDM4, RB1$\}$ (Figure 3c-d). All genes in S are members of the p53 pathway or of the RB pathway, well known glioma cancer pathways [31].

Interestingly, [2] did not report any subnetwork with significant difference in mutations among LGG and GBM samples. CDK4, MDM2, MDM4, and RB1 do not appear among the top-45 genes by differential coverage. We compared the results obtained by DAMOKLE with the results obtained by HotNet2. Of the genes in our subnetwork, only CDK4 and CDKN2A are reported in a significantly mutated subnetwork ($p < 0.05$) obtained by HotNet2 analyzing $\mathcal{D}$ but not analyzing $\mathcal{C}$, while MDM2, MDM4, and RB1 are not reported in any significant subnetwork obtained by HotNet2.

4 Conclusion

In this work we study the problem of finding subnetworks of a large interaction network with significant difference in mutation frequency in two sets of cancer samples. This problem is extremely important to identify mutated mechanisms that are specific to a cancer (sub)type as well as for the identification of mechanisms related to clinical features (e.g., response to therapy). We provide a formal definition of the problem and show that the associated computational problem is NP-hard. We design, analyze, implement, and test a simple and efficient algorithm, DAMOKLE, which we prove identifies significant subnetworks when enough data from a reasonable generative model for cancer mutations is provided. Our results also show that the subnetworks identified by DAMOKLE cannot be identified by methods not designed for the *comparative* analysis of mutations in two sets of samples. We tested DAMOKLE on simulated and real data. The results on simulated data show that DAMOKLE identifies significant subnetworks with currently available sample size. The results on two large cancer datasets, each comprising genome-wide measurements of DNA mutations in two cancer subtypes, shows that DAMOKLE identifies subnetworks that are not found by methods not designed for the *comparative* analysis of mutations in two sets of samples.

While we provide a first method for the differential analysis of cohorts of cancer samples, several research directions remain. First, differences in the frequency of mutation of a subnetwork in two sets of cancer cohorts may be due to external (or hidden) variables, as for example the mutation rate of each cohort. While at the moment we ensure before running the analysis that no significant difference in mutation rate is present between the two sets, performing the analysis while correcting for possible differences in such confounding variable or in others would greatly expand the applicability of our method. Second, different types of mutation patterns (e.g., mutual exclusivity) among two set of samples could be explored

[7] `https://media.githubusercontent.com/media/cBioPortal/datahub/master/public/lgggbm_tcga_pub.tar.gz`

(e.g., extending the method proposed in [1]). Third, the inclusion of additional types of measurements, as for example gene expression, may improve the power of our method. Fourth, the inclusion of noncoding variants in the analysis may provide additional information to be leveraged to assess the significance of subnetworks.

References

1 Rebecca Sarto Basso, Dorit S Hochbaum, and Fabio Vandin. Efficient algorithms to discover alterations with complementary functional association in cancer. *arXiv preprint arXiv:1803.09721*, 2018.

2 Michele Ceccarelli, Floris P Barthel, Tathiane M Malta, Thais S Sabedot, Sofie R Salama, Bradley A Murray, Olena Morozova, Yulia Newton, Amie Radenbaugh, Stefano M Pagnotta, et al. Molecular profiling reveals biologically discrete subsets and pathways of progression in diffuse glioma. *Cell*, 164(3):550–563, 2016.

3 Fengju Chen, Yiqun Zhang, Edwin Parra, Jaime Rodriguez, Carmen Behrens, Rehan Akbani, Yiling Lu, JM Kurie, Don L Gibbons, Gordon B Mills, et al. Multiplatform-based molecular subtypes of non-small-cell lung cancer. *Oncogene*, 36(10):1384, 2017.

4 Ara Cho, Jung Eun Shim, Eiru Kim, Fran Supek, Ben Lehner, and Insuk Lee. Muffinn: cancer gene discovery via network analysis of somatic mutation data. *Genome biology*, 17(1):129, 2016.

5 Giovanni Ciriello, Ethan Cerami, Chris Sander, and Nikolaus Schultz. Mutual exclusivity analysis identifies oncogenic network modules. *Genome research*, 22(2):398–406, 2012.

6 Giovanni Ciriello, Martin L Miller, Bülent Arman Aksoy, Yasin Senbabaoglu, Nikolaus Schultz, and Chris Sander. Emerging landscape of oncogenic signatures across human cancers. *Nature genetics*, 45(10):1127, 2013.

7 Lenore Cowen, Trey Ideker, Benjamin J Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews Genetics*, 2017.

8 Phuong Dao, Kendric Wang, Colin Collins, Martin Ester, Anna Lapuk, and S Cenk Sahinalp. Optimally discriminative subnetwork markers predict response to chemotherapy. *Bioinformatics*, 27(13):i205–i213, 2011.

9 Jishnu Das and Haiyuan Yu. Hint: High-quality protein interactomes and their applications in understanding human disease. *BMC Syst Biol*, 6:92, 2012. doi:10.1186/1752-0509-6-92.

10 Levi A Garraway and Eric S Lander. Lessons from the cancer genome. *Cell*, 153(1):17–37, Mar 2013. doi:10.1016/j.cell.2013.03.002.

11 Katherine A Hoadley, Christina Yau, Denise M Wolf, Andrew D Cherniack, David Tamborero, Sam Ng, Max DM Leiserson, Beifang Niu, Michael D McLellan, Vladislav Uzunangelov, et al. Multiplatform analysis of 12 cancer types reveals molecular classification within and across tissues of origin. *Cell*, 158(4):929–944, 2014.

12 Matan Hofree, John P Shen, Hannah Carter, Andrew Gross, and Trey Ideker. Network-based stratification of tumor mutations. *Nat Methods*, 10(11):1108–15, Nov 2013. doi:10.1038/nmeth.2651.

13 Borislav H Hristov and Mona Singh. Network-based coverage of mutational profiles reveals cancer genes. *arXiv preprint arXiv:1704.08544*, 2017.

14 Cyriac Kandoth, Michael D McLellan, Fabio Vandin, Kai Ye, Beifang Niu, Charles Lu, Mingchao Xie, Qunyuan Zhang, Joshua F McMichael, Matthew A Wyczalkowski, Mark D M Leiserson, Christopher A Miller, John S Welch, Matthew J Walter, Michael C Wendl, Timothy J Ley, Richard K Wilson, Benjamin J Raphael, and Li Ding. Mutational landscape and significance across 12 major cancer types. *Nature*, 502(7471):333–9, Oct 2013. doi:10.1038/nature12634.

15 Shinuk Kim, Mark Kon, and Charles DeLisi. Pathway-based classification of cancer subtypes. *Biology direct*, 7(1):21, 2012.

16 Yoo-Ah Kim, Dong-Yeon Cho, Phuong Dao, and Teresa M Przytycka. Memcover: integrated analysis of mutual exclusivity and functional network reveals dysregulated pathways across multiple cancer types. *Bioinformatics*, 31(12):i284–i292, 2015.

17 Marine Le Morvan, Andrei Zinovyev, and Jean-Philippe Vert. Netnorm: capturing cancer-relevant information in somatic exome mutation data with gene networks for cancer stratification and prognosis. *PLoS computational biology*, 13(6):e1005573, 2017.

18 Mark D M Leiserson, Fabio Vandin, Hsin-Ta Wu, Jason R Dobson, Jonathan V Eldridge, Jacob L Thomas, Alexandra Papoutsaki, Younhun Kim, Beifang Niu, Michael McLellan, Michael S Lawrence, Abel Gonzalez-Perez, David Tamborero, Yuwei Cheng, Gregory A Ryslik, Nuria Lopez-Bigas, Gad Getz, Li Ding, and Benjamin J Raphael. Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes. *Nat Genet*, 47(2):106–14, Feb 2015. doi:10.1038/ng.3168.

19 Chao Lu and C David Allis. Swi/snf complex in cancer. *Nature genetics*, 49(2):178–179, 2017.

20 Raghvendra Mall, Luigi Cerulo, Halima Bensmail, Antonio Iavarone, and Michele Ceccarelli. Detection of statistically significant network changes in complex biological networks. *BMC systems biology*, 11(1):32, 2017.

21 Cancer Genome Atlas Network et al. Comprehensive molecular characterization of human colon and rectal cancer. *Nature*, 487(7407):330, 2012.

22 Cancer Genome Atlas Research Network et al. Integrated genomic characterization of oesophageal carcinoma. *Nature*, 541(7636):169–175, 2017.

23 Cancer Genome Atlas Research Network et al. Integrated genomic characterization of pancreatic ductal adenocarcinoma. *Cancer cell*, 32(2):185, 2017.

24 Sergio Pulido-Tamayo, Bram Weytjens, Dries De Maeyer, and Kathleen Marchal. Ssa-me detection of cancer driver genes using mutual exclusivity by small subnetwork analysis. *Scientific reports*, 6, 2016.

25 Srinivas Vinod Saladi, Kenneth Ross, Mihriban Karaayvaz, Purushothama R Tata, Hongmei Mou, Jayaraj Rajagopal, Sridhar Ramaswamy, and Leif W Ellisen. Actl6a is co-amplified with p63 in squamous cell carcinoma to drive yap activation, regenerative proliferation, and poor prognosis. *Cancer cell*, 31(1):35–49, 2017.

26 Raunak Shrestha, Ermin Hodzic, Thomas Sauerwald, Phuong Dao, Kendric Wang, Jake Yeung, Shawn Anderson, Fabio Vandin, Gholamreza Haffari, Colin C Collins, et al. Hit'ndrive: patient-specific multidriver gene prioritization for precision oncology. *Genome research*, 27(9):1573–1588, 2017.

27 Fenghao Sun, Xiaodong Yang, Yulin Jin, Li Chen, Lin Wang, Mengkun Shi, Cheng Zhan, Yu Shi, and Qun Wang. Bioinformatics analyses of the differences between lung adenocarcinoma and squamous cell carcinoma using the cancer genome atlas expression data. *Molecular medicine reports*, 16(1):609–616, 2017.

28 Fabio Vandin. Computational methods for characterizing cancer mutational heterogeneity. *Frontiers in genetics*, 8:83, 2017.

29 Fabio Vandin, Eli Upfal, and Benjamin J Raphael. Algorithms for detecting significantly mutated pathways in cancer. *Journal of Computational Biology*, 18(3):507–522, 2011.

30 Charles J Vaske, Stephen C Benz, J Zachary Sanborn, Dent Earl, Christopher Szeto, Jingchun Zhu, David Haussler, and Joshua M Stuart. Inference of patient-specific pathway activities from multi-dimensional cancer genomics data using paradigm. *Bioinformatics*, 26(12):i237–i245, 2010.

31 Bert Vogelstein and Kenneth W Kinzler. Cancer genes and the pathways they control. *Nature medicine*, 10(8):789–799, 2004.

32 Bert Vogelstein, Nickolas Papadopoulos, Victor E Velculescu, Shibin Zhou, Luis A Diaz, Jr, and Kenneth W Kinzler. Cancer genome landscapes. *Science*, 339(6127):1546–58, Mar 2013. doi:10.1126/science.1235122.

33 Michael R Young and David L Craft. Pathway-informed classification system (pics) for cancer analysis using gene expression data. *Cancer informatics*, 15:CIN–S40088, 2016.

34 Haiyuan Yu, Leah Tardivo, Stanley Tam, Evan Weiner, Fana Gebreab, Changyu Fan, Nenad Svrzikapa, Tomoko Hirozane-Kishikawa, Edward Rietman, Xinping Yang, Julie Sahalie, Kourosh Salehi-Ashtiani, Tong Hao, Michael E Cusick, David E Hill, Frederick P Roth, Pascal Braun, and Marc Vidal. Next-generation sequencing to generate interactome datasets. *Nat Methods*, 8(6):478–80, Jun 2011. doi:10.1038/nmeth.1597.

35 Ahmet Zehir, Ryma Benayed, Ronak H Shah, Aijazuddin Syed, Sumit Middha, Hyunjae R Kim, Preethi Srinivasan, Jianjiong Gao, Debyani Chakravarty, Sean M Devlin, et al. Mutational landscape of metastatic cancer revealed from prospective clinical sequencing of 10,000 patients. *Nature Medicine*, 2017.

D_{GEN}: A Test Statistic for Detection of General Introgression Scenarios

Ryan A. Leo Elworth
Department of Computer Science, Rice University, 6100 Main Street, Houston, TX, USA
r.a.leo.elworth@rice.edu

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

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

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

Luay Nakhleh
Department of Computer Science and Department of BioSciences, Rice University, 6100 Main Street, Houston, TX, USA
nakhleh@rice.edu

Abstract

When two species hybridize, one outcome is the integration of genetic material from one species into the genome of the other, a process known as introgression. Detecting introgression in genomic data is a very important question in evolutionary biology. However, given that hybridization occurs between closely related species, a complicating factor for introgression detection is the presence of incomplete lineage sorting, or ILS. The D-statistic, famously referred to as the "ABBA-BABA" test, was proposed for introgression detection in the presence of ILS in data sets that consist of four genomes. More recently, D_{FOIL} – a set of statistics – was introduced to extend the D-statistic to data sets of five genomes.

The major contribution of this paper is demonstrating that the invariants underlying both the D-statistic and D_{FOIL} can be derived automatically from the probability mass functions of gene tree topologies under the null species tree model and alternative phylogenetic network model. Computational requirements aside, this automatic derivation provides a way to generalize these statistics to data sets of any size and with any scenarios of introgression. We demonstrate the accuracy of the general statistic, which we call D_{GEN}, on simulated data sets with varying rates of introgression, and apply it to an empirical data set of mosquito genomes.

We have implemented D_{GEN} and made it available, both as a graphical user interface tool and as a command-line tool, as part of the freely available, open-source software package ALPHA (https://github.com/chilleo/ALPHA).

2012 ACM Subject Classification Applied computing → Genomics, Applied computing → Computational biology

Keywords and phrases Introgression, genealogies, phylogenetic networks

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.19

Funding This work was partially supported by NSF grants DBI-1355998, CCF-1302179, CCF-1514177, and DMS-1547433.

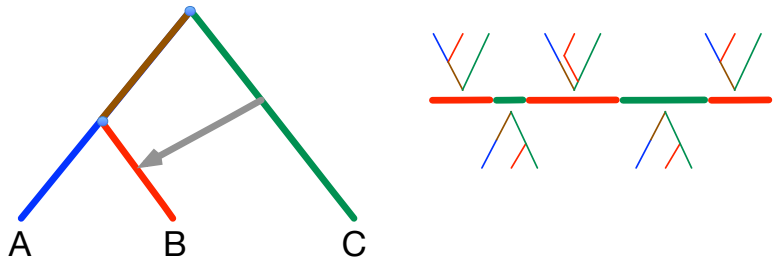


Figure 1 Hybridization and introgression. (Left) A phylogenetic network modeling the evolutionary history of three species (or, populations) A, B, and C. Species C split from the most recent common ancestor of A and B, and hybridization between (an ancestor of) C and (an ancestor of) B occurred. (Right) Due to hybridization and backcrossing, the genome of an individual in species B is a mosaic with different genomic segments having different genealogies. In particular, the genealogy of the middle segment involves incomplete lineage sorting.

1 Introduction

Hybridization – the interbreeding of individuals from two "different" species, or populations – has been recognized as an important evolutionary process underlying genomic diversification and species adaptation [1, 2, 22, 14, 15, 21, 8, 19, 16, 26]. Immediately upon interbreeding, each chromosome in the hybrid individual has a single source – the genome of one of the two parents. However, after multiple generations of backcrossing and recombination, the genomes of descendants of the hybrid individual turn into mosaics of genomic segments, each having a genealogy that could potentially differ from that of other segments (Fig. 1). The integration of genetic material from two different species into the genome of an individual is called *introgression*.

The discordance among the genealogies of different genomic segments could be used as a signal to detect introgression. For example, in the case of Fig. 1, the presence of some genealogies that place B closer to A than to C and others that place B closer to C than to A could indicate a potential hybridization event between B and C. However, a complicating factor in introgression detection is that incomplete lineage sorting, or ILS, could also be at play in cases where hybridization has occurred. ILS occurs when lineages from related populations fail to coalesce within the ancestral population, giving rise to the possibility that some lineages coalesce with others from farther populations. Mathematically, this process is often modeled by the multispecies coalescent [10, 24, 17, 5].

One class of methods for detecting hybridization and introgression, including in the presence of ILS, is to infer phylogenetic networks from the data of multiple unlinked loci sampled across the genomes. Indeed, several methods were introduced recently for this task [31, 29, 27, 23, 25, 32, 34, 33]. While providing accurate results, these methods are computationally very demanding.

A different approach is to use the so-called D-statistic [9, 6], which infers the presence of introgression based on significant deviation from equality between the frequencies of two site patterns in a 3-taxon (plus an outgroup) data set (details below). More recently, Pease and Hahn [18] introduced D_{FOIL}, which extends the D-statistic to detect introgression in a 5-taxon scenario (4 taxa plus an outgroup). The extension from three to four taxa involved a detailed analysis of site patterns and resulted in a set of statistics that, when combined, would aid in the detection of introgression. However, as stated, that work of Pease and Hahn extended the D-statistic from three to four taxa. Both the D-statistic and D_{FOIL} are examples of

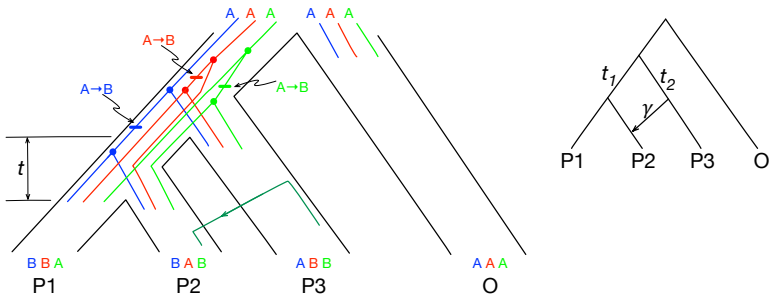


Figure 2 Illustration of the D-statistic. (Left) The demographic structure of three populations, P1, P2, and P3, along with an outgroup O, is shown. Patterns of the three parsimony-informative mutations (A→B) are shown for bi-allelic sites with states A and B, each mapped onto a different genealogy. The three genealogies give rise to patterns BBAA, BABA, and ABBA for the four taxa, respectively, when the taxa are listed in the order P1-P2-P3-O. The dark green arrow indicated gene flow from P3 to P2, which would result in excess of pattern ABBA. (Right) The phylogenetic network modeling the evolutionary history of the populations in the presence of gene flow from P3 to P2, where the gene flow is modeled as an instantaneous unidirectional event.

the use of phylogenetic invariants to detect deviation from the expected frequencies of site patterns under a neutral coalescent model with no gene flow. Similarly, the HyDe software package [3] implements an invariants-based method for identifying hybridization [13].

A major question is: Can one devise a statistic that is general enough to apply (the computational complexity issue aside) to data sets with any number of genomes and any set of postulated hybridization events?

In this paper, we address this question by showing that the phylogenetic invariants underlying both the D-statistic and D_{FOIL} could be generated automatically by contrasting gene tree distributions under the null multispecies coalescent [5] and the alternative multispecies network coalescent [30, 31]. Based on this observation, we devise an algorithm that automatically generates a statistic for detecting introgression in any evolutionary scenario. It is important to note, though, that as the number of genomes and number of postulated hybridization events increase, computing the statistic becomes computationally very demanding.

Our method, which we call D_{GEN}, is implemented in the publicly available, open-source software package ALPHA [7]. We demonstrate the accuracy of the method on simulated data sets, as well as its applicability to an empirical data set of mosquito genomes.

2 Methods

2.1 The D-statistic

Consider the species tree (((P1,P2),P3),O) in Fig. 2, which shows the evolutionary history of three species, or populations, P1, P2, and P3, along with an outgroup O. The significance of an outgroup in this scenario is that for any genomic site, the state that the outgroup has for that site is assumed to be the ancestral state of all three species P1, P2, and P3. We denote by A the ancestral state and by B the derived state.

Assuming all lineages from P1, P2, and P3 coalesce before any of them could coalesce with a lineage from O, there are three possible gene trees topologies, which are shown inside the branches of the species tree in Fig. 2, and are given by (((P1,P2),P3),O), (((P1,P3),P2),O), and (((P2,P3),P1),O). The probabilities of these three gene tree topologies when gene flow is

excluded (but incomplete lineage sorting is accounted for) are, respectively, $1-(2/3)e^{-t}$, $(1/3)e^{-t}$, and $(1/3)e^{-t}$, where t is the length, in coalescent units, of the branch that separates the splitting of P3 from the ancestor of P1 and P2 [4]. Clearly, the latter two gene tree topologies (those that are discordant with the species tree) have equal probabilities. Taking the two patterns BABA and ABBA to correspond to gene trees (((P1,P3),P2),O) and (((P2,P3),P1),O), then their expected frequencies in the absence of gene flow are equal.

However, when gene flow from P3 to P2 occurs and is modeled as an instantaneous event with probability γ (γ here is taken to represent the fraction of genomes in P2 that originated from P3 through gene flow), then the probabilities of the three gene tree topologies (((P1,P2),P3),O), (((P1,P3),P2),O), and (((P2,P3),P1),O) become, as derived in [28], $(1-\gamma)(1-(2/3)e^{-t_1})+(1/3)\gamma e^{-t_2}$, $(1/3)(1-\gamma)e^{-t_1}+\gamma(1-(2/3)e^{-t_2})$, and $(1/3)(1-\gamma)e^{-t_1}+(1/3)\gamma e^{-t_2}$, respectively, where t_1 and t_2 are the branch lengths, in coalescent units, in the phylogenetic network of Fig. 2. Now, with gene flow accounted for, when $\gamma \neq 0$ (and $t_2 > 0$), the expected frequencies of the two patterns BABA and ABBA are no longer equal. Thus, denoting by N_X the number of times site pattern X appears in a genomic data set, the D-statistic was defined as [6]

$$D = \frac{N_{ABBA} - N_{BABA}}{N_{ABBA} + N_{BABA}}, \tag{1}$$

and the significance of the deviation of D from 0 is assessed. Under no gene flow, we expect $D \approx 0$ (we do not write $D = 0$ since the counts in Eq. (1) are estimated from actual data and might not match the theoretical expectations exactly), and in the presence of gene flow, we expect D to deviate significantly from 0. Furthermore, when $D > 0$, it indicates introgression between P2 and P3 (in either or both directions), and when $D < 0$, it indicates introgression between P1 and P3.

To extend the D-statistic from the scenario depicted in Fig. 2 to the case of five taxa (four populations and an outgroup), Pease and Hahn [18] identified sets of site patterns that are expected to have equal frequencies under a no gene flow scenario but different frequencies when gene flow occurs. Next we show how to derive a general D-statistic that applies to a species phylogeny and any set of gene flow events, thus overcoming the need to derive a specialized D-statistic for individual evolutionary histories.

2.2 Towards the General Case

Let X be a set of taxa $X_1, X_2, \ldots, X_n$, where X_n is assumed to be an outgroup whose state A for a given bi-allelic marker is assumed to be the ancestral state. Then, for a given marker, a site pattern s is a sequence of length n where s_i $(1 \leq i < n)$, the state of the site in the genome of X_i, is either A or B.

Let $\mathcal{G}$ be the set of all rooted, binary gene trees on the n taxa $X_1, \ldots, X_n$. For a site pattern s, there might be multiple trees in $\mathcal{G}$ that are *compatible* with s; that is, trees on which the pattern s could have arisen in the presence of a single mutation (the infinite-sites assumption). We denote by $\mathcal{G}(s)$ the set of all trees in $\mathcal{G}$ that are compatible with pattern s. While the size of $\mathcal{G}$ only depends on the number of taxa n, the size of $\mathcal{G}(s)$ for a given s also depends on the number of ancestral versus derived alleles represented in s. For a given s with n total taxa and β taxa having the derived state (a 'B' instead of a 'A' in the site pattern), the size of $\mathcal{G}(s)$ will be the number of rooted, binary trees on β taxa times the number of rooted, binary trees on $n-\beta+1$ taxa. Given a species phylogeny Ψ, we have

$$P(s|\Psi) = \sum_{g \in \mathcal{G}(s)} P(g|\Psi) \tag{2}$$

where $P(g|\Psi)$ is the probability mass function (pmf) of [4] when Ψ is a species tree and the pmf of [28] when Ψ is a phylogenetic network.

Assuming independence among sites, the expected number of occurrences of site pattern s in a genomic data set given species phylogeny Ψ, denoted by $\mathbb{E}(n(s))$, is given by $n \cdot P(s|\Psi)$. Using this notation, a general statistic for detecting introgression proceeds as follows. Let Ψ be the species tree that corresponds to the evolutionary scenario of no gene flow. Let Ψ' be the phylogenetic network that is obtained by adding to Ψ the gene flow events (instantaneous events represented by horizontal edges) to be tested on Ψ. For example, the phylogenetic network in Fig. 2 is obtained by adding the gene flow event from P3 to P2. A general D-statistic, D_{GEN}, is then computed as follows:

1. Let S be the set of all distinct parsimony-informative site patterns.
2. Parameterize Ψ and Ψ' so that they define probability distributions on gene tree topologies.
3. For every site pattern $s \in S$, compute $P(s|\Psi)$ and $P(s|\Psi')$.
4. Let $\mathcal{P}^{tree}(S)$ be the partition of set S induced by the equivalence relations $\{(s_1, s_2) : P(s_1|\Psi) = P(s_2|\Psi)\}$.
5. Let $\mathcal{P}^{network}(S)$ be the partition of set S induced by the equivalence relations $\{(s_1, s_2) : P(s_1|\Psi') = P(s_2|\Psi')\}$.
6. Let $S' \subseteq \mathcal{P}^{tree}(S)$ where $Y \in S'$ if and only if $Y \not\subseteq Z$ for any $Z \in \mathcal{P}^{network}(S)$. In other words, Y is an element of S' if it consists of a set of site patterns that all have equal probabilities under Ψ but not equal probabilities under Ψ'.
7. Let $U = \{(T_Y, B_Y, M_Y) : Y \in S', T_Y = \text{argmax}_{\{Y\cap Z : Z\in\mathcal{P}^{network}(S), Y\cap Z\neq\emptyset\}} P(s|\Psi')$ and $B_Y = \text{argmin}_{\{Y\cap Z : Z\in\mathcal{P}^{network}(S), Y\cap Z\neq\emptyset\}} P(s|\Psi')\}$ where s is an arbitrary element of $Y \cap Z$, and $M_Y = Y - (T_Y \cup B_Y)$. Put simply, site pattern probabilities that were previously equal in the tree case become totally ordered in the network case and can be divided into sets based on their new relation to one another. In other words, as an equivalence class $Y \in S'$ is refined by the elements of $\mathcal{P}^{network}(S)$, T_Y and B_Y are the two subsets of site patterns in Y with the highest and lowest probabilities, respectively, and M_Y is the set of remaining site patterns.
8. $D_{\text{GEN}} = \left(\sum_{(T,B,M)\in U} N_T - N_B\right) / \left(\sum_{(T,B,M)\in U} N_T + N_B + 2N_M\right)$ where, as above, N_T is the number of times site patterns in T appear in the genomic data set (and similarly for N_B and N_M).
9. Similar to [18], calculate the χ^2 goodness of fit (df=1) using

$$\chi^2 = \left(\sum_{(T,B,M)\in U} N_T - N_B\right)^2 \Bigg/ \left(\sum_{(T,B,M)\in U} N_T + N_B + 2N_M\right).$$

Applying this algorithm to the case illustrated in Fig. 2, we have

- $\mathcal{P}^{tree}(S) = \{\{BBAA\}, \{BABA, ABBA\}\}$.
- $\mathcal{P}^{network}(S) = \{\{BBAA\}, \{BABA\}, \{ABBA\}\}$.
- Step (6) returns $S' = \{\{BABA, ABBA\}\}$.
- Step (7) returns $U = \{(\{BABA\}, \{ABBA\})\}$ which, indeed, is the D-statistic in the case of three taxa.

In Step (2) of the algorithm, we parameterize Ψ (and Ψ') by trying branch lengths (in coalescent units) in the set of values $\{0.5, 1.0, 2.0, 4.0\}$ and the set S' (in Step (6)) is determined based on the sets of site patterns whose equality does not break across the different settings of branch lengths. For the inheritance probability, we set it to 0.9.

In Step (7), if for an element (T, B, M) of U, we have $|T| \neq |B|$, we remove (arbitrarily) elements from the larger of the two sets to make them of equal size. Here, $|T|$ is distinguished

from N_T as $|T|$ represents how many site patterns are contained in set T (the cardinality of set T), whereas N_T represents the occurrence of the site patterns in T in an actual multiple sequence alignment.

For the χ^2 test, we used a threshold of 0.01 on the p-value to determine significance. That is, if the p-value is smaller than 0.01 we considered support for introgression to be statistically significant; otherwise, it is not. Given that our formulation bases its determination of whether introgression is present or not off of significant deviations of D_{GEN} away from zero, sign changes are treated equivalently and thus one could equivalently choose to take the absolute value of D_{GEN}. In our implementation of D_{GEN} we have chosen to leave the sign. More information on this is given in the discussion section.

Why not contrast the site pattern distribution to the known distribution of gene trees? One question that might arise is: Why do we not use a χ^2 test to compare the two distributions – the empirical one and the theoretical one; that is,

$$\chi^2 = \sum_s \frac{(N_s - n_s)^2}{n_s},$$

where the sum is taken over all distinct site patterns s, N_s is the observed count of site pattern s, and $n_s = n \cdot P(\mathcal{G}(s)|\Psi)$. The problem with this approach is that to compute n_s, we need knowledge of the parameters (branch lengths) of the species phylogeny, which are unknown in this case. One potential remedy to this limitation is to first estimate the species tree parameters from the data, say under maximum likelihood, and then use this parameterized model to compute the n_s frequencies. However, it is unknown how the estimated parameters compare to the (unknown) true values when gene flow had occurred but the assumed topology in the estimation is a tree. In our solution above, this problem is remedied by not focusing on the parameter values in an absolute sense, but rather use arbitrary settings to find the site patterns whose relative frequencies change between a model of no gene flow and another with gene flow.

3 Results

3.1 Simulations

We first studied the performance of our method on the five-taxon scenario studied in [18] and given by species tree Ψ_1 in Fig. 3. All simulations share the same values for several parameters. As in [18], we have a constant fixed population size of $N_e = 10^6$ and recombination rate of $r = 10^{-8}$. We also use a fixed mutation rate of $\mu = 7 \times 10^{-9}$. In our simulation pipeline, we first generate gene trees for a 50kbp multiple sequence alignment using `ms` [11], followed by simulating the sequences under the Jukes-Cantor model of evolution [12] using `seq-gen` [20]. In other words, the sequences are evolved under a finite-sites model. The parameter values were chosen primarily to accomplish the two goals of being similar to relevant past work as well as being biologically relevant. An example of the full commands for this pipeline, before adding any reticulations is as follows:

```
ms 5 1 -t 14000 -T -r 2000 50000 -I 5 1 1 1 1 1 -ej 1.0 2 1 -ej 1.0 4 3
-ej 1.2 3 1 -ej 1.5 5 1 | tail +4 | grep -v // > treefile
seq-gen -mHKY -l 50000 -s .028 -p 50000 < treefile > seqfile
```

We then added a migration event between P1 and P3 at time 0.5, with varying migration rates, and calculated our D_{GEN} statistic on the resulting genomic data sets; results are shown

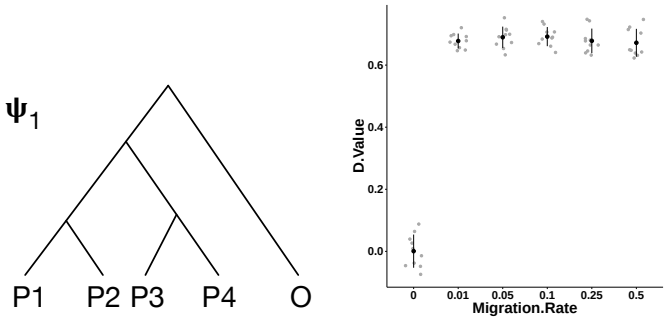


Figure 3 5-taxon simulation results. (Left) A 5-taxon species tree. The two most recent divergence events are set at 1.0 coalescent units, the divergence time of the ancestor of all in-group taxa (P1–P4) is set to 1.2 coalescent units, and the time of the root node is set to 1.5 coalescent units. A migration event between P1 and P3 at time 0.5 was added to species tree. (Right) Values of D_{GEN} on data sets with varying migration rates. Each point corresponds to a D_{GEN} value whose p-value was lower than 0.01 obtained from a different data set simulated under the same settings. The dark dots correspond to the mean and the lines correspond to 1 standard deviation around the mean.

in Fig. 3. As the results show, D_{GEN} performs very well at determining the presence of introgression in data sets. In particular, when the data evolved with no migration (migration rate 0), the D_{GEN} values hardly deviate from 0, and when the migration rate is non-zero, the method detects the presence of introgression with a strong deviation from 0. These results are consistent with the performance of D_{FOIL} [18].

Next, we considered cases beyond that of five taxa (i.e., cases not possible with either the D-statistic or D_{FOIL}). We conducted simulations that show the effect of migration rate and time of the migration event on the performance of D_{GEN}, as shown in Fig. 4.

As the results show, the D_{GEN} statistic performs very well at detecting introgression in this case as well. In particular, as the migration rate increases, so does the accuracy of the method. For a migration rate of 10^{-6} or higher, the method detects, with high significance, the presence of introgression. In the cases of extremely low migration rates (10^{-7} and 10^{-8}), the method tends to indicate slight deviation from a no-introgression scenario.

As for varying the time of the migration event, the accuracy of the method is what one would expect. As the time between the migration event and the divergence event increases (the time of the migration event decreases), the power to detect introgression is much higher. That power starts decreasing as the migration event becomes more ancient and, as a result, less signal is present for its detection.

3.1.1 Multiple Reticulations

The question we set out to investigate next is: Given that the D-statistic is designed to work under the assumption of a single gene flow event, how does it perform when there is more than one event? Fig. 5 shows a typical scenario for the D-Statistic, Scenario S1, in which the standard 4-taxon backbone tree has a single reticulation from P3 to P2.

The following four scenarios add an extra reticulation with a high migration rate. The effect of adding these reticulations on the value of the D-Statistic are shown in Fig. 6.

As expected, the S1 case yields the best results, followed by the S2 case with a weaker D value. All other statistic values demonstrate that even in the presence of a significant migration with introgression from P3 to P2, multiple introgressions can cause that information to be lost from inference. These results show that it is important to account for multiple

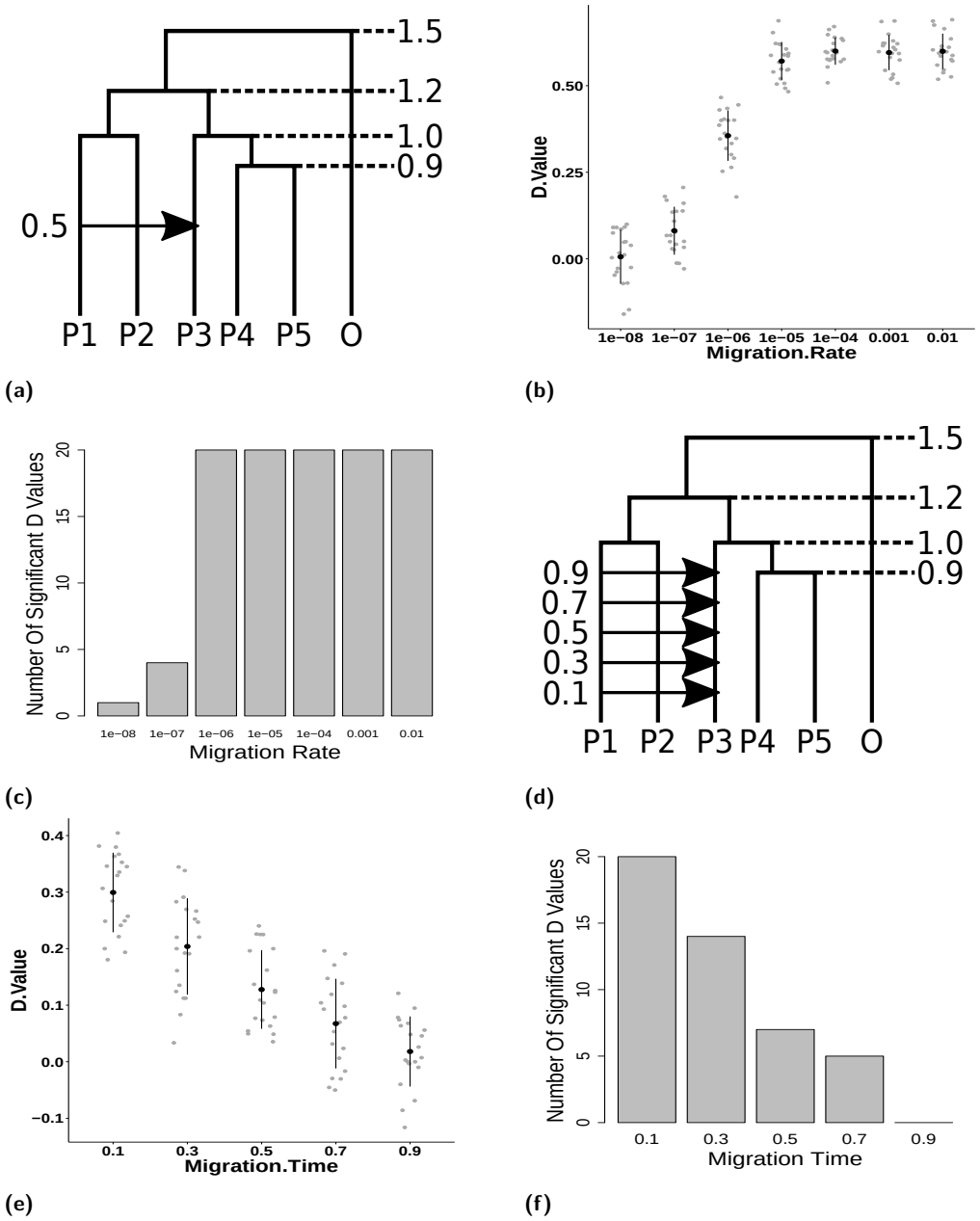


Figure 4 Simulation results on 6-taxon scenarios. (a) The network used for analyzing the effects of varying migration rate of a reticulation. The results of the corresponding DG values and the number of data sets where the D_{GEN} values were significant (p-value smaller than 0.01) are shown in (b) and (c), respectively. (d) The network used for analyzing the effects of varying the time of the migration event. The results of the corresponding DG values and the number of data sets where the D_{GEN} values were significant (p-value smaller than 0.01) are shown in (e) and (f), respectively.

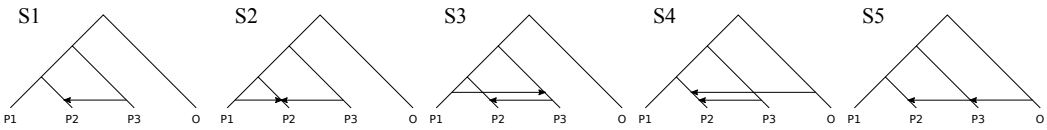


Figure 5 The standard D-Statistic scenario followed by four scenarios where an additional reticulation is added. S1 adds the first reticulation which is held constant throughout all scenarios and has M=0.1. The added reticulations in S2 through S5 have M=0.5.

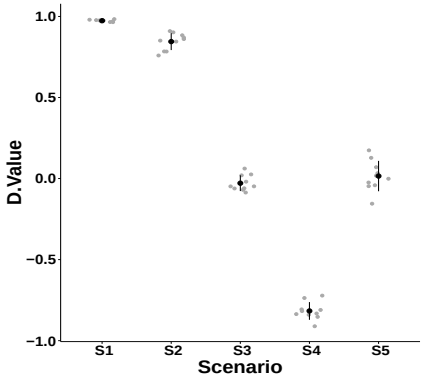


Figure 6 ***D*-Statistic values for scenarios with a "hidden" reticulation.** The scenarios S1–S5 are shown in Fig. 5.

reticulations simultaneously, which our D_{GEN} statistic allows for given that by its design it is not restricted to any specific number of reticulations.

3.1.2 D-Statistic Subsetting

When data sets with more than four taxa are to be analyzed by the *D*-statistic, a workaround is to subset the set of taxa into groups of four genomes (one outgroup and three in-group taxa) and conduct *D*-statistic analyses on each subset independently. Our method, being general, allows for analyzing the data set without any subsetting. The question we set out to investigate here is: Does subsetting and running the *D*-statistic on individual 4-taxon subsets equate to running D_{GEN} on the full data set? To answer this question, we considered the evolutionary scenario of Fig. 7.

In the case of the *D*-statistic, only two out of the ten simulations recovered a significant non-zero *D* value, whereas D_{GEN} inferred significant *D* values for all runs. This further demonstrates the need for and significance of a method that works directly on a full data set and accounting for multiple migration events.

3.2 Analysis Of a Mosquito Genomic Data Set

Finally, we present results from a real biological data set with six taxa. In both [8] and [26], the evolutionary history of the *Anopheles gambiae* species complex was found to be reticulate. Both studies found particularly strong signals of introgression in the 3L chromosome in an area known as the 3La inversion. This reticulation between *Anopheles quadriannulatus* (Q) and *Anopheles merus* (R) is shown in the network of Fig. 8(a) with the other species of *An. coluzzii* (C), *An. arabiensis* (A), *An. melas* (L), and *An. christyi* (O).

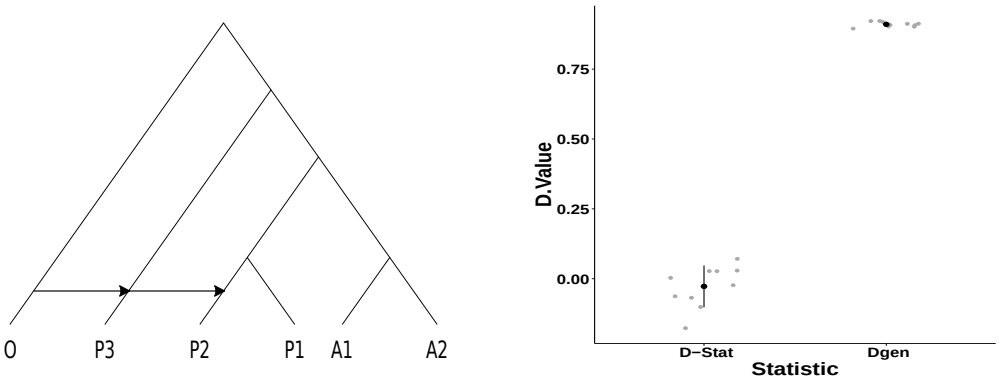
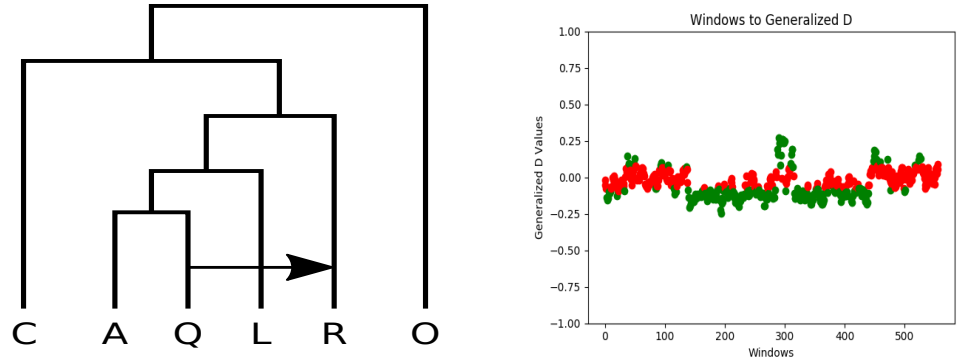


Figure 7 The effect of subsetting on the detectability of introgression. (Left) A 6-taxon evolutionary history with two migration events. (Right) The values of the D-statistic on subsets of four taxa, and D_{GEN} on the full data set.



(a) **(b)**

Figure 8 D_{GEN} values for the 3L mosquito chromosome. (a) Evolutionary history of six mosquito genomes. (b) D_{GEN} analysis of the 3L chromosome with window length set to 500kbp and with a 100kbp offset between windows.

The results from Fig. 8 (b) show that D_{GEN} does recover the introgressed region around the 3La inversion in comparison to the rest of the 3L chromosome, consistent with previous studies. Fig. 8(b) is an example of a figure that can be generated directly through the graphical user interface of the ALPHA toolkit [7] and is presented as generated directly from ALPHA. The figures output by ALPHA can be run on the full genome or on variable sized windows with variable sized offsets between windows. Here the window size used was 500kbp with a 100kbp offset between windows. The software can also vary the significance cutoff with which to display values as significant (green) or not significant (red). Here a significance cutoff value of 0.01 was used, as is used throughout the paper.

4 Discussion and Conclusions

In this paper, we extended the popular D-statistic to general cases of evolutionary histories of any number of taxa and any number and placement of migration events. What enabled this extension is the observation that the "ABBA-BABA" phylogenetic invariant underlying the D-statistic can be derived automatically by making use of the probability mass function of gene tree topologies under the multispecies coalescent and multispecies network coalescent models.

Our simulation results show that the new statistic D_{GEN} and method for deriving and computing it are very powerful for detecting introgression in various settings. In particular, we demonstrated that hidden migration events could negatively affect the performance of the D-statistic, which operates under the assumption of a single migration event. Furthermore, subsetting a data set of more than four taxa into data sets with four taxa is problematic. Our D_{GEN} statistic addresses these two issues by enabling the analysis of data sets with more than four taxa and more than a single migration event. While analyses in the style of the D-statistic make major assumptions, such as assuming the infinite sites model as well as ignoring dependence between sites, they are resilient to violations in these assumptions. Our results further support this given that our simulations violate both of these assumptions, having been performed under the full coalescent with recombination model with a mutation model allowing for recurrent mutation.

It is important to note that the D-statistic provides values that could be positive or negative. The sign of these values give an indication on the directionality of the migration in the case of four taxa. However, in the case of larger data sets, the sign of the D_{GEN} values is not easily interpretable in terms of directionality. It is also important to note that the actual D value is not the quantity of interest; rather, it is the statistical significance of its deviation from 0. There is of course, however, a strong correlation between the two.

As stated above, the method has been implemented in the ALPHA toolkit, which allows for conducting D_{GEN} analyses on the command-line as well as through a graphical user interface.

Finally, while we present the D_{GEN} statistic and its computation as a way of analyzing introgression under general evolutionary scenarios, computational complexity will become prohibitive for increasingly large, complex data sets. In particular, Step (3) in the algorithm above for computing D_{GEN} entails computing the probabilities of all gene tree topologies under a species tree and a phylogenetic network model. This calculation is very demanding, especially in the case of the phylogenetic network. For example, while generating D_{GEN} for four or five taxa takes approximately ten and forty seconds, respectively, generating it for six taxa takes thirteen minutes and for seven taxa thirty-eight hours. Fortunately, our implementation allows a D_{GEN} statistic to only ever need to be generated once for a particular evolutionary scenario, as the statistic itself is saved to a file that can be used on all current and future data sets for that scenario. This process of running a previously generated statistic on a new data set is, of course, computationally trivial. It will be important future work, however, to address the computational limits of D_{GEN} when going to arbitrarily large numbers of taxa.

References

1 M.L. Arnold. *Natural Hybridization and Evolution*. Oxford U. Press, 1997.

2 N.H. Barton. The role of hybridization in evolution. *Molecular Ecology*, 10(3):551–568, 2001.

3 P.D. Blischak, J. Chifman, A.D. Wolfe, and L.S. Kubatko. HyDe: a Python package for genome-scale hybridization detection. *Systematic Biology*, 2018.

4 J. H. Degnan and L. A. Salter. Gene tree distributions under the coalescent process. *Evolution*, 59:24–37, 2005.

5 J.H. Degnan and N.A. Rosenberg. Gene tree discordance, phylogenetic inference and the multispecies coalescent. *Trends in Ecology and Evolution*, 24(6):332–340, 2009.

6 Eric Y. Durand, Nick Patterson, David Reich, and Montgomery Slatkin. Testing for ancient admixture between closely related populations. *Molecular Biology and Evolution*, 28(8):2239–2252, 2011.

7 RA Leo Elworth, Chabrielle Allen, Travis Benedict, Peter Dulworth, and Luay Nakhleh. ALPHA: A toolkit for automated local phylogenomic analyses. *Bioinformatics*, 1:3, 2018.

8 Michael C Fontaine, James B Pease, Aaron Steele, Robert M Waterhouse, Daniel E Neafsey, Igor V Sharakhov, Xiaofang Jiang, Andrew B Hall, Flaminia Catteruccia, Evdoxia Kakani, Sara N. Mitchell, Yi-Chieh Wu, Hilary A. Smith, R. Rebecca Love, Mara K. Lawniczak, Michel A. Slotman, Scott J. Emrich, Matthew W. Hahn, and Nora J. Besansky. Extensive introgression in a malaria vector species complex revealed by phylogenomics. *Science*, 347(6217):1258524, 2015.

9 Richard E. Green, Johannes Krause, Adrian W. Briggs, Tomislav Maricic, Udo Stenzel, Martin Kircher, Nick Patterson, Heng Li, Weiwei Zhai, Markus Hsi-Yang Fritz, Nancy F. Hansen, Eric Y. Durand, Anna-Sapfo Malaspinas, Jeffrey D. Jensen, Tomas Marques-Bonet, Can Alkan, Kay Prafer, Matthias Meyer, Hern A. Burbano, Jeffrey M. Good, Rigo Schultz, Ayinuer Aximu-Petri, Anne Butthof, Barbara Hober, Barbara Hoffner, Madlen Siegemund, Antje Weihmann, Chad Nusbaum, Eric S. Lander, Carsten Russ, Nathaniel Novod, Jason Affourtit, Michael Egholm, Christine Verna, Pavao Rudan, Dejana Brajkovic, Oeljko Kucan, Ivan Guic, Vladimir B. Doronichev, Liubov V. Golovanova, Carles Lalueza-Fox, Marco de la Rasilla, Javier Fortea, Antonio Rosas, Ralf W. Schmitz, Philip L. F. Johnson, Evan E. Eichler, Daniel Falush, Ewan Birney, James C. Mullikin, Montgomery Slatkin, Rasmus Nielsen, Janet Kelso, Michael Lachmann, David Reich, and Svante Paabo. A draft sequence of the Neandertal genome. *Science*, 328(5979):710–722, 2010.

10 R. R. Hudson. Testing the constant-rate neutral allele model with protein sequence data. *Evolution*, 37:203–217, 1983.

11 Richard R Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.

12 T. Jukes and C. Cantor. Evolution of protein molecules. In H.N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academic Press, NY, 1969.

13 Laura Kubatko and Julia Chifman. An invariants-based method for efficient identification of hybrid species from large-scale genomic data. *bioRxiv*, page 034348, 2015.

14 J. Mallet. Hybridization as an invasion of the genome. *TREE*, 20(5):229–237, 2005.

15 J. Mallet. Hybrid speciation. *Nature*, 446:279–283, 2007.

16 J. Mallet, N. Besansky, and M.W. Hahn. How reticulated are species? *BioEssays*, 38(2):140–149, 2016.

17 P. Pamilo and M. Nei. Relationship between gene trees and species trees. *Mol. Bio. Evol.*, 5:568–583, 1998.

18 James B Pease and Matthew W Hahn. Detection and polarization of introgression in a five-taxon phylogeny. *Systematic biology*, 64(4):651–662, 2015.

19 Fernando Racimo, Sriram Sankararaman, Rasmus Nielsen, and Emilia Huerta-Sánchez. Evidence for archaic adaptive introgression in humans. *Nature Reviews Genetics*, 16(6):359–371, 2015.

20 Andrew Rambaut and Nicholas C Grass. Seq-gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Computer Applications in the Biosciences*, 13(3):235–238, 1997.

21 L. H. Rieseberg. Hybrid origins of plant species. *Annual Review of Ecology and Systematics*, 28:359–389, 1997.

22 Loren H Rieseberg, Olivier Raymond, David M Rosenthal, Zhao Lai, Kevin Livingstone, Takuya Nakazato, Jennifer L Durphy, Andrea E Schwarzbach, Lisa A Donovan, and Chris-

tian Lexer. Major ecological transitions in wild sunflowers facilitated by hybridization. *Science*, 301(5637):1211–1216, 2003.

23 Claudia Solís-Lemus and Cécile Ané. Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting. *PLoS Genet*, 12(3):e1005896, 2016.

24 N. Takahata. Gene genealogy in three related populations: Consistency probability between gene and population trees. *Genetics*, 122:957–966, 1989.

25 Dingqiao Wen and Luay Nakhleh. Co-estimating reticulate phylogenies and gene trees from multi-locus sequence data. *Systematic Biology*, 67(3):439–457, 2018.

26 Dingqiao Wen, Yun Yu, Matthew W Hahn, and Luay Nakhleh. Reticulate evolutionary history and extensive introgression in mosquito species revealed by phylogenetic network analysis. *Molecular Ecology*, 25(11):2361–2372, 2016.

27 Dingqiao Wen, Yun Yu, and Luay Nakhleh. Bayesian inference of reticulate phylogenies under the multispecies network coalescent. *PLoS Genetics*, 12(5):e1006006, 2016.

28 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.

29 Y. Yu and L. Nakhleh. A maximum pseudo-likelihood approach for phylogenetic networks. *BMC Genomics*, 16:S10, 2015.

30 Yun Yu, James H Degnan, and Luay Nakhleh. The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. *PLoS Genet*, 8(4):e1002660, 2012.

31 Yun Yu, Jianrong Dong, Kevin J Liu, and Luay Nakhleh. Maximum likelihood inference of reticulate evolutionary histories. *Proceedings of the National Academy of Sciences*, 111(46):16448–16453, 2014.

32 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.

33 Jiafan Zhu and Luay Nakhleh. Inference of species phylogenies from bi-allelic markers using pseudo-likelihood. *Bioinformatics*, 2018. (to appear).

34 Jiafan Zhu, Dingqiao Wen, Yun Yu, Heidi M Meudt, and Luay Nakhleh. Bayesian inference of phylogenetic networks from bi-allelic genetic markers. *PLoS Computational Biology*, 14(1):e1005932, 2018.

PRINCE: Accurate Approximation of the Copy Number of Tandem Repeats

Mehrdad Mansouri*
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
mansouri@sfu.ca

Julian Booth*
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
julius_booth@sfu.ca

Margaryta Vityaz*
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
rvityaz@sfu.ca

Cedric Chauve
Department of Mathematics, Simon Fraser University, Burnaby, BC, Canada
cedric_chauve@sfu.ca

Leonid Chindelevitch
School of Computing Science, Simon Fraser University , Burnaby, BC, Canada
leonid_chindelevitch@sfu.ca

Abstract

Variable-Number Tandem Repeats (VNTR) are genomic regions where a short sequence of DNA is repeated with no space in between repeats. While a fixed set of VNTRs is typically identified for a given species, the copy number at each VNTR varies between individuals within a species. Although VNTRs are found in both prokaryotic and eukaryotic genomes, the methodology called multi-locus VNTR analysis (MLVA) is widely used to distinguish different strains of bacteria, as well as cluster strains that might be epidemiologically related and investigate evolutionary rates.

We propose PRINCE (Processing Reads to Infer the Number of Copies via Estimation), an algorithm that is able to accurately estimate the copy number of a VNTR given the sequence of a single repeat unit and a set of short reads from a whole-genome sequence (WGS) experiment. This is a challenging problem, especially in the cases when the repeat region is longer than the expected read length. Our proposed method computes a statistical approximation of the local coverage inside the repeat region. This approximation is then mapped to the copy number using a linear function whose parameters are fitted to simulated data. We test PRINCE on the genomes of three datasets of *Mycobacterium tuberculosis* strains and show that it is more than twice as accurate as a previous method.

An implementation of PRINCE in the Python language is freely available at `https://github.com/WGS-TB/PythonPRINCE`.

2012 ACM Subject Classification Applied computing → Molecular sequence analysis

Keywords and phrases Variable-Number Tandem Repeats, Copy number, Bacterial genomics

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.20

* indicates equal contribution

Funding CC and LC are funded by NSERC Discovery grants and a CIHR/Genome Canada Bioinformatics/Computational Biology grant. LC is funded by a Sloan Foundation Fellowship.

Acknowledgements The authors would like to thank Ted Cohen and Vineet Bafna for helpful discussion and Jennifer Gardy and Jennifer Guthrie for providing assistance with the data used in our analysis.

1 Introduction

Variable number tandem repeats (VNTRs) are genomic locations where identical or highly similar sequences of DNA are repeated in tandem (i.e. with no spaces in between repeats). One reason for the importance of VNTRs in bacterial genomics is their use in the identification of related bacterial strains [2, 29]. For instance, 24 VNTR loci are used for *Mycobacterium tuberculosis* typing because of their reproducible nature and highly discriminatory typing results [30]. Related bacterial strains will typically have similar or identical copy numbers (CNs) at these loci, a feature used to identify clusters of potentially related strains in the molecular epidemiology of infectious diseases [20, 19]. In addition, the study of VNTR data has been used to glean information about bacterial lineage and pathogenicity [23].

The prediction of CNs for targeted VNTR regions from whole-genome sequencing (WGS) is a well-recognized problem, that needs to be solved in order to relate bacterial strains sequenced using WGS to those genotyped in the past with PCR-based methods [17]. Unlike the previously solved problem of reconstructing spoligotypes [5], this problem presents specific challenges. The difficulty arises from the fact that in many instances, the reads produced by WGS are too short to cover the entire repeat region [13]. Instead, they only cover small sections of the repeats and cannot be assembled to reconstruct the entire, multi-copy, VNTR region [27], preventing the direct resolution of CNs. However, reads generated from the VNTR region are likely to result in an apparent higher depth of coverage of the repeated pattern compared to the average depth of coverage for the rest of the genome, a signal that can be leveraged to predict the CN.

In theory, the "coverage depth ratio" (CDR) between the repeated pattern and the rest of the genome should be roughly equal to the CN of the VNTR region. However, several factors complicate matters. Repeat sequences can be similar to other regions within the genome, so a subsequence of a repeat sequence can appear in unrelated parts of the genome. What makes things even more challenging is that the repeated sequences may share subsequences with one another. As a result, the CDR between repeat and non-repeat regions can be highly influenced by similar, but unrelated, regions, biasing it toward higher values. Additionally, read errors and single-nucleotide polymorphisms (SNPs) between the sample and the reference repeated sequence can cause reads to not be identified as belonging to the VNTR region, biasing this ratio toward lower values.

There are three types of tandem repeats, which differ according to the length of the sequence being repeated [31], which we refer to as the *template* for the tandem repeat. The first one is short tandem repeats (STRs) or microsatellites, whose templates range between 1 and 6 base pairs. The second one, the focus of this paper, is VNTRs or minisatellites, whose templates are typically between 10 and 100 base pairs. The third type of repeat, copy number variations (CNVs), involves large templates of size 1000 base pairs and above.

There are a number of methods that work with microsatellites. These methods typically rely on the tandem repeat to be fully contained within the read or paired-end reads, and are therefore limited by their length. Examples of such methods include lobSTR [15] and

HipSTR [33]. In addition, a number of tools, such as STRViper [4] and STRait Razor [34], focus on identifying repeat templates, rather than predicting the CN for a given template.

A wide range of algorithms has also been developed for CNVs. They generally fall into five categories: paired-end mapping, split read, read depth, de novo assembly of a genome, and combination of the above approaches. Read depth based methods are typically used for CNV detection, and the rest are typically used for CNV identification [36]. Read depth methods typically rely on dividing the genome into windows of at least 100 base pairs [35]. However, this cannot provide enough resolution for VNTRs, whose templates typically have a length smaller than 100 base pairs. Another issue is that these methods tend to pre-train their model on the human genome, and are not directly applicable to bacterial genomes [35, 1].

Relatively few methods have been developed to address the problem of predicting CNs of minisatellites (VNTRs) from WGS data. To the best of our knowledge, there are only two such methods that are broadly applicable: CNVeM [32] and ExpansionHunter [7]. CNVem uses a probabilistic model that utilizes the inherent uncertainty of read mapping and uses maximum likelihood to estimate locations and copy numbers. ExpansionHunter is designed to work with PCR-free WGS short-read data. It distinguishes three categories of reads: spanning, flanking and in-repeat reads in a BAM file and used basic statistical techniques to estimate the copy number from them. Unfortunately, we were not able to successfully apply CNVeM to our data, and thus report on the results of comparing our method to ExpansionHunter in this paper.

Three other existing methods work in a similar context to ours and are worth mentioning. The first one of these methods, TGS-TB [28], was specifically developed to work for *Mycobacterium tuberculosis*, the organism which we use as the pilot application in our work. However, it requires a minimum read length of at least 300 base pairs, which exceeds the most commonly used short read technologies, and is not applicable to our data. The second one, adVNTR [3], is based on a Hidden Markov model (HMM) trained on the human genome. This method is not directly comparable to ours as it uses not only the template sequence, but also information about the region flanking each VNTR. The final one, VNTRseek [12], is designed for detecting VNTRs, not predicting the CNs of VNTRs with known templates.

In this paper we propose a method called PRINCE (Processing Reads to Infer the Number of Copies Exactly), which successfully addresses the challenges of determining CNs for VNTR in bacterial genomes. PRINCE accomplishes the goal of VNTR CN estimation by training a model using reads simulated from a reference genome with computationally "spiked-in" VNTRs that have known copy numbers. For each template, the prediction takes place in two stages, the first of which is independent of the considered reference genome and depends only on the template while the second one is reference-dependent. In the first stage, reads are recruited in a computationally efficient manner by finding exact k-mer matches to the template within the reads. The recruited reads are then compared to the template to compute an overall matching score based on coverage and sequence features. In the second stage, PRINCE fits the dependency of the known CNs on the matching score using linear regression. For an input consisting of previously unseen WGS data, the reads are recruited in the same way and the CNs are calculated from the resulting match score using the fitted parameters.

We have tested PRINCE on two datasets consisting of *M. tuberculosis* genome reads (135 samples in total) as well as a simulated dataset. We show that the CNs estimated by PRINCE are consistently closer to the true CNs than the ones predicted by ExpansionHunter, and that the estimation remains robust for a range of coverages, read qualities and copy numbers. PRINCE is freely available at `https://github.com/WGS-TB/PythonPRINCE`.

2 Methods

2.1 Problem Description

The objective of PRINCE is to find the CNs within VNTRs in a genome exclusively from a set of short reads, a set of templates (one per VNTR), and a reference genome. In this section we define the relevant terminology.

Definition 1: A *template* is the repeat unit of a VNTR, sometimes also referred to as the pattern of a VNTR. We define T as the set of templates, i.e. $T := \{t_i\}_{i=1}^{m}$, where t_i is the i-th template and m is the number of templates.

Definition 2: The *copy number* c_i is the number of times a template t_i is repeated in tandem in a genome G, possibly with errors. We define $\mathbf{c}$ as the vector of the c_i's corresponding to each t_i in T, and write $\mathbf{c} := [c_1, \ldots, c_m]$.

Definition 3: The *match score* s_i is a proxy for the copy number of a template computed by PRINCE. We define $\mathbf{s}$ as the vector of match scores corresponding to each t_i in T, and write $\mathbf{s} := [s_1, \ldots, s_m]$.

Definition 4: A *read* r_j is a subsequence of length L of a genome G, possibly corrupted by sequencing errors. We define R as the set of reads, i.e. $R := \{r_j\}_{j=1}^{n}$, where n is the number of reads.

Given an assembled reference genome G_r for training and the template set T, PRINCE infers the parameters of a linear model for predicting copy numbers of the templates in the set T. Using this model, given an input of a set of reads R from a genome $G \neq G_r$, PRINCE estimates the copy number vector $\mathbf{c}$, i.e. the number of times c_i that the i-th template $t_i \in T$ is repeated in tandem in the genome G, for each of the VNTRs.

2.2 Overview

PRINCE can be thought of as the function composition $\mathbf{g}(f(R))$. Let $\mathcal{R}$ represent the domain of read sets, S represent the domain of match score vectors and C represent the domain of copy number vectors. Then

$$f\colon \mathcal{R} \to S$$
$$\mathbf{g}\colon S \to C$$

A different g_i is fitted to each t_i and is dependent of the reference genome G, whereas f is a recruitment and scoring algorithm that works in the same way for all the templates and is independent of the reference genome G. The process PRINCE uses is shown in Figure 1.

2.3 Parameter Fitting and Copy Number Prediction

PRINCE's $\mathbf{g}$ function is fitted using simulated reads, generated from the given reference genome G. This is done to account for all possible copy numbers. Using a range of copy numbers at each VNTR allows PRINCE to uncover the true relationship between match score and copy number. The relationship is assumed to be linear, which follows from the assumption that the depth of coverage is approximately uniform throughout the genome.

We generated artificial genomes by removing the VNTR regions in assembled genomes and inserting varying but known numbers of copies of each template back into the genome. Simulated reads were generated from these artificial genomes using the ART software [16].

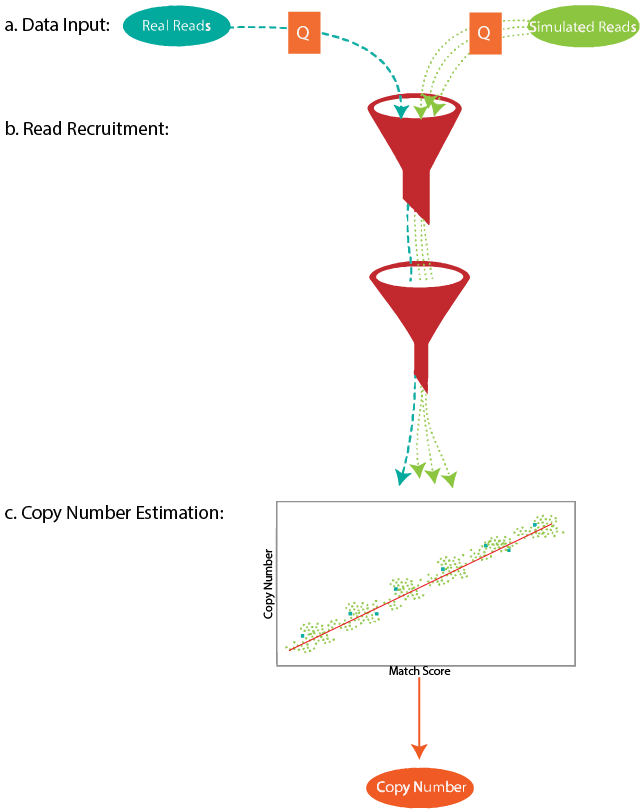


Figure 1 Flowchart of PRINCE. a. Reads are divided into mini-reads and low-quality mini-reads are discarded. b. Reads are compared against templates in two stages and match scores are computed. c. PRINCE uses the match scores to estimate copy numbers via a linear regression model.

From these artificial genome reads, PRINCE computes a vector of match scores $\mathbf{s} = f(r)$, as is described in the next section. PRINCE fits the linear function

$$g_i(s_i) = a_i \cdot s_i - b_i = c_i$$

for each template using a linear regression [10], and uses this function to estimate copy numbers. By fitting the coverage depth ratio a_i, PRINCE becomes more robust to template-specific coverage biases such as PCR bias or substitution errors, which can be influenced by template length [8, 26]. PRINCE also accounts for the presence of homologous short sequences that may be erroneously added to s_i during read recruitment by fitting the constant b_i to be subtracted. This assumes that the erroneously recruited reads are from stable genomic regions, an assumption that appears to hold given our results.

The parameters used in PRINCE by default come from two sets of simulated reads generated from 40 artificial *Mycobacterium tuberculosis* genomes each, made from 8 real assembled genomes [25, 22] that had 1 to 5 copies of each template t_i inserted, for a total of 80 sample points. The template set T we use is the standard 24-locus MIRU-VNTR [29, 30].

2.4 Read Recruitment and Match Score Computation

In order to determine whether a read comes from a given template, we compare one to the other. This can be a computationally expensive task. To overcome this, PRINCE divides each read into its k-splits (non-overlapping k-mers), which are defined as consecutive

non-overlapping substrings of length k. Here we use $k = 25$ as most commonly used read lengths are divisible by 25. We call each of the k-splits a "mini-read". In the first step, PRINCE discards low quality mini-reads, as measured by the average Phred score [9]. We use a previously suggested minimum quality threshold of 20 [18].

PRINCE then performs a k-mer matching step. In this step all the k-mers of a mini-read, as well as its reverse complement (to allow for paired-end data), are compared to the unique k-mers of the template concatenated with itself (to allow for mini-reads that fall on the boundary between two copies) using a hash table. For this step, we use the smaller value $k = 9$, chosen via calibration. To ensure that the mini-read almost certainly comes from the VNTR region, only those reads with all k-mers matching a template k-mer (as determined via the hash table) are recruited.

The fraction of mini-reads that are recruited for template t_i out of all the ones that survive the quality filtering step gives the match score s_i. The normalization by the number of surviving mini-reads ensures that all the read sets from the same genome have the same expected match score for the same VNTR, no matter what the read length or depth of coverage is.

2.5 Simulated Data Generation

We generated a simulated dataset for testing PRINCE. 18 sets of reads were generated from 3 fully assembled genomes (Beijing-391, Beijing-like-1104, Beijing-like-35049) [25] of 6 different kinds using ART: HiSeq 2500 profile with read length of 50bp, 100bp and 150bp, HiSeqX PCR free with read length of 150 bp, and MiSeq v3 with read lengths of 200bp and 250bp.

3 Results

We have evaluated our method on three different datasets: the BC dataset (accession numbers are in Table 1 of the Supplementary Materials), the Beijing dataset, and a simulated dataset. Each one enabled us to explore the performance of our method under various conditions.

The BC dataset is a subset of the dataset collected in British Columbia from 2005 to 2014 [14], and contains 5 sets of high quality, high coverage WGS reads, each one containing 5 *Mycobacterium tuberculosis* samples sharing the same MIRU-VNTR pattern. This dataset allowed us to test PRINCE under optimal conditions. The Beijing dataset [21] contains 110 genomes belonging to the Beijing lineage of *Mycobacterium tuberculosis*, with variable quality and coverage, allowing us to explore their effects on performance. The copy numbers for these datasets were established using standard experimental protocols for VNTR copy number prediction [11] which are known to occasionally have errors. According to one study, when done with a commercial kit, they have 88% reproducibility, and have lower reproducibility when done using other methods [6]. According to another study, some loci (templates) are more reproducible than others: reproducibility agreement rates are 98.9% and 91.3% for standard and hypervariable loci, respectively [24].

The reason for introducing the simulated dataset is to be able to verify the method on a dataset with known copy numbers (ground truth). We have attempted to compare PRINCE to ExpansionHunter and CNVeM. Unfortunately, we were unable to use CNVeM as we could not successfully compile its source code. ExpansionHunter was designed to work for diploid organisms, and outputs two estimates and their confidence intervals. In order to correctly interpret the results, we used the median of the widest confidence interval, as suggested by the authors of ExpansionHunter.

3.1 Performance

We expected PRINCE to perform better on simulated reads than on real reads as it was trained on reads produced by ART. Indeed, PRINCE had a mean absolute error (MAE) of 0.57 over all 24 VNTR regions and 18 simulated genomes, but the MAE increased to 0.80 for both the BC and Beijing datasets, which is somewhat worse than the simulated data. The drop in performance is likely due to dissimilarities between the simulated training data and the real datasets, due to factors that may include a higher presence of read errors and varying coverage in the real data. Fortunately, PRINCE accounts for differences in input in its match score. We tested how well it adapts to variability in the input on the Beijing dataset, which had the most variation in coverage and read quality.

There was no change in PRINCE's performance across different coverages ($R^2 = 0.002$) (Fig. 3). We expect that performance might sharply decline at excessively low coverages as the variance of depth along the genome becomes more significant. However, for all reasonable coverages PRINCE's performance remains consistent.

PRINCE performed well on lower quality datasets (Fig. 4). However there was a slight increase in MAE as read quality diminished ($R^2 = 0.128$). We measured dataset quality by measuring the average Phred score per nucleotide.

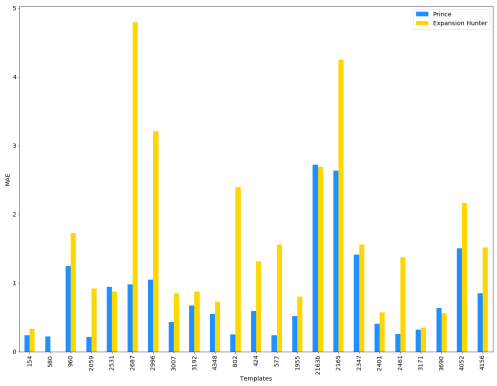
PRINCE was trained on copy numbers between 1 and 5. Some real datasets have VNTR regions with copy numbers above 10. We wanted to know how well PRINCE can extrapolate to higher copy numbers. We simulated 27 datasets using ART from genomes with copy number insertions ranging from 1 to 90 (Fig. 5). PRINCE exhibits a linear increase in MAE as copy number increases. However, PRINCE still has an MAE under 1 for copy numbers below 15. No VNTR in any genome from the two real datasets we used had a VNTR region with a copy number above 12. PRINCE may perform worse on higher copy numbers because g_i is trained on small copy numbers. Errors in the fitted parameter a_i are magnified as s_i increases, like when the number of copies increases. Although we only chose to train PRINCE with copy numbers from 1 to 5, PRINCE still performed well on the copy numbers seen in our testing datasets. PRINCE can be trained with arbitrarily high copy numbers, as long as the length of the repeat region does not become significant relative to the genome length and affect the depth of coverage calculation. Training PRINCE with high copy number data may be necessary when estimating high copy number VNTR regions.

3.2 Comparison with ExpansionHunter

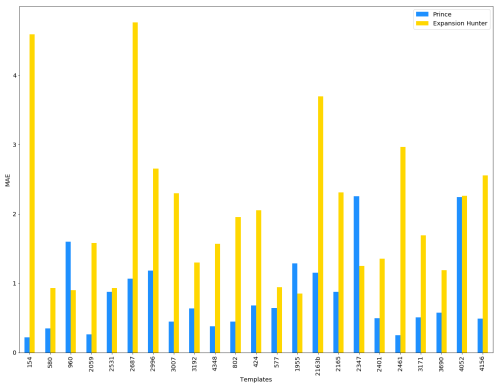
PRINCE outperforms ExpansionHunter on all three datasets. ExpansionHunter had an MAE of 1.80 for the simulated dataset and 1.62 and 2.07 for the BC and Beijing datasets respectively, an error over twice that of PRINCE in each case. ExpansionHunter performs particularly poorly on the Beijing dataset, most likely due to the variable read quality.

We performed a per-template comparison with ExpansionHunter on our three datasets. PRINCE greatly outperforms ExpansionHunter on most templates, with a few exceptions.

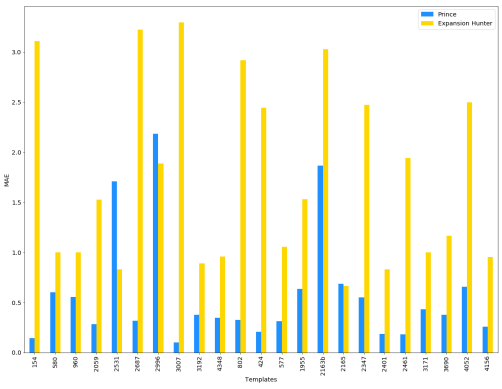
More specifically, PRINCE performs worse than ExpansionHunter on locus 2163b on the BC dataset (Fig. 2a). However, PRINCE performs relatively well on 2163b in the Beijing dataset (PRINCE : 1.15, ExpansionHunter : 3.70) (Fig. 2b). The true values for these datasets were determined using traditional typing methods. Each locus is PCR-amplified using primers that align to the flanking regions of the VNTR, the resulting PCR products are then measured by standard gel electrophoresis and a copy number is inferred based on its size [11]. The insertion of DNA between VNTR flanking regions that does not resemble the template may cause some true values to be mislabelled. The average true value for 2163b in



(a) BC



(b) Beijing



(c) Simulated

Figure 2 MAE for PRINCE and ExpansionHunter for BC, Beijing and simulated datasets.

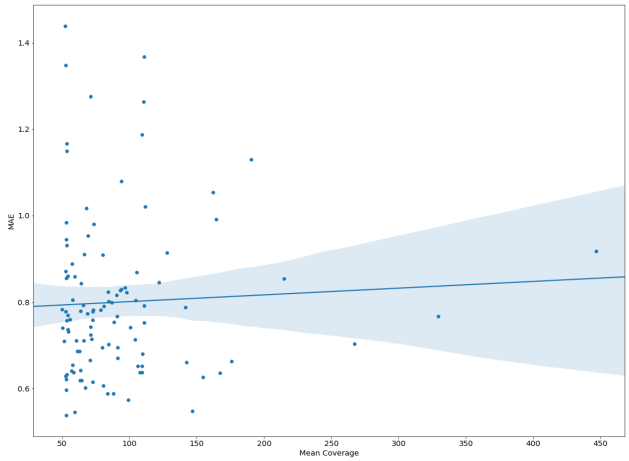


Figure 3 MAE as a function of coverage over the Beijing dataset.

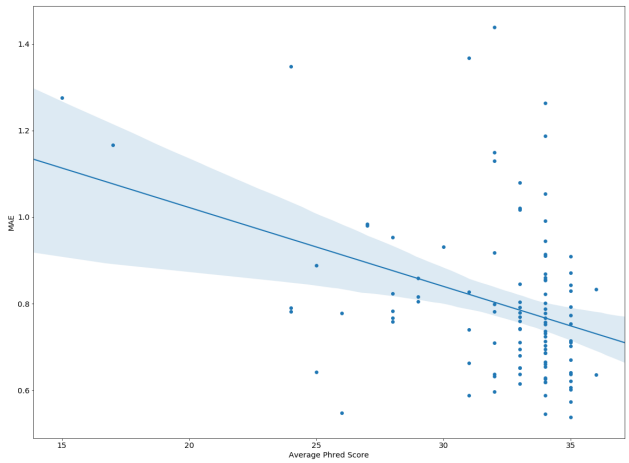


Figure 4 MAE as a function of Phred score over the Beijing dataset.

the Beijing dataset was 5.51 compared to 3.00 in the BC dataset. The poor performance of PRINCE on 2163b might be due to incorrectly labeled true values. The poor performance seen on loci 2163b, 2531 and 2996 in the simulated dataset (Fig. 2c), is more curious, as the reads come from assembled genomes. This may be due to using only three samples. Additionally, the three loci have the highest average copy number of all templates in the simulated genomes, at 8.66, 5 and 7 copies respectively. Again, training PRINCE with higher copy numbers may improve performance on these loci.

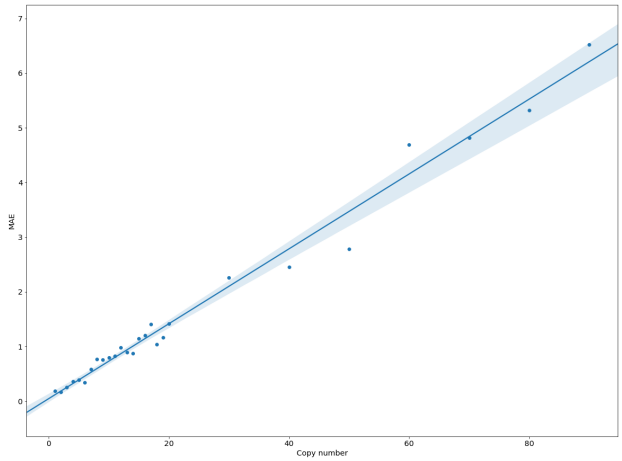


Figure 5 MAE as a function of copy number over the simulated dataset. Note that the relative error remains relatively constant over the full range of copy numbers explored.

3.3 Running Time

With Illumina paired-end WGS data at 50x coverage, PRINCE takes an average of 156 seconds per genome to query. Training takes the same amount of time per genome; however, training may use hundreds of genomes. PRINCE allows for each genome to be processed in parallel, and we were able to train PRINCE using 80 genomes in under 5 minutes.

4 Conclusion

PRINCE provides an accurate estimation of the CN within a VNTR region. PRINCE fits a linear regression to the relationship between CN and the estimated depth of coverage at each loci using simulated data. We provide an example of how it could be used to estimate the copy numbers for *Mycobacterium tuberculosis* genomes, where these values could be used to compare bacteria sequenced with WGS technology to those interrogated only at their VNTR loci. In the past, this could only be done with an experimental technique specialized for tandem repeat amplification, rather than computationally from WGS data.

As input, in addition to a reference genome PRINCE only requires a set of reads and a set of templates. Unlike the method described in TGS-TB [28] that computes copy numbers exclusively for *Mycobacterium tuberculosis*, and requires reads of a minimum length of 300 base pairs and computes the copy number exclusively for predefined templates. PRINCE can be trained on any bacterial genome, can work with very short reads, and can use any set of templates.

Our work presents the first software tool that is designed to accurately infer copy numbers of VNTR templates directly from WGS data for any bacterial species. We believe that this will open the door to a comparison between historical isolates and modern-day ones, and facilitate the extraction of novel insights on the epidemiology and transmission patterns of important bacterial pathogens such as *Mycobacterium tuberculosis*, which we use here as an illustrative example.

References

1 A Abyzov, A E Urban, M Snyder, and M Gerstein. CNVnator: an approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing. *Genome Research*, 21(6):974–984, 2011.

2 Lindstedt B. Multiple-locus variable number tandem repeats analysis for genetic fingerprinting of pathogenic bacteria. *Electrophoresis*, 26(13):2567–2582, 2005.

3 M Bakhtiari, S Shleizer-Burko, M Gymrek, V Bansal, and V Bafna. Targeted genotyping of variable number tandem repeats with adVNTR. *bioRxiv*, 2017.

4 MD Cao, E Tasker, K Willadsen, M Imelfort, S Vishwanathan, et al. Inferring short tandem repeat variation from paired-end short reads. *Nucleic Acids Research*, 42(3):e16–e16, 2013.

5 F Coll, K Mallard, MD Preston, S Bentley, J Parkhill, et al. SpolPred: rapid and accurate prediction of *Mycobacterium tuberculosis* spoligotypes from short genomic sequences. *Bioinformatics*, 28(22):2991–2993, 2012.

6 JL De Beer, K Kremer, C Ködmön, P Supply, D Van Soolingen, Global Network for the Molecular Surveillance of Tuberculosis 2009, et al. First worldwide proficiency study on variable-number tandem-repeat typing of *Mycobacterium tuberculosis* complex strains. *Journal of Clinical Microbiology*, 50(3):662–669, 2012.

7 E Dolzhenko, JJFA van Vugt, RJ Shaw, MA Bekritsky, M van Blitterswijk, et al. Detection of long repeat expansions from PCR-free whole-genome sequence data. *Genome Research*, 27(11):1895–1903, 2017.

8 M Escalona, S Rocha, and D Posada. A comparison of tools for the simulation of genomic next-generation sequencing data. *Nature Reviews Genetics*, 17(8):459, 2016.

9 B Ewing, L Hillier, MC Wendl, and P Green. Base-calling of automated sequencer traces using Phred. I. Accuracy assessment. *Genome Research*, 8(3):175–185, 1998.

10 J Friedman, T Hastie, and R Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer Series in Statistics New York, 2001.

11 R Frothingham and WA Meeker-O'Connell. Genetic diversity in the *Mycobacterium tuberculosis* complex based on variable numbers of tandem DNA repeats. *Microbiology*, 144(5):1189–1196, 1998.

12 Y Gelfand, Y Hernandez, J Loving, and G Benson. VNTRseek - a computational tool to detect tandem repeat variants in high-throughput sequencing data. *Nucleic Acids Research*, 42(14):8884–8894, 2014.

13 S Goodwin, JD McPherson, and WR McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.

14 JL Guthrie, C Kong, D Roth, D Jorgensen, M Rodrigues, et al. Molecular epidemiology of tuberculosis in British Columbia, Canada–a 10-year retrospective study. *Clinical Infectious Diseases*, 2017.

15 M Gymrek, D Golan, S Rosset, and Y Erlich. lobSTR: a short tandem repeat profiler for personal genomes. *Genome Research*, 22(6):1154–1162, 2012.

16 W Huang, L Li, JR Myers, and GT Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

17 T Jagielski, J van Ingen, N Rastogi, J Dziadek, PK Mazur, and J Bielecki. Current methods in the molecular typing of *Mycobacterium tuberculosis* and other mycobacteria. *BioMed Research International*, 2014(645802), 2014.

18 P Liao, GA Satten, and Y Hu. PhredEM: a Phred-score-informed genotype-calling approach for next-generation sequencing studies. *Genetic Epidemiology*, 41(5):375–387, 2017.

19 B Mathema, NE Kurepina, PJ Bifani, and BN Kreiswirth. Molecular epidemiology of *Tuberculosis*: Current Insights. *Clinical Microbiology Reviews*, 19(4):658–685, 2006.

20 CJ Meehan, P Moris, TA Kohl, J Pečerska, S Akter, et al. The relationship between transmission time and clustering methods in *Mycobacterium tuberculosis* epidemiology. *bioRxiv*, 2018.

21 M Merker, C Blin, S Mona, N Duforet-Frebourg, S Lecher, et al. Evolutionary history and global spread of the *Mycobacterium tuberculosis* Beijing lineage. *Nature Genetics*, 47(3):242–249, 2015.

22 T Miyoshi-Akiyama, K Satou, M Kato, A Shiroma, K Matsumura, et al. Complete annotated genome sequence of *Mycobacterium tuberculosis* (Zopf) Lehmann and Neumann (ATCC35812)(Kurono). *Tuberculosis*, 95(1):37–39, 2015.

23 CA Nadon, E Trees, LK Ng, E Møller Nielsen, A Reimer, et al. Development and application of MLVA methods as a tool for inter-laboratory surveillance. *Euro Surveillance*, 18(35), 2013.

24 V Nikolayevskyy, A Trovato, A Broda, E Borroni, D Cirillo, and F Drobniewski. MIRU-VNTR genotyping of *Mycobacterium tuberculosis* strains using QIAxcel technology: A multicentre evaluation study. *PLoS One*, 11(3):e0149435, 2016.

25 JG Rodríguez, C Pino, A Tauch, and MI Murcia. Complete genome sequence of the clinical Beijing-like strain *Mycobacterium tuberculosis* 323 using the PacBio real-time sequencing platform. *Genome Announcements*, 3(2):e00371–15, 2015.

26 MG Ross, C Russ, M Costello, A Hollinger, NJ Lennon, et al. Characterizing and measuring bias in sequence data. *Genome Biology*, 14(5):R51, 2013.

27 SL Salzberg and JA Yorke. Beware of mis-assembled genomes. *Bioinformatics*, 21(24):4320–4321, 2005.

28 T Sekizuka, A Yamashita, Y Murase, T Iwamoto, S Mitarai, S Kato, and M Kuroda. TGS-TB: Total genotyping solution for *Mycobacterium tuberculosis* Using Short-Read Whole-Genome Sequencing. *PLoS One*, 10(11):e0142951, 2015.

29 P Supply. Multilocus Variable Number Tandem Repeat genotyping of *Mycobacterium tuberculosis*. Technical report, Institut de Biologie/Institut Pasteur de Lille, 2005.

30 P Supply, C Allix, S Lesjean, M Cardoso-Oelemann, S Rüsch-Gerdes, et al. Proposal for standardization of optimized mycobacterial interspersed repetitive unit-variable-number tandem repeat typing of *Mycobacterium tuberculosis*. *Journal of Clinical Microbiology*, 44(12):4498–4510, 2006.

31 DW Ussery, TM Wassenaar, and S Borini. *Computing for Comparative Microbial Genomics: Bioinformatics for Microbiologists*, volume 8 of *Computational Biology*. Springer, 2009.

32 Z Wang, F Hormozdiari, W Yang, E Halperin, and E Eskin. CNVeM: copy number variation detection using uncertainty of read mapping. *Journal of Computational Biology*, 20(3):224–236, 2013.

33 T Willems, D Zielinski, J Yuan, A Gordon, M Gymrek, and Y Erlich. Genome-wide profiling of heritable and de novo STR variations. *Nature Methods*, 14(6):590, 2017.

34 AE Woerner, JL King, and B Budowle. Fast STR allele identification with STRait Razor 3.0. *Forensic Science International: Genetics*, 30:18–23, 2017.

35 S Yoon, Z Xuan, V Makarov, K Ye, and J Sebat. Sensitive and accurate detection of copy number variants using read depth of coverage. *Genome Research*, 19(9):1586–1592, 2009.

36 M Zhao, Q Wang, Q Wang, P Jia, and Z Zhao. Computational tools for copy number variation (CNV) detection using next-generation sequencing data: features and perspectives. *BMC Bioinformatics*, 14(11):S1, 2013.

Supplementary Materials

Table 1 SRA accession numbers for the BC data and the corresponding 24 MIRU-VNTR patterns.

SRS2774518	ERS1062942	SRS2774801	SRS2774445	SRS2774521	224325153323444234423373
SRS2774727	SRS2774503	SRS2774647	SRS2774692	SRS2774726	234315153323441444223352
SRS2774500	SRS2774680	SRS2774388	SRS2774715	SRS2774747	223325163533245544423382
SRS2774536	SRS2577355	SRS2774746	SRS2774387	SRS2774426	228225113221343244423383
SRS2577413	SRS2577389	SRS2577020	SRS2577272	SRS2577201	225425173533524244223384

Degenerate String Comparison and Applications

Mai Alzamel
Department of Informatics, King's College London, UK and Department of Computer Science, King Saud University, KSA
mai.alzamel@kcl.ac.uk

Lorraine A. K. Ayad
Department of Informatics, King's College London, UK
lorraine.ayad@kcl.ac.uk

Giulia Bernardini[1]
Department of Informatics, Systems and Communication (DISCo), University of Milan-Bicocca, Italy
giulia.bernardini@unimib.it

Roberto Grossi[2]
Department of Computer Science, University of Pisa, Italy and ERABLE Team, INRIA, France
grossi@di.unipi.it

Costas S. Iliopoulos
Department of Informatics, King's College London, UK
costas.iliopoulos@kcl.ac.uk

Nadia Pisanti[3]
Department of Computer Science, University of Pisa, Italy and ERABLE Team, INRIA, France
pisanti@di.unipi.it

Solon P. Pissis[4]
Department of Informatics, King's College London, UK
solon.pissis@kcl.ac.uk

Giovanna Rosone[5]
Department of Computer Science, University of Pisa, Italy
giovanna.rosone@unipi.it

Abstract

A generalised degenerate string (GD string) $\hat{S}$ is a sequence of n sets of strings of total size N, where the ith set contains strings of the same length k_i but this length can vary between different sets. We denote the sum of these lengths $k_0, k_1, \ldots, k_{n-1}$ by W. This type of uncertain sequence can represent, for example, a gapless multiple sequence alignment of width W in a compact form. Our first result in this paper is an $\mathcal{O}(N+M)$-time algorithm for deciding whether the intersection

[1] Partially supported by the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".
[2] Partially supported by the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".
[3] Partially supported by the project MIUR-SIR CMACBioSeq "Combinatorial methods for analysis and compression of biological sequences" grant n. RBSI146R5L and the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".
[4] Partially supported by the Royal Society project IE 161274 "Processing uncertain sequences: combinatorics and applications".
[5] Partially supported by the project MIUR-SIR CMACBioSeq "Combinatorial methods for analysis and compression of biological sequences" grant n. RBSI146R5L, the Royal Society project IE 161274 "Processing uncertain sequences: combinatorics and applications", and the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen; Article No. 21; pp. 21:1–21:14

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

of two GD strings of total sizes N and M, respectively, over an integer alphabet, is non-empty. This result is based on a combinatorial result of independent interest: although the intersection of two GD strings can be exponential in the total size of the two strings, it can be represented in only *linear* space. A similar result can be obtained by employing an automata-based approach but its cost is alphabet-dependent. We then apply our string comparison algorithm to compute palindromes in GD strings. We present an $\mathcal{O}(\min\{W, n^2\}N)$-time algorithm for computing all palindromes in $\hat{S}$. Furthermore, we show a similar conditional lower bound for computing maximal palindromes in $\hat{S}$. Finally, proof-of-concept experimental results are presented using real protein datasets.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases degenerate strings, generalised degenerate strings, elastic-degenerate strings, string comparison, palindromes

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.21

1 Introduction

A *degenerate string* (or indeterminate string) over an alphabet Σ is a sequence of subsets of Σ. A great deal of research has been conducted on degenerate strings (see [1, 11, 20, 29, 32] and references therein). These types of uncertain sequences have been used extensively for flexible modelling of DNA sequences known as IUPAC-encoded DNA sequences [23].

In [19], the authors introduced a more general definition of degenerate strings: an *elastic-degenerate string* (ED string) $\tilde{S}$ over Σ is a sequence of subsets of Σ^* (see also network expressions [28]) with the aim of representing multiple genomic sequences [10]. That is, any set of $\tilde{S}$ does not contain, in general, only letters; a set may also contain strings, including the empty string. In a few recent papers on this notion, the authors provided several algorithms for pattern matching; specifically, for finding all exact [17] and approximate [8] occurrences of a standard string pattern in an ED text.

We introduce here another special type of uncertain sequence called generalised degenerate string; this can be viewed as an extension of degenerate strings or as a restricted variant of ED strings. Formally, a *generalised degenerate string* (GD string) $\hat{S}$ over Σ is a sequence of n sets of strings over Σ of total size N, where the ith set contains strings of the same length $k_i > 0$ but this length can vary between different sets. We denote the sum of these lengths $k_0, k_1, \ldots, k_{n-1}$ by W. Thus a GD string can be used to represent a *gapless* multiple sequence alignment (MSA) of fixed width, that is, for example, a high-scoring local alignment of multiple sequences, in a compact form; see Figure 1. This type of alignment is used for finding *functional* sequence elements [14]. For instance, searching for palindromic motifs in these type of alignments is an important problem since many transcription factors bind as homodimers to palindromes [26]. Specifically, a set of virus species can be clustered using high-scoring MSA to obtain subsets of viruses that have a *common* hairpin structure [27].

Our motivation for this paper comes from finding palindromes in these types of uncertain sequences. Let us start off with standard strings. A palindrome is a sequence that reads the same from left to right and from right to left. Detection of palindromic factors in texts is a classical and well-studied problem in algorithms on strings and combinatorics on words with a lot of variants arising out of different practical scenarios. In molecular biology, for instance, palindromic sequences are extensively studied: they are often distributed around promoters, introns, and untranslated regions, playing important roles in gene regulation and other cell

```
CA--AGCTCTATCTCGTA--TT
C---AGCCGAAGCTCGTATATT
CATCAAGTCAACGCAG----TT
```

(a) Multiple sequence alignment.

```
AGCTCTATCTCG
AGCCGAAGCTCG
AAGTCAACGCAG
```

(b) Local gapless alignment.

$$\hat{S} = \{\underline{\mathtt{A}}\} \cdot \begin{Bmatrix} \underline{\mathtt{GC}} \\ \mathtt{AG} \end{Bmatrix} \cdot \begin{Bmatrix} \mathtt{TCT} \\ \underline{\mathtt{CGA}} \\ \mathtt{TCA} \end{Bmatrix} \cdot \{\mathtt{A}\} \cdot \begin{Bmatrix} \mathtt{TCTC} \\ \underline{\mathtt{GCTC}} \\ \mathtt{CGCA} \end{Bmatrix} \cdot \{\underline{\mathtt{G}}\}$$

(c) GD string obtained from the local gapless alignment.

Figure 1 A GD string representing a gapless multiple sequence alignment.

processes (e.g. see [4]). In particular these are strings of the form $X\bar{X}^R$, also known as complemented palindromes, occurring in single-stranded DNA or, more commonly, in RNA, where X is a string and $\bar{X}^R$ is the reverse complement of X. In DNA, `C-G` are complements and `A-T` are complements; in RNA, `C-G` are complements and `A-U` are complements.

A string $X = X[0]X[1]\dots X[n-1]$ is said to have an initial palindrome of length k if its prefix of length k is a palindrome. Manacher first discovered an on-line algorithm that finds all initial palindromes in a string [25]. Later Apostolico et al observed that the algorithm given by Manacher is able to find all maximal palindromic factors in the string in $\mathcal{O}(n)$ time [6]. Gusfield gave an off-line linear-time algorithm to find all maximal palindromes in a string and also discussed the relation between biological sequences and gapped palindromes [18].

For uncertain sequences, we first need to have an algorithm for efficient *string comparison*, where automata provide the following baseline. Let $\hat{X}$ and $\hat{Y}$ be two GD (or two ED) strings of total sizes N and M, respectively. We first build the non-deterministic finite automaton (NFA) A of $\hat{X}$ and the NFA B of $\hat{Y}$ in time $\mathcal{O}(N+M)$. We then construct the product NFA C such that $L(C) = L(A) \cap L(B)$ in time $\mathcal{O}(NM)$. The non-emptiness decision problem, namely, checking if $L(C) \neq \emptyset$, is decidable in time linear in the size of C, using breadth-first search (BFS). Hence the comparison of $\hat{X}$ and $\hat{Y}$ can be done in time $\mathcal{O}(NM)$. It is known that if there existed faster methods for obtaining the automata intersection, then significant improvements would be implied to many long standing open problems [24]. Hence an immediate reduction to the problem of NFA intersection does not particularly help. For GD strings we show at the beginning of Section 3 that we can build an ad-hoc deterministic finite automaton (DFA) for $\hat{X}$ and $\hat{Y}$, so that the intersection can be performed efficiently, but this simple solution cannot achieve $\mathcal{O}(N+M)$ time as its cost is alphabet-dependent.

Our Contribution. Our first result in this paper is an $\mathcal{O}(N+M)$-time algorithm for deciding whether the intersection of two GD strings of sizes N and M, respectively, over an integer alphabet is non-empty. This result is based on a combinatorial result of independent interest: although the intersection of two GD strings can be exponential in the total size of the two strings, it can be represented in only linear space. An automata model of computation can also be employed to obtain these results but we present here an efficient implementation in the standard word RAM model with word size $w = \Omega(\log(N+M))$ that works also for integer alphabets. We then apply our string comparison tool to compute palindromes in GD strings. We present an $\mathcal{O}(\min\{W, n^2\}N)$-time algorithm for computing all palindromes in $\hat{S}$. Furthermore, we show a non-trivial $\Omega(n^2|\Sigma|)$ lower bound under the Strong Exponential Time Hypothesis [21, 22] for computing all maximal palindromes. Note that there exists an infinite

family of GD strings over an integer alphabet of size $|\Sigma| = \Theta(N)$ on which our algorithm requires time $\mathcal{O}(n^2 N)$ thus matching the conditional lower bound. Finally, proof-of-concept experimental results are presented using real protein datasets; specifically, on applying our tools to find the location of palindromes in immunoglobulins genes of the human V regions.

2 Preliminaries

An *alphabet* Σ is a non-empty finite set of letters of size $\sigma = |\Sigma|$. A *string* X on an alphabet Σ is a sequence of elements of Σ. The set of all strings on an alphabet Σ, including the *empty string* ε of length 0, is denoted by Σ^*. For any string X, we denote by $X[i \dots j]$ the *substring* or *factor* of X that *starts* at position i and *ends* at position j. In particular, $X[0 \dots j]$ is the *prefix* of X that ends at position j, and $X[i \dots |X| - 1]$ is the *suffix* of X that starts at position i, where $|X|$ denotes the *length* of X. The *suffix tree* of X (*generalised suffix tree* for a set of strings) is a compact trie representing all suffixes of X. We denote the *reversal* of X by string X^R, i.e. $X^R = X[|X| - 1]X[|X| - 2] \dots X[0]$.

A string P is said to be a *palindrome* if and only if $P = P^R$. If factor $X[i \dots j]$, $0 \leq i \leq j \leq n - 1$, of string X of length n is a palindrome, then $\frac{i+j}{2}$ is the *center* of $X[i \dots j]$ in X and $\frac{j-i+1}{2}$ is the *radius* of $X[i \dots j]$. In other words, a palindrome is a string that reads the same forward and backward, i.e. a string P is a palindrome if $P = YaY^R$ where Y is a string, Y^R is the reversal of Y and a is either a single letter or the empty string. Moreover, $X[i \dots j]$ is called a *palindromic factor* of X. It is said to be a *maximal palindrome* if there is no other palindrome in X with center $\frac{i+j}{2}$ and larger radius. Hence X has exactly $2n - 1$ maximal palindromes. A maximal palindrome P of X can be encoded as a pair (c, r), where c is the center of P in X and r is the radius of P.

▶ **Definition 1.** A *generalised degenerate string (GD string)* $\hat{S} = \hat{S}[0]\hat{S}[1] \dots \hat{S}[n - 1]$ of *length* n over an alphabet Σ is a finite sequence of n degenerate letters. Every *degenerate letter* $\hat{S}[i]$ of *width* $k_i > 0$, denoted also by $w(\hat{S}[i])$, is a finite non-empty set of strings $\hat{S}[i][j] \in \Sigma^{k_i}$, with $0 \leq j < |\hat{S}[i]|$. For any GD string $\hat{S}$, we denote by $\hat{S}[i] \dots \hat{S}[j]$ the *GD substring* of $\hat{S}$ that starts at position i and ends at position j.

▶ **Definition 2.** The *total size* N and *total width* W, denoted also by $w(\hat{S})$, of a GD string $\hat{S}$ are respectively defined as $N = \sum_{i=0}^{n-1} |\hat{S}[i]| \times k_i$ and $W = \sum_{i=0}^{n-1} k_i$.

In this work, we generally consider GD strings over an *integer alphabet* of size $\sigma = N^{\mathcal{O}(1)}$.

▶ **Example 3.** The GD string $\hat{S}$ of Figure 1(c) has length $n = 6$, size $N = 28$, and $W = 12$.

▶ **Definition 4.** Given two degenerate letters $\hat{X}$ and $\hat{Y}$, their *Cartesian concatenation* is

$$\hat{X} \otimes \hat{Y} = \{xy \mid x \in \hat{X}, y \in \hat{Y}\}.$$

When $\hat{Y} = \emptyset$ (resp. $\hat{X} = \emptyset$) we set $\hat{X} \otimes \hat{Y} = \hat{X}$ (resp. $= \hat{Y}$). Notice that $\otimes$ is associative.

▶ **Definition 5.** Consider a GD string $\hat{S}$ of length n. The *language of* $\hat{S}$ is

$$L(\hat{S}) = \hat{S}[0] \otimes \hat{S}[1] \otimes \cdots \otimes \hat{S}[n - 1].$$

Given two GD strings $\hat{R}$ and $\hat{S}$ of equal total width the *intersection* of their languages is defined by $L(\hat{R}) \cap L(\hat{S})$.

▶ **Definition 6.** Let $\hat{X} = \{ x_i \in \Sigma^k \}$ and $\hat{Y} = \{ y_j \in \Sigma^h \}$ be two degenerate letters on alphabet Σ. Further let us assume without loss of generality that $\hat{Y}$ is the set that contains the shorter strings (i.e. $h \leq k$). We define the *chop of* $\hat{X}$ *and* $\hat{Y}$ and the *active suffixes of* $\hat{X}$ *and* $\hat{Y}$ as follows:

- $\text{chop}_{\hat{X},\hat{Y}} = \{\, y_j \in \hat{Y} \mid y_j \text{ matches a prefix of } x_i \in \hat{X} \,\}$
- $\text{active}_{\hat{X},\hat{Y}} = \{\, x_i[h \dots k-1] \mid x_i[0 \dots h-1] \in \text{chop}_{\hat{X},\hat{Y}} \,\}$

Let $w(\text{chop}_{\hat{X},\hat{Y}}) = \min\{w(\hat{X}), w(\hat{Y})\}$. When $\text{active}_{\hat{X},\hat{Y}} = \{\varepsilon\}$, we set $\text{active}_{\hat{X},\hat{Y}} = \emptyset$. We then have that $\text{active}_{\hat{X},\hat{Y}} = \emptyset$ either if $h = k$ or if there is no match between any of the strings in $\hat{Y}$ and the prefix of a string in $\hat{X}$; i.e. $\text{chop}_{\hat{X},\hat{Y}} = \emptyset$.

▶ **Example 7.** Consider the following degenerate letters $\hat{X}$ and $\hat{Y}$ where $w(\hat{Y}) < w(\hat{X})$. The underlined strings in letter $\hat{Y}$ are prefixes of strings in letter $\hat{X}$, hence they are in $\text{chop}_{\hat{X},\hat{Y}}$. The suffixes of such strings in $\hat{X}$ are the active suffixes in $\text{active}_{\hat{X},\hat{Y}}$.

$$\hat{X} = \begin{Bmatrix} \underline{\texttt{TCC}}\texttt{TA} \\ \texttt{ATCGA} \\ \underline{\texttt{TCC}}\texttt{AC} \\ \underline{\texttt{CAT}}\texttt{TA} \end{Bmatrix} \quad \hat{Y} = \begin{Bmatrix} \texttt{GCA} \\ \underline{\texttt{CAT}} \\ \underline{\texttt{TCC}} \end{Bmatrix} \quad \text{chop}_{\hat{X},\hat{Y}} = \begin{Bmatrix} \texttt{CAT} \\ \texttt{TCC} \end{Bmatrix} \quad \text{active}_{\hat{X},\hat{Y}} = \begin{Bmatrix} \texttt{TA} \\ \texttt{AC} \end{Bmatrix}$$

▶ **Definition 8.** Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively. $\hat{R}[0] \dots \hat{R}[i]$ is the *prefix* of $\hat{R}$ that ends at position i. It is called *proper* if $i \neq r-1$. We say that $\hat{R}[0] \dots \hat{R}[i]$ is *synchronized* with $\hat{S}[0] \dots \hat{S}[j]$ if $w(\hat{R}[0] \dots \hat{R}[i]) = w(\hat{S}[0] \dots \hat{S}[j])$. We call these the *shortest synchronized prefixes* of $\hat{R}$ and $\hat{S}$, respectively, when $\forall\ i' < i, j' < j$ $w(\hat{R}[0] \dots \hat{R}[i']) \neq w(\hat{S}[0] \dots \hat{S}[j'])$.

3 GD String Comparison

In this section, we consider the fundamental problem of GD string comparison. Let $\hat{R}$ and $\hat{S}$ be of total size N and M, respectively. We provide an $\mathcal{O}(N+M)$-time algorithm in the standard word RAM model with word size $w = \Omega(\log(N+M))$ that works also for integer alphabets.

Before presenting our efficient implementation, we observe that there is the following simple algorithm based on DFAs. Each degenerate letter of $\hat{R}$ and $\hat{S}$ can be represented by a trie, where its leaves are collapsed to a single one. For every two consecutive degenerate letters, the collapsed leaves of the former trie coincide with the root of the latter trie. An acyclic DFA is obtained in this way, as illustrated in Appendix A. We can perform the comparison of $\hat{R}$ and $\hat{S}$ by intersecting their corresponding DFAs using BFS on their product DFA. The trivial upper bound on the number of reachable states is $\mathcal{O}(NM)$, but this can be improved to $\mathcal{O}(N+M)$ by exploiting the structure of the two input DFAs. Each state in such a DFA has a unique level: the common length of paths from the initial state; and this structure is *inherited* by the product DFA. In other words, a level-i state in the product DFA corresponds to a pair of level-i states in the input DFAs. Observe that a level-i state in one DFA is uniquely represented by the label of the path from the root of its trie, and for a fixed DFA and level, these labels have uniform lengths. Considering the two states composing a reachable state in the product DFA, it is easy to see that the shorter label must be a suffix of the longer label. Hence, the state in the DFA with longer labels at level i uniquely determines the state in the DFA with shorter labels at level i. Consequently, the number of reachable level-i states in the product DFA is bounded by the number of level-i states in the input DFAs, and the size is $\mathcal{O}(N+M)$.

We observe that the cost of implementing the above ideas has an extra logarithmic factor due to state branching and, moreover, GD string comparisons require to build the DFAs each time. We show how to obtain $\mathcal{O}(N+M)$ time for integer alphabets, *without* creating DFAs. We show that, even if the size of $L(\hat{R}) \cap L(\hat{S})$ can be exponential in the total sizes of $\hat{R}$ and $\hat{S}$ (Fact 9), the problem of GD string comparison, i.e. deciding whether $L(\hat{R}) \cap L(\hat{S})$ is non-empty, can be solved in time linear with respect to the sum of the total sizes of the two GD strings (Theorem 17) and is thus of independent interest.

▶ **Fact 9.** *Given two GD strings $\hat{R}$ and $\hat{S}$, $L(\hat{S}) \cap L(\hat{R})$ can have size exponential in the total sizes of $\hat{R}$ and $\hat{S}$.*

We next show when it is possible to factorize $L(\hat{R}) \cap L(\hat{S})$ into a Cartesian concatenation.

▶ **Lemma 10.** *Consider two GD strings $\hat{S} = \hat{S}'\hat{S}''$ and $\hat{R} = \hat{R}'\hat{R}''$ such that $w(\hat{S}) = w(\hat{R})$. If $\hat{S}'$ is synchronized with $\hat{R}'$, then $L(\hat{R}) \cap L(\hat{S}) = (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{R}'') \cap L(\hat{S}''))$.*

Proof. It is clear that $L(\hat{S}) \cap L(\hat{R}) \supseteq (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{S}'') \cap L(\hat{R}''))$. Indeed, consider a string $x \in L(\hat{R}') \cap L(\hat{S}')$ and a string $y \in L(\hat{S}'') \cap L(\hat{R}'')$: then, by the definition of Cartesian concatenation, $xy \in L(\hat{R}') \otimes L(\hat{R}'') = L(\hat{R})$ and $xy \in L(\hat{S}') \otimes L(\hat{S}'') = L(\hat{S})$.
We now prove the opposite inclusion. Consider a string $z \in L(\hat{S}) \cap L(\hat{R})$. By definition, $z = x_0x1 \dots x_{r-1} = y_0y_1 \dots y_{s-1}$, with $x_i \in \hat{R}[i], y_j \in \hat{S}[j], \forall\ 0 \le i \le r-1, \forall\ 0 \le j \le s-1$. Let $\hat{R}' = \hat{R}[0] \dots \hat{R}[\bar{i}]$, $\hat{S}' = \hat{S}[0] \dots \hat{S}[\bar{j}]$. Assume by contradiction that $z \notin (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{S}'') \cap L(\hat{R}''))$: without loss of generality, $x_0 \dots x_{\bar{i}} \notin L(\hat{S}')$. Since $L(\hat{S}') \otimes L(\hat{S}'') = L(\hat{S})$, it follows that $z = x_0x_1 \dots x_{r-1} \notin L(\hat{S}) \implies z \notin L(\hat{S}) \cap L(\hat{R})$, that is a contradiction. ◀

By applying Lemma 10 wherever $\hat{R}$ and $\hat{S}$ have synchronized prefixes, we are then left with the problem of intersecting GD strings with no synchronized proper prefixes. We now define an alternative decomposition within such strings (see also Example 12).

▶ **Definition 11.** Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively, with no synchronized proper prefixes. We define

$$\text{c-chain}(\hat{R}, \hat{S}) = \max_q \{0 \le q \le r+s-2 \mid \text{chop}_q \neq \emptyset\},$$

where chop_i denotes the set $\text{chop}_{\hat{A}_i, \hat{B}_i}$, and $(\hat{A}_0, \hat{B}_0), (\hat{A}_1, \hat{B}_1), \dots, (\hat{A}_q, \hat{B}_q)$, $\text{pos}(\hat{A}_i), \text{pos}(\hat{B}_i)$ are recursively defined as follows:

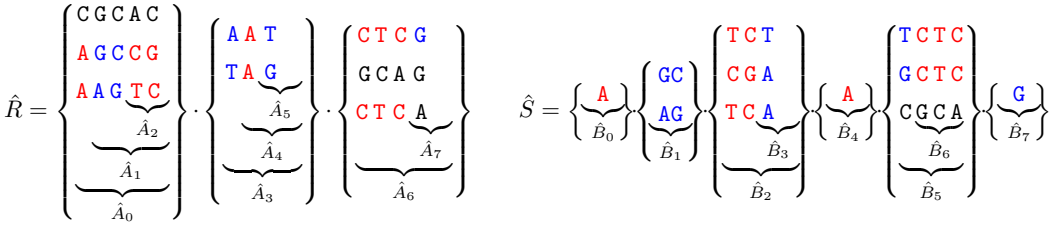
$\hat{A}_0 = \hat{R}[0], \hat{B}_0 = \hat{S}[0]$, and $\text{pos}(\hat{A}_0) = \text{pos}(\hat{B}_0) = 0$. For $0 < i \le r+s-2$, if $\text{chop}_{i-1} \neq \emptyset$,

$$\hat{A}_i = \begin{cases} \hat{R}[\text{pos}(\hat{A}_{i-1}) + 1] \text{ and } \text{pos}(\hat{A}_i) = \text{pos}(\hat{A}_{i-1}) + 1 & \text{if } w(\text{chop}_{i-1}) = w(\hat{A}_{i-1}) \\ \text{active}_{\hat{A}_{i-1}, \hat{B}_{i-1}} \text{ and } \text{pos}(\hat{A}_i) = \text{pos}(\hat{A}_{i-1}) & \text{otherwise} \end{cases}$$

$$\hat{B}_i = \begin{cases} \hat{S}[\text{pos}(\hat{B}_{i-1}) + 1] \text{ and } \text{pos}(\hat{B}_i) = \text{pos}(\hat{B}_{i-1}) + 1 & \text{if } w(\text{chop}_{i-1}) = w(\hat{B}_{i-1}) \\ \text{active}_{\hat{A}_{i-1}, \hat{B}_{i-1}} \text{ and } \text{pos}(\hat{B}_i) = \text{pos}(\hat{B}_{i-1}) & \text{otherwise} \end{cases}$$

The generation of pairs $(\hat{A}_i, \hat{B}_i)$ stops at $i = q$ either if $q = r+s-2$, or when $\text{chop}_{q+1} = \emptyset$, in which case $\hat{R}$ and $\hat{S}$ only match until $(\hat{A}_q, \hat{B}_q)$. Intuitively, $\hat{A}_i$ (respectively, $\hat{B}_i$) represents suffixes of the current position of $\hat{R}$ (respectively, of $\hat{S}$), while $\text{pos}(\hat{B}_i)$ (respectively, $\text{pos}(\hat{A}_i)$) tells *which* position of $\hat{R}$ (respectively, $\hat{S}$) we are chopping.

▶ **Example 12** (Definition 11)**.** Consider the following GD strings $\hat{R}$ and $\hat{S}$ with no synchronized proper prefixes: chop_0 is the first red set from the left, chop_1 is the first blue one, chop_2 is the second red one, etc. The c-chain$(\hat{R}, \hat{S})$ terminates when $q = 7$.



▶ **Definition 13.** Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively, with $w(\hat{R}) = w(\hat{S})$ and no synchronized proper prefixes. We define $G_{\hat{R},\hat{S}}$ as a directed acyclic graph with a structure of up to $r + s - 1$ levels, each node being a set of strings, as follows, where we assume without loss of generality that $w(\hat{R}[0]) > w(\hat{S}[0])$:

Level $k = 0$: consists of a single node:

$n_0 = \{x \in \hat{R}[0] \mid x = y_0 \dots y_{q_0}$ with $y_j \in \text{chop}_j \; \forall j : 0 \leq j \leq q_0\}$, where q_0 is the index of the rightmost chop containing suffixes of $\hat{R}[0]$.

Level $k > 0$: consists of $\ell = |\text{chop}_{q_{k-1}}|$ nodes. Assuming without loss of generality that level $k-1$ has been built with suffixes of $\hat{R}[\text{pos}(\hat{A}_{q_{k-1}})]$, level k contains suffixes of a position of $\hat{S}$. Let $c_0, \dots, c_{\ell-1}$ denote the elements of $\text{chop}_{q_{k-1}}$. Then, for $0 \leq i \leq \ell-1$, the i-th node of level k is:

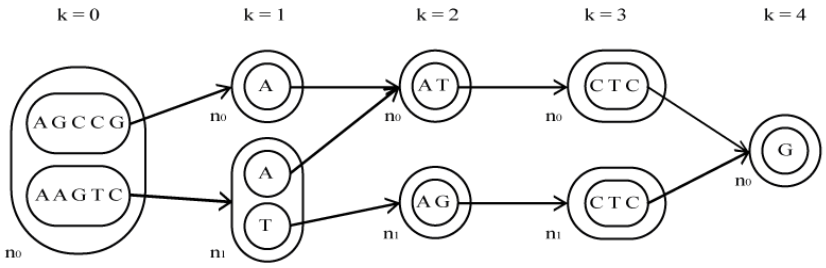
$n_i = \{y_{q_{k-1}+1} \dots y_{q_k} \mid c_i y_{q_{k-1}+1} \dots y_{q_k} \in \hat{B}_{q_{k-1}}$ with $y_j \in \text{chop}_j \; \forall j : q_{k-1}+1 \leq j \leq q_k\}$, where q_k is the index of the rightmost chop containing suffixes of $\hat{S}[\text{pos}(\hat{B}_{q_{k-1}})]$.

Every string in level $k-1$ whose suffix is c_i is the source of an edge having the whole node n_i as a sink.

We define $\text{paths}(G_{\hat{R},\hat{S}})$ as the set of strings spelled by a path in $G_{\hat{R},\hat{S}}$ that starts at n_0 and ends at the last level.

Note that the size of $G_{\hat{R},\hat{S}}$ is at most linear in the sum of the sizes of $\hat{R}$ and $\hat{S}$, as the nodes contain strings either in $\hat{R}$ or in $\hat{S}$ with no duplications, and each node has out-degree equal to the number of strings it contains.

▶ **Example 14** (Definition 13). $G_{\hat{R},\hat{S}}$ for the GD strings $\hat{R}, \hat{S}$ of Example 12 is:



$q_0 = 2$ and the strings in level 0 belong to $(\text{chop}_0 \otimes \text{chop}_1 \otimes \text{chop}_2) \cap \hat{R}[0]$. Level 1 contains suffixes of strings in $\hat{B}_2$ (and of strings in $\hat{B}_3$ as $\text{chop}_3 = \{\texttt{A}, \texttt{T}\}$ and indeed $q_1 = 3$), level 2 suffixes of strings in $\hat{A}_3$ (as $q_2 = 5$), level 3 suffixes of strings in $\hat{B}_5$ ($q_3 = 6$), level 4 suffixes of strings in $\hat{A}_6$ ($q_4 = 7$). The three paths from level 0 to level 4 correspond to the three strings in $L(\hat{R}) \cap L(\hat{S})$: `AGCCGAATCTCG, AAGTCAATCTCG, AAGTCTAGCTCG`.

Let $G^k_{\hat{R},\hat{S}}$ be $G_{\hat{R},\hat{S}}$ truncated at level k, and let $|G^k_{\hat{R},\hat{S}}|$ be the length of the strings it spells. Let $L_k(\hat{S})$ denote the set of prefixes of length $|G^k_{\hat{R},\hat{S}}|$ of $L(\hat{S})$.

▶ **Lemma 15.** *Let $\hat{R}, \hat{S}$ be two GD strings with $w(\hat{R}) = w(\hat{S}) = W$ and no synchronized proper prefixes. Then $L_k(\hat{S}) \cap L_k(\hat{R}) = \text{paths}(G^k_{\hat{R},\hat{S}})$ for all levels k of $G_{\hat{R},\hat{S}}$ such that $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$.*

Proof. Again, let us assume without loss of generality that $w(\hat{R}[0]) > w(\hat{S}[0])$. We prove the result by induction on k.

[Level $k = 0$] By construction, n_0 contains strings in $\hat{R}[0] \cap (\text{chop}_0 \otimes \cdots \otimes \text{chop}_{q_0})$, which have length $|G^0_{\hat{R},\hat{S}}|$, and are also in $\hat{S}[0]$, and hence belong to both $L_0(\hat{S})$ and $L_0(\hat{R})$.

[Level $k > 0$] By inductive hypothesis, we have that $L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R}) = \text{paths}(G^{k-1}_{\hat{R},\hat{S}})$: suppose that $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$, otherwise the graph ends at level $k-1$. We first show that $\text{paths}(G^k_{\hat{R},\hat{S}}) \subseteq L_k(\hat{S}) \cap L_k(\hat{R})$: by Definition 13, any $z \in \text{paths}(G^k_{\hat{R},\hat{S}})$ can be written as $z = z'z''$ with z' in $\text{paths}(G^{k-1}_{\hat{R},\hat{S}})$ and with z'' that belongs to some node at level k of $G^k_{\hat{R},\hat{S}}$ reached by an edge leaving a suffix of z'. By inductive hypothesis $z' \in L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R})$ and, again by Definition 13, $z'' \in \text{chop}_{q_{k-1}+1} \otimes \cdots \otimes \text{chop}_{q_k}$; since $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$ these chops are not empty, their concatenation contains the suffix of length $|G^k_{\hat{R},\hat{S}}| - |G^{k-1}_{\hat{R},\hat{S}}|$ of strings in both $L_k(\hat{R})$ and $L_k(\hat{S})$, and hence $z \in L_k(\hat{S}) \cap L_k(\hat{R})$.
We now show that $L_k(\hat{S}) \cap L_k(\hat{R}) \subseteq \text{paths}(G^k_{\hat{R},\hat{S}})$: consider string $u \in L_k(\hat{S}) \cap L_k(\hat{R})$ that can be written as $u = u'u''$ with u' the prefix of u having length $|G^{k-1}_{\hat{R},\hat{S}}|$ which then belongs to $L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R})$; then, by inductive hypothesis, $u' \in \text{paths}(G^{k-1}_{\hat{R},\hat{S}})$ and, since $u \in L_k(\hat{S}) \cap L_k(\hat{R})$, then there is an edge linking a suffix of u' at level $k-1$ with a node at level k of $G^k_{\hat{R},\hat{S}}$ containing a $|G^k_{\hat{R},\hat{S}}| - |G^{k-1}_{\hat{R},\hat{S}}|$ long suffix u'' of u, and hence $u \in \text{paths}(G^k_{\hat{R},\hat{S}})$. ◀

As a special case of Lemma 15, if $L(\hat{S}) \cap L(\hat{R}) \neq \emptyset$, then $G_{\hat{R},\hat{S}}$ is built up to the last level and the following holds.

▶ **Theorem 16.** *Let $\hat{R}, \hat{S}$ be two GD strings having lengths, respectively, r and s, with $w(\hat{R}) = w(\hat{S})$ and no synchronized proper prefixes. Then $G_{\hat{R},\hat{S}}$ has exactly $r + s - 1$ levels, and we have that $L(\hat{S}) \cap L(\hat{R}) = \text{paths}(G_{\hat{R},\hat{S}})$.*

$G_{\hat{R},\hat{S}}$ is thus a linear-sized representation of the possibly exponential-sized (Fact 9) set $L(\hat{S}) \cap L(\hat{R})$.

We now show an $\mathcal{O}(N + M)$-time algorithm for the standard word RAM model, denoted by GDSC, that decides whether $L(\hat{R})$ and $L(\hat{S})$ share at least one string (returns 1) or not (returns 0). GDSC starts with constructing the generalized suffix tree $T_{\hat{R},\hat{S}}$ of all the strings in $\hat{R}$ and $\hat{S}$. Then it scans $\hat{R}$ and $\hat{S}$ starting with $\hat{R}[0]$ and $\hat{S}[0]$ storing in $\text{chop}_{\hat{R},\hat{S}}$ the latest chop_i and in $\text{active}_{\hat{R},\hat{S}}$ the latest $\text{active}_{\hat{A}_i,\hat{B}_i}$ using $T_{\hat{R},\hat{S}}$. For an efficient implementation, suffixes in $\text{active}_{\hat{R},\hat{S}}$ are stored (e.g. for $\text{active}_{\hat{A}_0,\hat{B}_0}$ assuming that $w(\hat{R}[0]) > w(\hat{S}[0])$) as index positions of $\hat{R}[0]$ and the starting position of the suffix as $\text{active}_{\hat{R},\hat{S}}$.suff. The next comparison is made between the corresponding suffixes of $\hat{R}[0]$ of length $w(\hat{R}[0]) - \text{active}_{\hat{R},\hat{S}}$.suff and $\hat{S}[1]$, identifying first the minimum length of the two, and proceeding with the same process. The comparison of letters can be: (i) between $\hat{R}[i]$ and $\hat{S}[j]$; or (ii) between the corresponding strings of $\text{active}_{\hat{R},\hat{S}}$.index and $\hat{R}[i]$; or (iii) between the corresponding strings of $\text{active}_{\hat{R},\hat{S}}$.index and $\hat{S}[j]$. If the two GD strings have a synchronized proper prefix, this will result in $\text{active}_{\hat{R},\hat{S}} = \emptyset$ at positions i in $\hat{R}$ and j in $\hat{S}$. At this point, the comparison is restarted with the immediately following pair of degenerate letters.

▶ **Theorem 17.** *Algorithm* GDSC *is correct. Given two GD strings $\hat{R}$ and $\hat{S}$ of total sizes N and M, respectively, over an integer alphabet, algorithm* GDSC *requires $\mathcal{O}(N + M)$ time.*

Proof. The correctness follows directly from Lemma 10, Lemma 15, and Theorem 16.

Constructing the generalized suffix tree $T_{\hat{R},\hat{S}}$ can be done in time $\mathcal{O}(N + M)$ [12]. For the sets pair $(\hat{A}_i, \hat{B}_i)$ as in Definition 11, such that $w(\hat{A}_i) = k$ and $w(\hat{A}_i) \leq w(\hat{B}_i)$, we query $T_{\hat{R},\hat{S}}$ with the k-length prefixes of strings in $\hat{B}_i$. For integer alphabets, instead of spelling the strings from the root of $T_{\hat{R},\hat{S}}$, we locate the corresponding terminal nodes for $(\hat{A}_i, \hat{B}_i)$. It then suffices to find longest common prefixes between these suffixes to simulate the querying

process. Since all suffixes are lexicographically sorted during the construction of $T_{\hat{R},\hat{S}}$, we can also have the suffixes considered by pair $(\hat{A}_i, \hat{B}_i)$ lexicographically ranked with respect to $(\hat{A}_i, \hat{B}_i)$. Hence we do not perform the longest common prefix operation for all possible suffix pairs, but only for the lexicographically adjacent ones within this group. This can be done in $\mathcal{O}(1)$ time per pair after $\mathcal{O}(N+M)$-time pre-processing over $T_{\hat{R},\hat{S}}$ [7]. chop_i is thus populated with the k-length prefixes of strings in $\hat{B}_i$ found in $\hat{A}_i$. The set $\text{active}_{\hat{A}_i,\hat{B}_i}$ of active suffixes can be found by chopping the suffixes of the string in $\hat{B}_i$ from their prefixes successfully queried in $T_{\hat{R},\hat{S}}$. This requires time $\mathcal{O}(|\hat{A}_i| + |\hat{B}_i|)$ for processing $(\hat{A}_i, \hat{B}_i)$.

Let $\hat{R}$ and $\hat{S}$ be of length r and s, respectively. Assume that $\hat{R}$ and $\hat{S}$ have no synchronized proper prefixes. Then Theorem 16 ensures that the total number of comparisons cannot exceed $r+s-2$: this results in a time complexity of $\mathcal{O}(N+M+\sum_{i=0}^{r+s-2}(|\hat{A}_i|+|\hat{B}_i|)) = \mathcal{O}(N+M)$.

If $\hat{R}$ and $\hat{S}$ have synchronized proper prefixes, we perform the comparison up to the shortest synchronized prefixes (i.e. the set of active suffixes becomes empty) and then restart the procedure from the immediately following pair of degenerate letters. Clearly the total number of comparisons also in this case cannot be more than $r+s-2$. ◀

4 Computing Palindromes in GD Strings

Armed with the efficient GD string comparison tool, we shift our focus on our initial motivation, namely, computing palindromes in GD strings.

▶ **Definition 18.** A GD string $\hat{S}$ is a *GD palindrome* if there exists a string in $L(\hat{S})$ that is a palindrome.

A GD palindrome $\hat{S}[i]\dots\hat{S}[j]$ in $\hat{S}$, whose total width is $w(\hat{S}[i]\dots\hat{S}[j])$, can be encoded as a pair (c,r), where its *center* is $c = \frac{w(\hat{S}[0]\dots\hat{S}[i-1])+w(\hat{S}[0]\dots\hat{S}[j])-1}{2}$, when $i>0$, otherwise, $c = \frac{w(\hat{S}[0]\dots\hat{S}[j])-1}{2}$, when $i=0$; its *radius* is $r = \frac{w(\hat{S}[i]\dots\hat{S}[j])}{2}$. $\hat{S}[i]\dots\hat{S}[j]$ is called *maximal* if no other GD palindrome (c,r') exists in $\hat{S}$ with $r'>r$. Note that we only consider the GD palindromes $\hat{S}[i]\dots\hat{S}[j]$ that start with the first letter of some string $X \in \hat{S}[i]$ and end with the last letter of some string $Y \in \hat{S}[j]$, while the center can be anywhere: in between or inside degenerate letters. That is, in $\hat{S}$ there are $2\cdot w(\hat{S}) - 1 = 2W-1$ possible centers.

▶ **Example 19.** Consider the GD string $\hat{S}$ of Figure 1(c) where palindromes are underlined; one starts at $\hat{S}[0]$ and ends at $\hat{S}[2]$: it corresponds to $(c,r) = (2.5, 3)$. A second palindrome starts at $\hat{S}[4]$ and ends at $\hat{S}[5]$: it corresponds to $(c,r) = (9, 2.5)$.

In this section, we consider the following problem. Given a GD string $\hat{S}$ of length n, total size N, and total width W, find all GD strings $\hat{S}[i]\dots\hat{S}[j]$, with $0 \le i \le j \le n-1$, that are GD palindromes. We give two alternative algorithms: one finds all GD palindromes seeking them for all (i,j) pairs; and the other one finds them starting from all possible centers. The two algorithms have different time complexities: which one is faster depends on W, N, and n. In fact, they compute all GD palindromes, but report only the maximal ones.

We first describe algorithm MaxPalPairs. For all i,j positions within $\hat{S}$, in order to check whether $\hat{S}[i]\dots\hat{S}[j]$ is a GD palindrome, we apply the GDSC algorithm to $\hat{S}[i]\dots\hat{S}[j]$ and its reverse, denoted by $rev(\hat{S}[i]\dots\hat{S}[j])$; the reverse is defined by reversing the sequence of degenerate letters and also reversing the strings in every degenerate letter. GD palindromes are, finally, sorted per center, and the maximal GD palindromes are reported. Sorting the (i,j) pairs by their centers can be done in $\mathcal{O}(W)$ time using bucket sort, which is bounded by $\mathcal{O}(N)$ since $N \ge W$.

Since there are $\mathcal{O}(n^2)$ pairs (i, j), and since by Theorem 17 algorithm GDSC takes time proportional to the total size of $\hat{S}[i] \dots \hat{S}[j]$ to check whether $\hat{S}[i] \dots \hat{S}[j]$ is a GD palindrome, algorithm MaxPalPairs takes $\mathcal{O}(n^2 N)$ time in total. In algorithm MaxPalCenters, we consider all possible centers c of $\hat{S}$. In the case when c is in between two degenerate letters we simply try to extend to the left and to the right via applying GDSC. In the case when c is inside a degenerate letter we intuitively split the letter vertically into two letters and try to extend to the left and to the right via applying GDSC. At each extension step of this procedure we maintain two GD strings $\hat{L}$ (left of the center) and $\hat{R}$ (right of the center) such that they are of the same total width. We consider the reverse of $\hat{L}$ (similar to algorithm MaxPalPairs) for the comparison. In the case where c occurs inside a degenerate letter to make sure we do not identify palindromes which do not exist, for all j split strings of the degenerate letter, we check that $\hat{L}^R[0][j][0 \dots k-1] = \hat{R}[0][j][0 \dots k-1]$ where $\hat{L}^R = rev(\hat{L})$ and $k = \min(w(L^R[0]), w(\hat{R}[0]))$. If no matches are found, we move onto the next center. Otherwise, when a match is found, we update $rev(\hat{L})$ and $\hat{R}$ with the remainder of the split degenerate letter (if its length is greater than k), as well as the next degenerate letters. Algorithm GDSC is applied to compare $rev(\hat{L})$ and $\hat{R}$. After a positive comparison, we overwrite $\hat{L}$ and $\hat{R}$ by adding the degenerate letters of the current extension until $w(\hat{L}) = w(\hat{R})$ (or until the end of the string is reached). This process is repeated as long as GDSC returns a positive comparison, that is, until the maximal GD palindrome with center c is found. The radius reported is then the total sum of all values of $w(\hat{L})$. If GDSC returns a negative comparison at center c, we proceed with the next center, because we clearly cannot have a GD palindrome centered at c extended further if $rev(\hat{L}) \cap \hat{R}$ is empty.

By Theorem 17 and the fact that there are $2W - 1$ possible centers, we have that algorithm MaxPalCenters takes $\mathcal{O}(WN)$ time in total. We obtain the following result.

▶ **Theorem 20.** *Given a GD string of length n, total size N, and total width W, over an integer alphabet, all (maximal) GD palindromes can be computed in time $\mathcal{O}(\min\{W, n^2\}N)$.*

The problem that gained significant attention recently is the factorization of a string X of length n into a sequence of palindromes [3, 13, 30, 9, 5, 2]. We say that $X_1, X_2, \dots, X_\ell$ is a (maximal) palindromic factorization of string X, if every X_i is a (maximal) palindrome, $X = X_1 X_2 \dots X_\ell$, and ℓ is minimal. In biological applications we need to factorize a sequence into palindromes in order to identify *hairpins*, patterns that occur in single-stranded DNA or, more commonly, in RNA. Next, we define and solve the same problem for GD strings.

▶ **Definition 21.** A *(maximal) GD palindromic factorization* of a GD string $\hat{S}$ is a sequence $\hat{P}_1, \dots, \hat{P}_\ell$ of GD strings, such that: (i) every $\hat{P}_i$ is either a (maximal) GD palindrome or a degenerate letter of $\hat{S}$; (ii) $\hat{S} = \hat{P}_1 \dots \hat{P}_\ell$; (iii) ℓ is minimal.

After locating all (maximal) GD palindromes in $\hat{S}$ using Theorem 20, we are in a position to amend the algorithm of Alatabbi et al [3] to find a (maximal) GD palindromic factorization of $\hat{S}$. We define a directed graph $\mathcal{G}_{\hat{S}} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{i \mid 0 \leq i \leq n\}$ and $\mathcal{E} = \{(i, j+1) \mid \hat{S}[i \dots j]$ (maximal) GD palindrome of $\hat{S}\} \cup \{(i, i+1) | 0 \leq i < n\}$. Note that $\mathcal{V}$ contains a node n being the sink of edges representing (maximal) GD palindromes ending at $\hat{S}[n-1]$. For maximal GD palindromes, $\mathcal{E}$ contains no more than $3W$ edges, as the maximum number of maximal GD palindromes is $2W - 1$. For GD palindromes, $\mathcal{E}$ contains $\mathcal{O}(n^2)$ edges, as the maximum number of GD palindromes is $\mathcal{O}(n^2)$. A shortest path in $\mathcal{G}_{\hat{S}}$ from 0 to n gives a (maximal) GD palindromic factorization. For maximal GD palindromes, the size of $\mathcal{G}_{\hat{S}}$ is $\mathcal{O}(W)$, as $n \leq W$, and so finding this shortest path requires $\mathcal{O}(W)$ time using a standard algorithm. For GD palindromes, the size of $\mathcal{G}_{\hat{S}}$, and thus the time, is $\mathcal{O}(n^2)$.

▶ **Theorem 22.** *Given a GD string $\hat{S}$ of length n, total size N, and total width W, over an integer alphabet, a (maximal) GD palindromic factorization of $\hat{S}$ can be computed in time $\mathcal{O}(\min\{W, n^2\}N)$.*

5 A Conditional Lower Bound under SETH

In this section, we show a conditional lower bound for computing palindromes in degenerate strings. Let us first define the 2-Orthogonal Vectors problem. Given two sets $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$ of d-bit vectors, where $d = \omega(\log n)$, the *2-Orthogonal Vectors* problem asks the following question: is there any pair α_i, β_j of vectors that is orthogonal? Namely, is $\sum_{k=0}^{d-1} \alpha_i[k] \cdot \beta_j[k]$ equal to 0? For the moderate dimension of this problem, we follow [16], assuming $n^{2-\epsilon}d^{\mathcal{O}(1)} \leq n^2 d$. The following result is known.

▶ **Theorem 23** ([16, 21, 22, 33])**.** *The 2-Orthogonal Vectors problem cannot be solved in $\mathcal{O}(n^{2-\epsilon} \cdot d^{\mathcal{O}(1)})$ time, for any $\epsilon > 0$, unless the* Strong Exponential Time Hypothesis *fails.*

We next show that the 2-Orthogonal Vectors problem can be reduced to computing maximal palindromes in degenerate strings thus obtaining a similar conditional lower bound to the upper bound obtained in Theorem 20 for computing all GD palindromes.

▶ **Theorem 24.** *Given a degenerate string of length $4n$ over an alphabet of size $\sigma = \omega(\log n)$, all maximal GD palindromes cannot be computed in $\mathcal{O}(n^{2-\epsilon} \cdot \sigma^{\mathcal{O}(1)})$ time, for any $\epsilon > 0$, unless the* Strong Exponential Time Hypothesis *fails.*

Proof. Let $d = \sigma$ and consider the alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$. We say that two subsets of Σ *match* if they have a common element. Given a d-bit vector α, we define $\mu(\alpha)$ to be the following subset of Σ: $s \in \mu(\alpha)$ if and only if $\alpha[s] = 1$. Thus, two vectors α and β are orthogonal if and only if the sets $\mu(\alpha)$ and $\mu(\beta)$ are disjoint. In the string comparison setting, two degenerate letters $\mu(\alpha)$ and $\mu(\beta)$ *do not match* if and only if α and β are orthogonal. The reduction works as follows. Given $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$, we construct the following simple degenerate string of length $4n$ in time $\mathcal{O}(n\sigma)$:

$$S = \underbrace{\mu(\alpha_1)\mu(\beta_1)\mu(\alpha_2)\mu(\beta_2)\ldots\mu(\alpha_n)\mu(\beta_n)}\,\mu(\alpha_1)\mu(\beta_1)\mu(\alpha_2)\mu(\beta_2)\ldots\mu(\alpha_n)\mu(\beta_n).$$

$$\ldots \qquad\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}$$

Then the 2-Orthogonal Vectors problem for the sets A and B has a positive answer if and only if at any position of S, from 0 to $2n$, there *does not occur* a palindrome of length at least $2n$. All such occurrences can be easily verified from the respective palindrome centers in time $\mathcal{O}(n)$. In other words, if at any position of S there does not occur a palindrome of length at least $2n$, this is because we have a mismatch between a pair $\mu(\alpha_i), \mu(\beta_j)$ of letters, which implies that there exists a pair α_i, β_j of orthogonal vectors. Also, by the construction, all such pairs are to be (implicitly) compared, and thus, if there exists any pair that is orthogonal the corresponding mismatch will result in a palindrome of length less than $2n$. ◀

6 Experimental Results

We present here a proof-of-concept experiment but we anticipate that the algorithmic tools developed in this paper are applicable in a wide range of biological applications.

We first obtained the amino acid sequences of 5 immunoglobulins within the human V regions [15] and converted these into mRNA sequences [31]. The letters `X`, `S`, `T`, `Y`, `Z`, `R`

Table 1 Coordinates of (maximal) palindromes identified within hypervariable regions I and II.

	Hypervariable Region			
	I		II	
V	[34]	**This paper**	[34]	**This paper**
V_kII	18-27	11-36	119-130	118-131
	104-113	104-113	169-180	169-180
V_kIII	18-27	11-30	132-142	131-145
V_λII	63-74	62-81	140-152	140-152
V_λIII	51-74	50-75	132-143	131-144
V_λV	96-104	95-104	134-141	134-141

and `H` were replaced by degenerate letters according to IUPAC [23]. Each other letter, $c \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$, was treated as a single degenerate letter $\{c\}$. An average of 47% of the total number of positions within the 5 sequences consisted of one of the following: `X`, `S`, `T`, `Y`, `Z`, `R` and `H`. We then used algorithm MaxPalPairs to find all maximal palindromes in the 5 sequences. Table 1 shows the palindromes identified within hypervariable regions I and II. Our results are in accordance with Wuilmart et al [34] who presented a *statistical* (fundamentally different) method to identify the location of palindromes within regions of immunoglobulin genes. The ranges we report are greater than or equal to the ones of [34] due to the *maximality* criterion.

References

1 Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

2 Michał Adamczyk, Mai Alzamel, Panagiotis Charalampopoulos, Costas S. Iliopoulos, and Jakub Radoszewski. Palindromic decompositions with gaps and errors. In *CSR*, volume 10304 of *LNCS*, pages 48–61. Springer International Publishing, 2017.

3 Ali Alatabbi, Costas S. Iliopoulos, and M. Sohel Rahman. Maximal palindromic factorization. In *PSC*, pages 70–77, 2013.

4 Yannis Almirantis, Panagiotis Charalampopoulos, Jia Gao, Costas S. Iliopoulos, Manal Mohamed, Solon P. Pissis, and Dimitris Polychronopoulos. On avoided words, absent words, and their application to biological sequence analysis. *Algorithms for Molecular Biology*, 12(1):5, 2017.

5 Mai Alzamel, Jia Gao, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis. Efficient computation of palindromes in sequences with uncertainties. In *EANN*, volume 744 of *CCIS*, pages 620–629. Springer, 2017.

6 Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(1):163–173, 1995.

7 Michael A. Bender and Martín Farach-Colton. The LCA problem revisited. In *LATIN*, volume 1776 of *LNCS*, pages 88–94. Springer, 2000.

8 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Pattern matching on elastic-degenerate text with errors. In *SPIRE*, volume 10508 of *LNCS*, pages 74–90. Springer, 2017.

9 Kirill Borozdin, Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Palindromic Length in Linear Time. In *CPM*, volume 78 of *LIPIcs*, pages 23:1–23:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

10 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, pages 1–18, 2016.

11 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Covering problems for partial words and for indeterminate strings. *Theoretical Computer Science*, 698:25–39, 2017.

12 Martin Farach. Optimal suffix tree construction with large alphabets. In *FOCS*, pages 137–143. IEEE, 1997.

13 Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *Journal of Discrete Algorithms*, 28:41–48, 2014.

14 Martin C. Frith, Ulla Hansen, John L. Spouge, and Zhiping Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res.*, 32(1):189–200, 2004.

15 J. A. Gally and G. M. Edelman. The genetic control of immunoglobulin synthesis. *Annual Review of Genetics*, 6(1):1–46, 1972.

16 Jiawei Gao and Russell Impagliazzo. Orthogonal vectors is hard for first-order properties on sparse graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:53, 2016.

17 Roberto Grossi, Costas S. Iliopoulos, Chang Liu, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, Giovanna Rosone, Fatima Vayani, and Luca Versari. On-line pattern matching on a set of similar texts. In *CPM*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

18 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

19 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate texts. In *LATA*, volume 10168 of *LNCS*, pages 131–142. Springer International Publishing, 2017.

20 Costas S. Iliopoulos and Jakub Radoszewski. Truly Subquadratic-Time Extension Queries and Periodicity Detection in Strings with Uncertainties. In *CPM*, volume 54 of *LIPIcs*, pages 8:1–8:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

22 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

23 IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970.

24 Richard J. Lipton. *On The Intersection of Finite Automata*, pages 145–148. Springer US, Boston, MA, 2010.

25 Glenn Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, 1975.

26 Lee Ann McCue, William Thompson, Steven Carmack, Michael P. Ryan, Jun S. Liu, Victoria Derbyshire, and Charles E. Lawrence. Phylogenetic footprinting of transcription factor binding sites in proteobacterial genomes. *Nucleic Acids Res.*, 29(3):774–782, 2001.

27 Brejnev Muhizi Muhire, Michael Golden, Ben Murrell, Pierre Lefeuvre, Jean-Michel Lett, Alistair Gray, Art YF Poon, Nobubelo Kwanele Ngandu, Yves Semegni, Emil Pavlov Tanov, et al. Evidence of pervasive biologically functional secondary structures within the genomes of eukaryotic single-stranded DNA viruses. *Journal of virology*, 88(4):1972–1989, 2014.

28 Eugene W Myers. Approximate matching of network expressions with spacers. *Journal of Computational Biology*, 3(1):33–51, 1996.

29 Nadia Pisanti, Henry Soldano, Mathilde Carpentier, and Joël Pothier. A relational extension of the notion of motifs: Application to the common 3d protein substructures searching problem. *Journal of Computational Biology*, 16(12):1635–1660, 2009.

30 Mikhail Rubinchik and Arseny M. Shur. Eertree: An efficient data structure for processing palindromes in strings. In *IWOCA*, volume 9538 of *LNCS*, pages 321–333. Springer International Publishing, 2016.

31 Randall T. Schuh. Major patterns in vertebrate evolution. *Systematic Biology*, 27(2):172, 1978.

32 Henry Soldano, Alain Viari, and Marc Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognition Letters*, 16(3):233–246, 1995.

33 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci*, 348(2-3):357–365, 2005.

34 C. Wuilmart, J. Urbain, and D. Givol. On the location of palindromes in immunoglobulin genes. *Proceedings of the National Academy of Sciences of the United States of America*, 74(6):2526–2530, 1977.

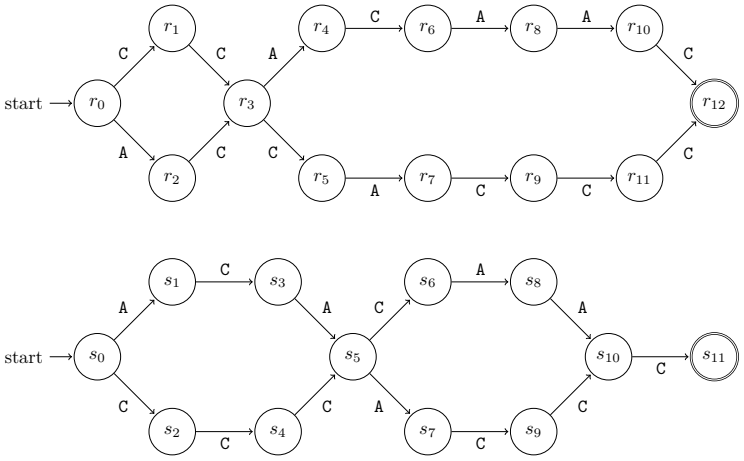
APPENDIX

A GD String Comparison Using Automata

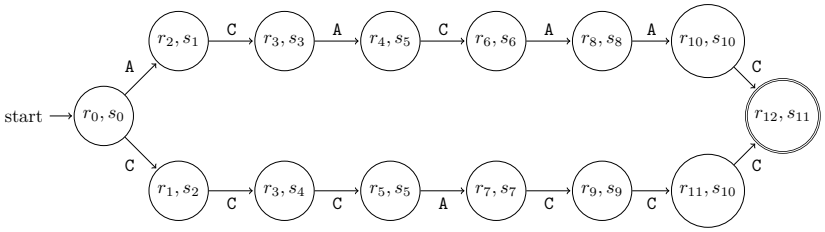
▶ **Example 25.** We illustrate here a simple automata-based approach. Say we want to compare the following two GD strings:

$$\hat{R} = \begin{Bmatrix} \texttt{AC} \\ \texttt{CC} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{ACAAC} \\ \texttt{CACCC} \end{Bmatrix} \qquad \hat{S} = \begin{Bmatrix} \texttt{ACA} \\ \texttt{CCC} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{ACC} \\ \texttt{CAA} \end{Bmatrix} \cdot \{\texttt{C}\}\,.$$

We construct the DFA for $\hat{R}$ and the DFA for $\hat{S}$.



Their product DFA gives their intersection: ACACAAC and CCCACCC.



We observe that computing the product DFA is alphabet-dependent, due to branching (transition function) on the same letter in the states of the two input DFAs.

A Multi-labeled Tree Edit Distance for Comparing "Clonal Trees" of Tumor Progression

Nikolai Karpov
Department of Computer Science, Indiana University, Bloomington, IN, USA
nkarpov@iu.edu

Salem Malikic
School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
smalikic@sfu.ca

Md. Khaledur Rahman
Department of Computer Science, Indiana University, Bloomington, IN, USA
morahma@iu.edu

S. Cenk Sahinalp
Department of Computer Science, Indiana University, Bloomington, IN, USA
cenksahi@iu.edu

Abstract

We introduce a new edit distance measure between a pair of "clonal trees", each representing the progression and mutational heterogeneity of a tumor sample, constructed by the use of single cell or bulk high throughput sequencing data. In a clonal tree, each vertex represents a specific tumor clone, and is labeled with one or more mutations in a way that each mutation is assigned to the oldest clone that harbors it. Given two clonal trees, our multi-labeled tree edit distance (MLTED) measure is defined as the minimum number of mutation/label deletions, (empty) leaf deletions, and vertex (clonal) expansions, applied in any order, to convert each of the two trees to the maximal common tree. We show that the MLTED measure can be computed efficiently in polynomial time and it captures the similarity between trees of different clonal granularity well. We have implemented our algorithm to compute MLTED exactly and applied it to a variety of data sets successfully. The source code of our method can be found in: `https://github.com/khaled-rahman/leafDelTED`.

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

Keywords and phrases Intra-tumor heterogeneity, tumor evolution, multi-labeled tree, tree edit distance, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.22

Funding NK is supported by NSF CCF-1525024 and IIS-1633215. SM is supported by a Vanier Canada Graduate Scholarship.

1 Introduction

According to the *clonal theory of cancer evolution* [26], cancer originates from a single cell which had acquired a set of mutations that provide it proliferative advantage compared to the neighboring healthy cells. As tumor grows, cancer cells acquire new mutations and some of them might accumulate set of mutations conferring further selective advantage or disadvantage compared to the other cells. This continues over a period of time and at the time of the clinical diagnosis, tumors are usually heterogeneous consisting of multiple cellular

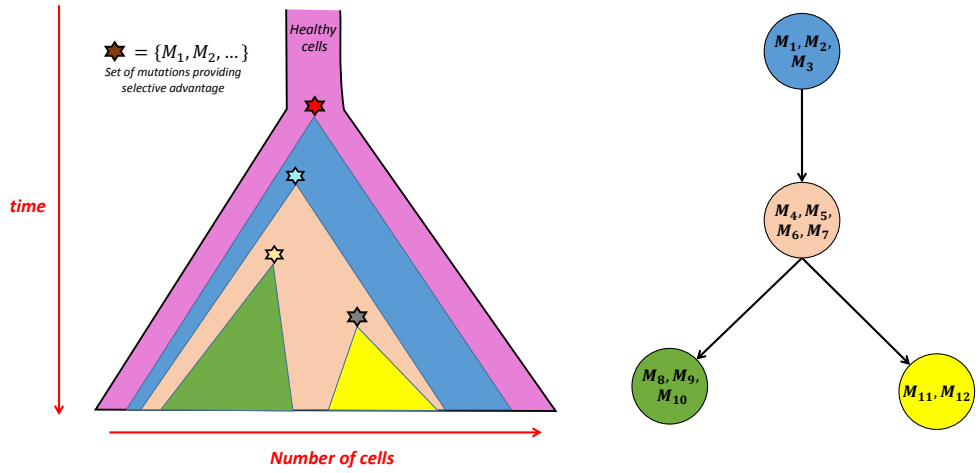


Figure 1 Graphical overview of tumor initiation and growth (left) and the corresponding clonal tree of tumor evolution (right). Sets of mutations providing proliferative advantage and driving the emergence of new clones are denoted as stars in the left and as sets of corresponding mutations in the right panel (e.g. red star from the left panel represents the set of mutations $\{M_1, M_2, M_3\}$.) Node corresponding to the healthy cells is omitted as it would be non-informative.

populations, harboring distinct sets of mutations, leading to different phenotypes. Each such cellular population is considered to be a clone. The whole process of tumor initiation and growth is illustrated in Figure 1 (left panel).

One of the most widely used ways of depicting mutational heterogeneity and tumor progression over time is by the use of a *clonal tree of tumor evolution*. Here, each individual vertex represents a distinct clone and each mutation (i.e. its label) is placed as part of the label of clone where it occurs for the first time in evolutionary history. In this work we focus on trees built by the use of single nucleotide variants (SNVs), which represent the most widely used type of mutations in reconstructing trees of tumor evolution [12]. We also assume that each SNV occurs exactly once during the course of tumor evolution and never gets lost (infinite sites assumption, usually abbreviated as ISA). Some recently introduced methods (e.g. SiFit [10]) allow for the violations of ISA and, in such cases, we expect that labels corresponding to mutations vioalating ISA are removed from the trees prior to distance calculation. In order to simplify our figures, in each figure in this work we omit the node representing population of healthy cells. Namely, such node would be non-informative as it would always be label-free (since healthy cells are assumed to contain none of the mutations relevant to cancer progression) and attached as the parent of root node in each of the figures presented in this work. See Figure 1 for an illustration of tumor growth (left panel) and the corresponding clonal tree of tumor evolution (right panel). Note that the children of a vertex in a clonal tree are unordered.

A popular alternative to the clonal tree is the *mutation tree*, a special case of the clonal tree, where along each edge there is exactly one mutation [21, 14] - thus a mutation tree is a clonal tree with the highest possible granularity. As can be expected, any clonal tree can be easily converted to the mutation tree as follows. Consider an arbitrary edge (u, v) and assume WLOG that a set of all mutations assigned to it is $\{M_1, M_2, \ldots, M_k\}$. Now replace each edge (u, v) by a path with vertices $\{w_0 = u, w_1, w_2, \ldots, w_{k-1}, w_k = v\}$ and

edges $\{(w_0, w_1), (w_1, w_2), \ldots, (w_{k-1}, w_k)\}$, such that exactly one mutation, WLOG M_i, is assigned to the edge (w_{i-1}, w_i) for each $i \in \{1, 2, \ldots, k\}$. Note that from a given clonal tree which is not mutation tree (i.e. contains at least one node with two or more labels), multiple different mutation trees can be obtained. There are additional tree representations [14] for tumor evolution but in this work we focus on clonal trees only.

Similarity/distance measures between tree representations of tumor evolution

In the past few years, we have witnessed rapid developments in computational methods for inferring trees of tumor evolution from both bulk and single cell high throughput sequencing (HTS) data [21, 14, 9, 18, 11, 6, 15, 27, 16, 25, 8, 5].

In order to assess the accuracy of the proposed method, many of these studies use simulated HTS data extracted from synthetic tumor compositions. The inferred tree is then compared against the (synthetic) ground truth. (We will call the *ground truth tree* the *true tree*.) Other studies, such as the Pan Cancer Analysis of Whole Genomes Project (PCAWG) compare trees inferred by participating methods on real tumor samples to reach a consensus tree. In order to compare clonal trees with varying granularity (granularity can be measured in terms of the average number of mutations assigned to a clone) the measure(s) used should be versatile enough to discriminate real topological differences between trees from those differences due to the type and coverage of the HTS data used by a method; e.g. such a distance measure should be equal to 0 between any clonal tree and its corresponding mutation tree (obtained using the procedure described above).

Unfortunately, comparing trees of tumor evolution has turned out to be a very challenging problem and available measures fail to capture the differences/similarities between inferred or true trees. In fact many of existing measures only aim to compare the relative placement of pairs of mutations across two trees, e.g. whether the two mutations maintain an ancestor-descendant relationship in both trees [27, 16]. Such measures can not capture topological differences between distinct trees, e.g. a simple topology with two vertices, where all but one of the mutations is assigned to the non-root vertex, v.s. a star topology where each vertex is assigned a single mutation. As a result it is of high importance to have measures of tree similarity that not only consider the relative placement of mutations but also the topological structure of the trees.

The standard measure to compare combinatorial objects - such as strings, especially in bioinformatics, is the edit distance. This measure has numerous applications and a large number of variants, not only for strings but also for labeled trees, have been considered in the past. The classical Levenstein edit distance between two strings is defined as the minimum number of single symbol insertions, deletions and replacements to convert one of the strings to the other and has a well established dynamic programming algorithm to compute it (e.g. [35]). The running time of this algorithm is proportional to the product of the lengths of the two input strings and the existence of a sub-quadratic algorithm is unlikely [1]. In general, the complexity of computing an edit distance strictly depends on the set of allowed edit operations. E.g. if we consider a variant of the problem where only single character mismatches and block reversals are allowed, then the running time reduces to $O(n \log^2 n)$ [29] - here n is the total length of the strings; on the other hand, the variant where only mismatches, block deletion and move operations are allowed is NP-hard [32].

Edit distance measures for rooted trees have typically been defined for those with ordered vertices, each with a single label; here the goal is to transform one tree to the other by the use of vertex insertions, vertex deletions and vertex label replacements [31]. Based on the tree edit distance, a notion of tree alignment has also been introduced, both for vertex ordered as

well as unordered trees [13]. For many of the vertex ordered cases, there are polynomial time algorithms that can solve the distance/alignment problem [33, 31, 22, 4, 38, 37, 30, 13, 20, 3], whereas for several unordered cases, the problems are NP-hard [24, 34] or MAX SNP-hard [39, 13].

Unfortunately for many of variants of the problem, the time complexity is still unknown [2]. As importantly none of the existing algorithms deal with trees where vertices may have more than a single (mutational) label - which is at the heart of clonal tree comparison problem.

In this paper we introduce for the first time a tree edit distance measure to compare trees where vertices may have one or more (mutational) labels. The multi-labeled tree edit distance (MLTED) measure we introduce in this paper successfully captures the differences between clonal trees: for example it satisfies a key condition that two clonal trees from which it is possible to produce two identical mutation trees have a distance 0. Even though MLTED is defined for (the more general) vertex unordered trees, we show that there is an algorithm to exactly compute it in polynomial time. We have implemented this algorithm and applied it to a number of synthetic and real data sets to compare trees inferred by some of the available tumor history reconstruction methods with success.

2 Definitions

A rooted tree $T = (V, E)$ is a connected, acyclic, undirected graph with set of vertices V (also denoted as $V(T)$) and edges E (also denoted as $E(T)$), with a particular vertex, r identified as the root. For each non-root vertex v, any vertex u that lies on the simple path between v and the root is considered to be its ancestor; in particular, the node $u = p(v)$ on this path which has an edge to v is considered to be its parent. The depth of vertex v denoted $d(v)$, is thus defined as the number of its ancestors. The lowest common ancestor of any pair of vertices u and v, denoted $\mathrm{lca}(u, v)$, is defined as a common ancestor of both u and v whose depth is maximum possible. The structure of a tree induces partial order $\preceq$ on its vertices: $u \preceq v$ denotes that u is an ancestor of v.

In this work, we consider multi-labeled trees in which each vertex v has a subset L_v of labels from a universe $\mathbb{L}$. Additionally, each label is unique to a vertex, i.e. $L_u \cap L_v = \emptyset$ for each pair of distinct vertices u and v. We denote the set of all labels assigned to the vertices of T as $L(T)$. In other words, $L(T) = \bigcup_{v \in V(T)} L_v$.

Consider the following types of edit operations on multi-labeled tree:

- *deleting a label* where one of the labels is removed from some set L_v,
- *deleting a vertex* where a vertex is removed from the tree. This operation is allowed to be performed only for unlabeled leaves, i.e. vertices with no labels and no children,
- *expanding a vertex* where vertex v is replaced by two vertices v_1 and v_2 such that all children of v after this operation are children of v_2, and the parent of v is the parent of v_1, and v_1 is the parent of v_2. Each of the labels from L_v is assigned to exactly one of the L_{v_1} and L_{v_2}.

We define the edit distance between two multi-labeled trees as the minimal number of label deletions, such that together with an arbitrary number of vertex deletions and vertex expansions (which do not contribute to the distance), they transform both trees to a common third tree (which will be defined as the *maximal common tree* of the two input trees). As can be observed, the distance between two trees is upper bounded by the total number of labels in the two trees.[1]

[1] Note that typically edit distance measures are based on symmetric edit operations, in a way that each

3 Set Alignment Problem

Our algorithm for computing the distance between trees is based on finding a *maximal common tree*, as will be defined below. We first show connection between this notion and our edit distance measure and then demonstrate how this notion works for paths, which form the base case. We later extend this definition to the general case and provide an efficient algorithm for computing the maximal common tree.

A *Common tree* of arbitrary multi-labeled trees T_1 and T_2 is any multi-labeled tree which can be obtained from each of T_1 and T_2 by the use of edit operations defined above. A *maximal common tree* of T_1 and T_2 is a common tree of T_1 and T_2 having the largest number of labels among all common trees of T_1 and T_2. Our MLTED measure between T_1 and T_2 is, by definition, equal to the difference between the total number of labels in T_1 and T_2 and twice the number of labels in their maximal common tree. In other words, tree edit distance is defined as the total number of labels required to be removed from the two trees in order to transform them to their maximal common tree.

If the two trees are simple paths, then any common tree between them is also a path. Let the ordered sequence of vertices of the first tree/path be $v_1, v_2, \ldots, v_n$ with respective label sets $S_1, S_2, \ldots, S_n$, and the ordered sequence of vertices of the second tree/path be $w_1, w_2, \ldots, w_m$ with respective label sets $P_1, P_2, \ldots, P_m$. (Assume that S_i, P_j are subsets of $\mathbb{L}$ and that any label $u \in \mathbb{L}$ occurs exactly in one of $S_1, S_2, \ldots, S_n$ and exactly in one of $P_1, P_2, \ldots, P_m$.) Let $f : \mathbb{L} \to \{1, 2, \ldots, n\}$ and $g : \mathbb{L} \to \{1, 2, \ldots, m\}$ be the functions that map labels to vertex indices, respectively in the first and the second tree such that $v_{f(a)}$ denotes the vertex of label a in the first tree and $w_{g(a)}$ denotes the vertex of the label a in the second tree.

It is easy to see that computing a maximal common tree in this special case is equivalent to the following generalized version of the string edit distance problem for a pair of ordered sets.

Set Alignment Problem
Instance: Two ordered set of labels: $(S_1, S_2, \ldots, S_i)$ and $(P_1, P_2, \ldots, P_j)$ where $1 \leq i \leq n$ and $1 \leq j \leq m$.
Task: Find set $A(i,j) \subseteq (\bigcup_{p=1}^{i} S_p) \bigcap (\bigcup_{q=1}^{j} P_q)$ of maximal size such that, for each pair (a, b) of labels from $A(i,j)$, the following holds: $f(a) \leq f(b) \iff g(a) \leq g(b)$.

The following lemma offers an efficient algorithm for solving the Set Alignment Problem. Our approach for computing edit distance between two arbitrary trees (presented in Section 4) uses this algorithm as a subroutine.

▶ **Lemma 1.** *Let* $\mathrm{D}(i,j)$ *be the size of the set which is answer of the* Set Alignment Problem *for the instance where input sequences are* $(S_1, \ldots, S_i)$ *and* $(P_1, \ldots, P_j)$ *(i.e. according to the notation from the above* $D(i,j) = |A(i,j)|$*). Then the following hold:*

- $\mathrm{D}(i,0) = \mathrm{D}(0,j) = 0$*, for all non-negative integers* i *and* j*.*
- $\mathrm{D}(i,j) = \max(\mathrm{D}(i,j-1), \mathrm{D}(i-1,j)) + |S_i \cap P_j|$*, for all positive integers* i *and* j*.*

operation is complemented by a reverse operation (e.g. deleting a label is the reverse of inserting the same label). In such cases, the edit distance is defined as the minimum number of operations required to transform one combinatorial object into another. Although it is possible to define our edit distance measure similarly (with label insertions complementing label deletions), we chose to present our distance by specifying deletions only for keeping the description compact.

Proof. The first equation easily follows from the fact that $A(i,0) \subseteq \emptyset$ and $A(0,j) \subseteq \emptyset$.

For the second equation, we first prove that $\mathrm{D}(i,j) \geq \max(\mathrm{D}(i,j-1), \mathrm{D}(i-1,j)) + |S_i \cap P_j|$. In order to prove this, observe that each of $A(i,j-1) \cup (S_i \cap P_j)$ and $A(i-1,j) \cup (S_i \cap P_j)$ represent a valid candidate solution for the instance of Set Alignment Problem with the input sequences $(S_1, \ldots, S_i)$ and $(P_1, \ldots, P_j)$. Namely, in the case of set $A(i,j-1) \cup (S_i \cap P_j)$ (analogous applies to the set $A(i-1,j) \cup (S_i \cap P_j)$), if we consider two arbitrary labels a and b of this set, then:

- If $a \in A(i,j-1)$ and $b \in A(i,j-1)$ then $f(a) \leq f(b) \iff g(a) \leq g(b)$ holds by the definition of $A(i,j-1)$.
- If $a \in A(i,j-1)$ and $b \in S_i \cap P_j$ then $f(a) \leq i$ and $g(a) \leq j-1$. On the other hand, $f(b) = i$ and $g(b) = j$ hence $f(a) \leq f(b) \iff g(a) \leq g(b)$ is obviously satisfied.
- Case where $a \in S_i \cap P_j$ and $b \in A(i,j-1)$ is analogous to the previous case.
- Case where both a and b are from $S_i \cap P_j$ is trivial since in this case $f(a) = f(b) = i$ and $g(a) = g(b) = j$ implying that $f(a) \leq f(b) \iff g(a) \leq g(b)$ holds in this case as well.

Now it suffices to prove that $\mathrm{D}(i,j) \leq \max(\mathrm{D}(i,j-1), \mathrm{D}(i-1,j)) + |S_i \cap P_j|$. In order to prove this, consider the partition of $A(i,j)$ into $A(i,j) \setminus (S_i \cap P_j)$ and $S_i \cap P_j$. We claim that at most one of the sets S_i and P_j has non-empty intersection with the set $A(i,j) \setminus (S_i \cap P_j)$. To prove this, assume on contrary that there exists $a \in S_i \cap (A(i,j) \setminus (S_i \cap P_j))$ and $b \in P_j \cap (A(i,j) \setminus (S_i \cap P_j))$. Since $a \in S_i$ we have $f(a) = i$. For b we have that $b \in A(i,j)$ and $b \notin S_i$ implying that $f(b) \leq i-1$. Similarly, $g(a) \leq j-1$ and $g(b) = j$. By the above assumption, both a and b belong to $A(i,j)$ but obviously they violate constraint $f(a) \leq f(b) \iff g(a) \leq g(b)$ which is, by definition of $A(i,j)$ satisfied for all of its labels. This contradiction directly implies our latest claim. To finalize the proof of inequality $\mathrm{D}(i,j) \leq \max(\mathrm{D}(i,j-1), \mathrm{D}(i-1,j)) + |S_i \cap P_j|$ assume WLOG that the intersection of S_i and $A(i,j) \setminus (S_i \cap P_j)$ is the empty set. This implies that $A(i,j)$ does not contain any label from $S_i \setminus (S_i \cap P_j)$. Therefore $D(i,j) \leq D(i-1,j) + |S_i \cap P_j| \leq \max(\mathrm{D}(i,j-1), \mathrm{D}(i-1,j)) + |S_i \cap P_j|$ which completes our proof. ◀

Lemma 1 provides a dynamic programming formulation for calculating distance $D(n,m)$ between trees T_1 and T_2.

▶ **Observation 2.** *Total time and total space required for calculating number of labels in each of the sets $S_i \cap P_j$, where $i \in [n]$ and $j \in [m]$ are both $O(\sum_{i=1}^{n} |S_i| + \sum_{j=1}^{m} |P_j| + nm)$.*

Proof. For each label from $u \in L$ we can store two indices $f(u)$ and $g(u)$. This can be implemented in the above time and space by using a hash table. If we know these indices, we can fill the table I_{ij}, where $I_{ij} = |S_i \cap P_j|$, by iterating through elements of $\mathbb{L}$ and increasing the value of $I_{f(x)g(x)}$ by one for each $x \in \mathbb{L}$. ◀

▶ **Lemma 3.** *The Set Alignment Problem is solvable in $O(\sum_{i=1}^{n} |S_i| + \sum_{j=1}^{m} |P_j| + nm)$ time and space.*

Proof. Follows straightforwardly from Lemma 1 and Observation 2. ◀

4 Computing a maximal common tree in the general case

We now describe an efficient algorithm for computing a maximal common tree. Note that in the remainder of the paper we call all vertices in a tree with exactly one child as *non-crucial* vertices and all other vertices, i.e. leaves, and vertices with two or more children, as *crucial*

vertices. Now consider the sequence of edit operations applied to a tree T_1 in the process to reaching a common tree T with another tree T_2.

▶ **Observation 4.** *Each edit operation applied to any vertex deletes at most one crucial vertex and creates at most one (new) crucial vertex; no edit operation can increase the number of crucial vertices.*

Proof. The proof follows through a simple case analysis on the edit operations we allow.

- The edit operation of deleting a label does not change the topology of the tree or the set of crucial vertices in the tree.
- The edit operation of deleting a leaf u does change the topology of a tree, but with respect to the set of crucial vertices, the only update is that u is lost, and, (i) provided that u was the only child of $p(u)$, $p(u)$ becomes crucial, or (ii) provided that u was one of the two children of $p(u)$, $p(u)$ becomes non-crucial, or (iii) provided that u was one of more than two children of $p(u)$, $p(u)$ stays crucial. All other vertices remain unaltered.
- Finally, the edit operation of expanding, i.e., splitting a vertex v into v_1 and v_2 does change the topology of the tree but it does not create a new crucial vertex if v is non-crucial; however, if a vertex v is crucial, then v_2 becomes crucial after the edit operation, but v_1 stays non-crucial. ◀

The observation above indicates that an edit operation applied to a crucial vertex u may create a new crucial vertex v. In that case, we say that the crucial vertex u in T_1 *corresponds to* a crucial vertex v in T_1' (if latter was created). In case of an expansion of vertex u in T_1 to two vertices u_1 and u_2, we say that u corresponds to u_2 in T_1'. In case of a deletion of a leaf u, if $p(u)$ which was originally non-crucial, became crucial, then we say that u in T_1 corresponds to $p(u)$ in T_1'. For any vertex v which remains unedited and crucial in T_1', we say that v in tree T_1 corresponds to v in the tree T_1'.

Finally, we say that v in T_1 corresponds to v' in T if for the sequence of trees $T_1 = T_1^0, T_1^1, \ldots, T_1^l = T$ (where T_1^{i+1} is obtained from T_1^i by an edit operation) there exists the sequence of vertices $v = v^0, v^1, \ldots, v_l = v$ (where $v^l \in V(T_1^l)$) such that v^i corresponds to v_{i+1} for all i. We extend the notion of correspondence to T_2 in a similar manner.

Given trees T_1 and T_2, their common tree T and the vertices in T_1 and T_2 that correspond to every crucial vertex in T, it is straightforward to establish the edit operations to transform T_1 and T_2 to T. The algorithm to compute T makes use of this observation.

▶ **Observation 5.** *Given two sets of crucial vertices $u_1, \ldots, u_l$ and $v_1, \ldots, v_l$ in T_1 and T_2 respectively such that u_i and v_i correspond to same crucial vertex in the common tree T for each i, we can reconstruct a common tree T' such that the number of labels in T' is at least that in T.*

Proof. Here we describe the procedure of reconstructing the tree T' in two steps.

In the first step we delete each label which cannot belong to T in a trivial manner: let S_1 (S_2) be the set of vertices which do not lie on a path from the root of T_1 (T_2) to some u_i (v_i). Then we delete all vertices from S_1 (and S_2) together with their labels. Note that no label which is present in tree T will be deleted: if a vertex v does not belong to a path from the root to some crucial vertex in T, then any label from L_v cannot be present in T. However, if any label in T that is in L_v for some vertex v which lies on a path from the root to a leaf w (which is necessarily crucial) then there must exist a pair of vertices u_i, v_i which correspond to the leaf w.

Thus, starting from the leaf level, we can delete all vertices which do not belong to a path from the root to any u_i (and v_i). It is easy to see that this first step transforms T_1 and T_2 into isomorphic trees - the isomorphism ϕ on $r_1, u_1, \ldots, u_l$ which transforms T_1 into T_2 is $\phi(r_1) = r_2, \phi(u_1) = v_1, \ldots, \phi(u_l) = v_l$ (here r_i is the root of T_i).

Let T_1' and T_2' denote the trees respectively produced from T_1 and T_2 after applying the first step. Notice that, T_1' and T_2' are also topologically isomorphic to T and T'.

In the second step, for each pair of vertices v_i and u_i we consider the pair of "maximal" paths from v_i and u_i to the associated root, which do not contain other vertices from $v_1, \ldots, v_l$ and $u_1, \ldots, u_l$. For this pair of paths we apply a sequence of edit operations that expand vertices and delete labels, such that the resulting paths will be identical with the maximum possible number of labels.

T' is the tree produced as a result of the second step. Note that on any pair of paths from the vertex pair u_i and v_i to the respective root, the set of labels observed will be identical. This implies that T' is a common tree with number of labels necessarily lower bounded by that of T. ◀

The above observation implies that we can reduce the problem of computing a maximal common tree between two multi-labeled trees to the problem of finding an *optimal pair of sequences* of vertices $u_1, \ldots, u_l$ and $v_1, \ldots, v_l$ corresponding to the maximal common tree.

Our general algorithm for computing the distance between two multi-labeled trees requires constant time access to the solutions to many instances of the Set Alignment Problem, which we compute in a preprocessing step.

Solving Set Alignment Problem for all pairs of sequences $u_1, \ldots, u_l$ and $v_1, \ldots, v_l$ is impractical. Fortunately, special conditions with respect to the structure of these sequences help us develop an efficient algorithm for finding an optimal pair of sequences as explained below.

The algorithm for computing an optimal pair of sequences will need the solutions to Set Alignment Problem for all possible downward paths; we call this auxiliary problem Pairwise Alignments on a Tree.

Given a pair of vertices u, v such that $u \preceq v$, let the following sequence of sets of vertex labels be denoted as $\mathrm{P}(u, v) = (L_{w_1}, \ldots, L_{w_k})$ where $w_1(= u), w_2, \ldots, w_k(= v)$ is called the downward path between u and v. Then we can define Pairwise Alignments on a Tree problem formally as follows.

Pairwise Alignments on a Tree
Instance: Two rooted unordered multi-labeled trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, T_2)$ with associated sets of labels for each vertex.
Task: For each 4-tuple (a, b, c, d) such that $a, b \in V_1$, $c, d \in V_2$, $a \preceq b$ and $c \preceq d$, compute and store the answer for Set Alignment Problem on $\mathrm{P}(a, b), \mathrm{P}(c, d)$.

In the next lemma, we introduce equations for computing Pairwise Alignments on a Tree which forms the basis of our dynamic programming algorithm.

▶ **Lemma 6.** *Given $a, b \in V(T_1)$; $c, d \in V(T_2)$; $a \preceq b$; $c \preceq d$, let $\mathrm{D}(a, c, b, d)$ be the solution for the instance $\mathrm{P}(a, b)$, $\mathrm{P}(c, d)$ of* Set Alignment Problem. *Then*

1. *If $a = b$ and $c = d$ then $\mathrm{D}(a, c, b, d) = |L_b \cap L_d|$.*
2. *If $a = b$ and $c \neq d$ then $\mathrm{D}(a, c, b, d) = \mathrm{D}(a, c, b, p(d)) + |L_b \cap L_d|$.*
3. *If $a \neq b$ and $c = d$ then $\mathrm{D}(a, c, b, d) = \mathrm{D}(a, c, p(b), d) + |L_b \cap L_d|$.*
4. *Otherwise $\mathrm{D}(a, c, b, d) = \max(\mathrm{D}(a, c, p(b), d), \mathrm{D}(a, c, b, p(d))) + |L_b \cap L_d|$.*

Proof. Each of the cases above holds true as a direct consequence of Lemma 1. ◀

Through a straightforward application of the above lemma, we obtain the following.

▶ **Lemma 7.** *If I_1 and I_2 denote the heights of T_1 and T_2, respectively,* Pairwise Alignments on a Tree *is solvable in $O\left(|V_1||V_2|I_1I_2 + |L(T_1)| + |L(T_2)|\right)$ time and space.*

Proof. The algorithm is a straightforward implementation of Observation 2 and Lemma 6. Namely, from Observation 2 it follows that the values of $|L_a \cap L_b|$, for all $a \in V_1$ and $b \in V_2$, can be computed by the use of algorithm having time and space complexity $O(|V_1||V_2| + |L(T_1)| + |L(T_2)|)$. After computing these values, all entries in D can be computed in the time and space that are proportional to the number of all possible combinations of a, b, c, d, which is bounded by $|V_1||V_2|I_1I_2$. Now, combining the above with the obvious inequality $|V_1||V_2|I_1I_2 \geq |V_1||V_2|$, we have that the overall time and space complexity of the proposed algorithm is $O(|V_1||V_2|I_1I_2 + |L(T_1)| + |L(T_2)|)$. ◀

Let $M : V(T_1) \cup V(T_2) \to V(T_1) \cup V(T_2)$ be the (partial) bijective mapping between vertices v and w of T_1 and T_2, respectively, which correspond to the same crucial vertex u in their common tree T; i.e. $M(v) = w$ and $M(w) = v$.

▶ **Observation 8.** *For any pair of vertices $a, b \in V_1$ (or V_2) the lowest common ancestor of a and b, namely $\mathrm{lca}(a, b)$, has a mapping, $M(\mathrm{lca}(a, b))$ which is equal to $\mathrm{lca}(M(a), M(b))$. For any triplet of vertices $a, b, c \in V_1$ (or V_2), the lowest common ancestor of a, b is equal to the lowest common ancestor of b, c if and only if $\mathrm{lca}(M(a), M(b)) = \mathrm{lca}(M(b), M(c))$.*

We now present our algorithm for computing the size of a maximal common tree, which is a combination of dynamic programming and an algorithm for finding a maximum cost matching.

▶ **Theorem 9.** *The mapping which corresponds to a maximal common tree can be computed in time $O(|V_1||V_2|(|V_1| + |V_2|)\log(|V_1| + |V_2|) + |V_1||V_2|I_1I_2 + |L(T_1)| + |L(T_2)|)$.*

Proof. For $i \in \{1, 2\}$ and $x \in V_i$, let $T_i(x)$ denote multi-labeled tree which is identical to subtree of T_i rooted at x, except that the set of labels assigned to the root node in $T_i(x)$ is empty. We first define function $G : V_1 \times V_2 \to \mathbb{N}$, such that $G(a, b)$ denotes the size of the maximal common tree between trees $T_1(a)$ and $T_2(b)$, where we are assuming that $M(a) = b$. The maximal common tree is then indicated by $\max_{(a,b)\in V_1\times V_2}[G(a, b) + \mathrm{D}(r_1, r_2, a, b)]$. The key observation of our algorithm is that the computation of $G(a, b)$ can be reduced to finding a maximum cost matching for an auxiliary graph. Let $a_1, \ldots, a_n$ be the children of a, and $b_1, \ldots, b_m$ be the children of b. The structure conditions on mapping provide the guarantee that all vertices which are leaves of downward paths from a without internal crucial vertices, lie in distinct subtrees. Using the Observation 8 this implies that each such vertex lies in distinct subtrees with roots $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$. For the simplicity of notation, for $i \in \{1, \ldots, n\}$, denote the subtree of T_1 with root at a_i as F_i. Analogously, we denote subtree of T_2 rooted at b_j, where $j \in \{1, 2, \ldots, m\}$, by H_j. For a pair of subtrees, we define the score of an optimal choice of pairs of mapped vertices; $c(F_i, H_j) = \max_{c\in V(F_i),d\in V(H_j)}(\mathrm{D}(a, b, c, d) + G(c, d))$. Thus the cost of edges in this graph is equal to the size of a maximal common tree if we choose an optimal mapped pair of vertices in these subtrees. However, we should find an optimal choice such that a subtree corresponds to a subtree in another tree, under the condition that we cannot construct correspondence of two subtrees into single subtree; this problem is the well known maximum weighted bipartite matching problem, which can be solved in a polynomial time [19]. Finally, we can construct a bipartite graph on the set of vertices $a_1, \ldots, a_n, b_1, \ldots, b_m$ with the cost of an edge (a_i, b_j) equal to $c(F_i, H_j)$. Thus our algorithm returns the score of an optimal assignment in this graph where the optimal value of the $G(a, b)$ can be reduced to the computation of $G(x, y)$ for all x and y such that $a \preceq x$ and $b \preceq y$ and finding the maximum weighted matching on auxiliary graph (which is complete bipartite graph with $n + m$ vertices and nm edges). Thus we can summarize our algorithm as follows.

- If a or b is a leaf then $G(a,b) = 0$.
- Otherwise let Q be a complete weighted bipartite graph on vertices $a_1, \ldots, a_n, b_1, \ldots, b_m$ (a_i is the root of F_i, and b_j is the root of H_j) in which the weight of an edge (a_i, b_j) is equal to $\max_{c \in V(F_i), d \in V(H_j)} (\mathrm{D}(a_i, b_j, c, d) + G(c,d))$;then $G(a,b)$ is equal to the cost of an optimal assignment in Q.

The time to construct auxiliary graphs is bounded by $O(|V_1||V_2|I_1I_2)$, the most time consuming part of this algorithm is the solution to the assignment problem. However, the time needed to solve this problem for a graph with n vertices and m edges is bounded by $O(nm \log n)$. If n_a is the number of children of a vertex a in the first tree, and n_b is the number of children of a vertex b in the second tree then the total time is bounded by $O(\sum_{a,b}(n_a + n_b)n_a n_b \log(n_a + n_b))$ which can be bound (up to constant factor) by $O(|V_1||V_2|(|V_1| + |V_2|)\log(|V_1| + |V_2|))$ or $O((|V_1|\sum_j n_b^2 + |V_2|\sum_i n_a^2)\log(|V_1| + |V_2|))$. The second bound is significantly better if the maximal degree of a vertex is bounded by a small value. ◀

The above theorem finalizes the description of our algorithm for computing the edit distance between two multi-labeled trees.

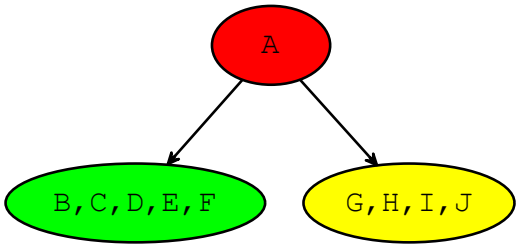
5 Discussion and an application

5.1 The existing measures and their limitations

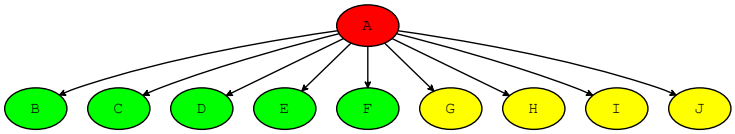
There are number of measures in the literature that are being used to compare clonal trees. Two of the most widely used measures include: (1) Ancestor-Descendant Accuracy (ADA), measure which considers only mutations originating at nodes(clones) which are in ancestor-descendant relationship in the true tree and returns the fraction of pairs of such mutations for which the relationship is preserved in the inferred tree. (2) Different-Lineage Accuracy (DLA), defined analogously as ADA, where only pairs of mutations originating from different clones which are in neither ancestor-descendant nor descendant-ancestor relationship are considered. In addition to these two measures, used in [27, 15, 16, 17] and elsewhere, (3) Clustering Accuracy (CA) [15] and (4) Co-Clustering Accuracy (CCA) [17] were also introduced in order to measure the accuracy in the placement of mutations originating from the same clone in true tree. CA measures the fraction of label pairs that are both co-located in the same vertex in both trees, whereas CCA measures the proximity in the inferred tree of pairs of mutations originating from the same clone in true tree (see [15] and [17] for definitions of CA and CCA). Finally, (5) Pair-wise Marker Shortest Path Distance (PMSPD) [25] is (symmetric) distance measure calculated as the sum, over all label pairs, of the absolute difference of path length between the two labels in true tree with the equivalent length calculated in the inferred tree.

All of the above mentioned are designed to compare inferred tree against the given true tree and no single measure can capture the overall similarity/difference between two arbitrary trees. Furthermore, for each of the measures there exist cases where it returns high similarity for topologically very different true and inferred trees. We will illustrate this below by presenting several examples using trees from Figure 2 where true tree and four trees inferred by (hypothetical) methods are shown. Each vertex in any one of these trees have one or more labels (corresponding to mutations in clonal trees) represented by $A, B, C, \ldots, J$.

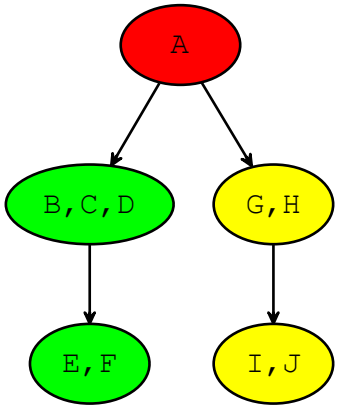
For ADA measure, one needs to consider all pairs of labels in the true tree: $\{(A,B), (A,C), (A,D), (A,E), (A,F), (A,G), (A,H), (A,I), (A,J)\}$. We see that 'Inferred tree 1' has the maximum score despite being topologically very different from 'True tree'. The same tree can be



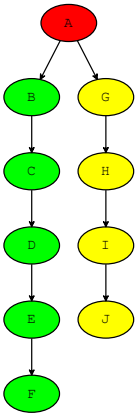
(a) True tree



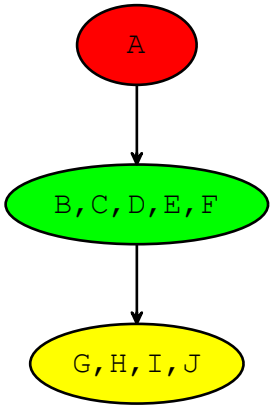
(b) Inferred tree 1



(c) Inferred tree 2



(d) Inferred tree 3



(e) Inferred tree 4

Figure 2 (a) True clonal tree depicting the evolution of hypothetical tumor. (b)-(e) Hypothetical trees inferred by methods for reconstructing history of tumor evolution (input data to these methods is assumed to be obtained from the hypothetical tumor mentioned in the description of 'True tree'). These trees are used as examples which demonstrate limitations of the existing measures for calculating similarity/distance between true and each of the four inferred trees (details provided in Section 5.1). In Section 5.2.1 we discuss the application of MLTED in calculating similarities between these pairs of trees.

used as an illustration for the limitations of DLA measure where the following set of label pairs need to be considered in true tree $\{(B,G),(B,H),(B,I),(B,J),(C,G),(C,H),(C,I),(C,J),(D,G),(D,H),(D,I),(D,J),(E,G),(E,H),(E,I),(E,J),(F,G),(F,H),(F,I),(F,J)\}$. Clustering of mutations in 'Inferred tree 4' is in the perfect agreement with the clustering in the 'True tree' hence both CA and CCA measures will return maximum score for this tree, even though it is also topologically very different from 'True tree'. Finally, the calculation of the PMSPD measure between the 'True tree' and 'Inferred tree 1', as well as 'Inferred tree 2', is shown in Figure 3. This measure assigns the same score to these two inferred trees, despite the fact that 'Inferred tree 2' is, from the perspective of interpreting tumor evolution, much closer to 'True tree'.

5.2 Applications of MLTED

In order to facilitate the interpretation of results, for two arbitrary trees T_1 and T_2, in addition to the MLTED similarity measure which returns the number of mutations in common tree of T_1 and T_2 and is denoted here as $MLTED(T_1, T_2)$, we also introduce MLTED-normalized(T_1, T_2) defined as $\frac{MLTED(T_1,T_2)}{\max(a,b)}$, where a and b denote number of mutations in T_1 and T_2. MLTED-normalized can be interpreted as similarity measure which takes values from $[0, 1]$, with higher values denoting higher similarity among trees. In the discussion of results below, all presented scores represent MLTED-normalized similarity measure, although it is obviously equivalent to MLTED (assuming that the sets of node labels are known for both trees, which is true in all of our comparisons).

5.2.1 Application to the synthetic examples with the available ground truth

In this section we discuss similarity between true and inferred trees shown in Figure 2.

'Inferred tree 1' has relatively low score equal to 0.3 which rewards the proper placement of mutation A and correctly inferred phylogenetic relations for pairs of mutations originating from different clones, but penalizes for extensive branching which leads to the inaccurate placement to different branches of mutations originating from the same clone, as well as to significant topological differences between this and true tree. In contrast, and as expected based on our discussion from the introduction, 'Inferred tree 2' (which represents slightly refined version of 'True tree' where green and yellow clones are each split into two adjacent clones belonging to the same branch) and 'Inferred tree 3' (which represents fully resolved mutation tree that can be obtained from 'True tree') both have score 1. 'Inferred tree 4', having score 0.6, is rewarded for the proper placement of mutation A and large cluster of mutations appearing for the first time at green clone, but is penalized for inaccurate placement of yellow clone from where 4 out of 10 mutations originate.

5.2.2 Application to real data

In order to demonstrate the application of measure developed in this work in real settings where true tree is usually not available, we analyzed two datasets obtained by sequencing real samples of triple-negative breast cancer (TNBC) and acute lymphoblastic leukemia (ALL). For each sample, we inferred trees of tumor evolution by the use of SCITE [14], SiFit [10] and PhISCS[2]. We provide more details about these methods and parameters used in running them, as well as details of obtaining real data, in the Appendix. Inferred trees and very detailed discussion of the calculated MLTED-normalized scores for pairs of inferred trees are shown in Figure 4 and Figure 5 (for the TNBC sample) and Figure 6 (for the ALL sample). We show that MLTED-normalized score recognizes high similarity in the placement of vast majority of mutations between two trees (as demonstrated for trees inferred by PhISCS and SiFit for TNBC sample where score equals 0.82), but also penalizes for topological differences and different sorting of mutations along linear chains (as demonstrated for trees inferred by SCITE and SiFit for ALL sample where the score equals 0.69).

[2] Available at `https://github.com/haghshenas/PhISCS`

6 Conclusion and Future Work

Comparing clonal and mutation trees inferred by different methods and/or by the use of different types of input data, currently represents an important challenge in intra-tumor heterogeneity and evolution studies. In this work, we show that the most widely used measures for comparing clonal/mutation trees usually capture only some specific aspects of tree similarity and can also report the similarity score which is in sharp contrast to the expected result (e.g. perfect score is sometimes given to inferred tree which is very different from the true tree). Motivated by these, we introduce new measure, termed MLTED, for calculating similarity between trees of tumor evolution. From the above discussion and results on synthetic and trees obtained by analyzing real datasets, we conclude that MLTED successfully captures similarities and differences, relevant in proper interpretation of tumor evolution, between two given trees. The proposed similarity measure represents a promising alternative to the existing measures and has wider domain of applications (as it does not require the presence of reference tree). We expect that MLTED becomes a part of the gold standard in assessing the performance of methods for inferring trees of tumor evolution on simulated data, as well as in comparing similarity and finding consensus tree of trees reported by different methods in the cases where real data is used as the input and ground truth is usually not available.

In the current implementation, MLTED is designed for trees built by the use of SNVs under the ISA. Such trees are an output of the vast majority of the existing methods for studying tumor evolution. However, some recent studies based on the analysis of single-cell sequencing datasets suggest the possibility of ISA violation in some cases [23]. With the rapid advancements in the field of single-cell sequencing, we expect developments of sophisticated computational methods based on finite sites model and using SNVs, copy number and other structural aberrations in inferring clonal/mutation trees. These will also require some future work in extending MLTED to properly capture the key differences among trees reported by such methods.

References

1. A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015. doi:10.1145/2746539.2746612.
2. P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005. doi:10.1016/j.tcs.2004.12.030.
3. W. Chen. More efficient algorithm for ordered tree inclusion. *J. Algorithms*, 26(2):370–385, 1998. doi:10.1006/jagm.1997.0899.
4. W. Chen. New algorithm for ordered tree-to-tree correction problem. *J. Algorithms*, 40(2):135–158, 2001. doi:10.1006/jagm.2001.1170.
5. Nilgun Donmez, Salem Malikic, Alexander W. Wyatt, Martin E. Gleave, Colin Collins, and S Cenk Sahinalp. Clonality inference from single tumor samples using low-coverage sequence data. *Journal of Computational Biology*, 24(6):515–523, 2017. doi:10.1089/cmb.2016.0148.
6. A. G. Deshwar et al. Phylowgs: reconstructing subclonal composition and evolution from whole-genome sequencing of tumors. *Genome biology*, 16(1):35, 2015.
7. C. Gawad et al. 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.

8 El-Kebir M. et al. Inferring the mutational history of a tumor using multi-state perfect phylogeny mixtures. *Cell systems*, 3(1):43–53, 2016.

9 F. Strino et al. Trap: a tree approach for fingerprinting subclonal tumor composition. *Nucleic acids research*, 41(17):e165–e165, 2013.

10 H. Zafar et al. Sifit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome biology*, 18(1):178, 2017.

11 Hajirasouliha I. et al. A combinatorial approach for analyzing intra-tumor heterogeneity from high-throughput sequencing data. *Bioinformatics*, 30(12):i78–i86, 2014.

12 J. Kuipers et al. Advances in understanding tumour evolution through single-cell sequencing. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1867(2):127–138, 2017.

13 Jiang T. et al. Alignment of trees - an alternative to tree edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995. doi:10.1016/0304-3975(95)80029-9.

14 K. Jahn et al. Tree inference for single-cell data. *Genome biology*, 17(1):86, 2016.

15 M. El-Kebir et al. Reconstruction of clonal trees and tumor composition from multi-sample sequencing data. *Bioinformatics*, 31(12):i62–i70, 2015.

16 S. Malikic et al. Clonality inference in multiple tumor samples using phylogeny. *Bioinformatics*, 31(9):1349–1356, 2015.

17 S. Malikic et al. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *To appear in proceedings of RECOMB*, 2018.

18 W. Jiao et al. Inferring clonal evolution of tumors from single nucleotide somatic mutations. *BMC bioinformatics*, 15(1):35, 2014.

19 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

20 J. Jansson and A. Lingas. A fast algorithm for optimal alignment between similar ordered trees. *Fundam. Inform.*, 56(1-2):105–120, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi56-1-2-07.

21 R. Kim, K.I. & Simon. Using single cell sequencing data to model the evolutionary history of a tumor. *BMC bioinformatics*, 15(1):27, 2014.

22 P.N. Klein. Computing the edit-distance between unrooted ordered trees. In *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, pages 91–102, 1998. doi:10.1007/3-540-68530-8_8.

23 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.

24 P. Kilpeläinen & H. Mannila. Ordered and unordered tree inclusion. *SIAM J. Comput.*, 24(2):340–356, 1995. doi:10.1137/S0097539791218202.

25 E. M. Ross & F. Markowetz. Onconem: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17(1):69, 2016.

26 P.C. Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976.

27 Victoria Popic, Raheleh Salari, Iman Hajirasouliha, Dorna Kashef-Haghighi, Robert B West, and Serafim Batzoglou. Fast and scalable inference of multi-sample cancer lineages. *Genome biology*, 16(1):91, 2015.

28 Daniele Ramazzotti, Alex Graudenzi, Luca De Sano, Marco Antoniotti, and Giulio Caravagna. Learning mutational graphs of individual tumor evolution from multi-sample sequencing data. *arXiv preprint arXiv:1709.01076*, 2017.

29 S. Muthukrishnan & S.C. Sahinalp. An efficient algorithm for sequence comparison with block reversals. *Theor. Comput. Sci.*, 321(1):95–101, 2004. doi:10.1016/j.tcs.2003.05.005.

30 S. M. Selkow. The tree-to-tree editing problem. *Inf. Process. Lett.*, 6(6):184–186, 1977. doi:10.1016/0020-0190(77)90064-3.

31 K. Zhang & D.E. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989. doi:10.1137/0218082.

32 D. Shapira & J.A. Storer. Edit distance with block deletions. *Algorithms*, 4(1):40–60, 2011. doi:10.3390/a4010040.

33 Kuo-Chung T. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979. doi:10.1145/322139.322143.

34 J. Matoušek & R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992. doi:10.1016/0012-365X(92)90687-B.

35 R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

36 Yong Wang, Jill Waters, Marco L Leung, Anna Unruh, Whijae Roh, Xiuqing Shi, Ken Chen, Paul Scheet, Selina Vattathil, Han Liang, et al. Clonal evolution in breast cancer revealed by single nucleus genome sequencing. *Nature*, 512(7513):155, 2014.

37 D. Shasha & K. Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990. doi:10.1016/0196-6774(90)90011-3.

38 K. Zhang. Algorithms for the constrained editing distance between ordered labeled trees and related problems. *Pattern Recognition*, 28(3):463–474, 1995. doi:10.1016/0031-3203(94)00109-Y.

39 K. Zhang and T. Jiang. Some MAX snp-hard results concerning unordered labeled trees. *Inf. Process. Lett.*, 49(5):249–254, 1994. doi:10.1016/0020-0190(94)90062-0.

Supplementary Figures

	J	I	H	G	F	E	D	C	B	A
A	1	1	1	1	1	1	1	1	1	0
B	2	2	2	2	0	0	0	0	0	
C	2	2	2	2	0	0	0	0		
D	2	2	2	2	0	0	0			
E	2	2	2	2	0	0				
F	2	2	2	2	0					
G	0	0	0	0						
H	0	0	0							
I	0	0								
J	0									

(a) Distances for 'True tree'

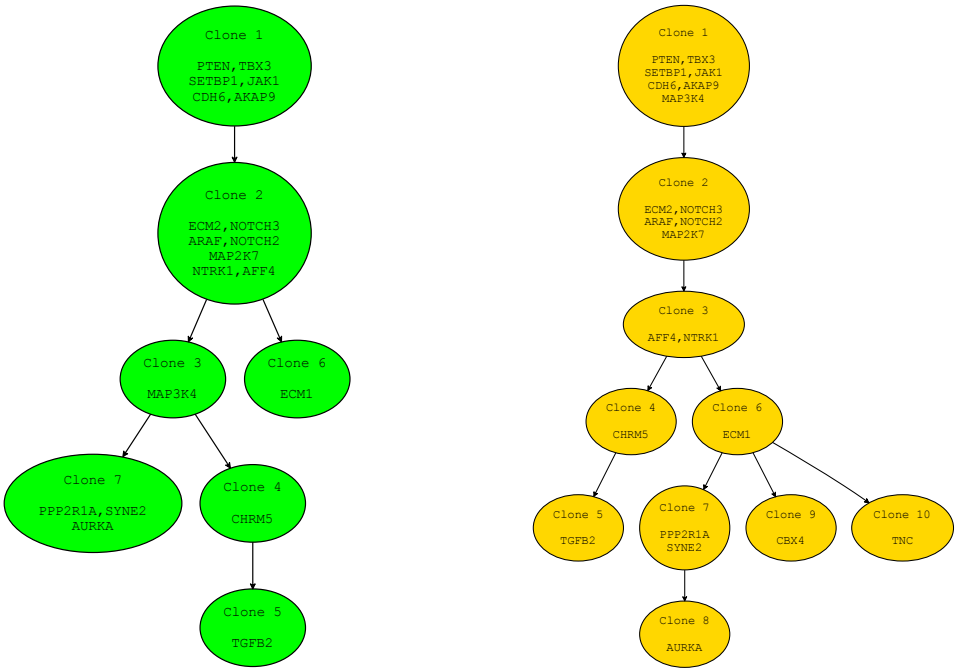
	J	I	H	G	F	E	D	C	B	A
A	1	1	1	1	1	1	1	1	1	0
B	2	2	2	2	2	2	2	2	0	
C	2	2	2	2	2	2	2	0		
D	2	2	2	2	2	2	0			
E	2	2	2	2	2	0				
F	2	2	2	2	0					
G	2	2	2	0						
H	2	2	0							
I	2	0								
J	0									

(b) Distances for 'Inferred tree 1'

	J	I	H	G	F	E	D	C	B	A
A	2	2	1	1	2	2	1	1	1	0
B	3	3	2	2	1	1	0	0	0	
C	3	3	2	2	1	1	0	0		
D	3	3	2	2	1	1	0			
E	4	4	3	3	0	0				
F	4	4	3	3	0					
G	1	1	0	0						
H	1	1	0							
I	0	0								
J	0									

(c) Distances for 'Inferred tree 2'

Figure 3 Distances between pairs of labels required for calculating Pair-wise Marker Shortest Path Distance (PMSPD) for trees from Figure 2. Entries in each matrix represent length of path between labels (note that labels are shown in the first row and the first column of each matrix). Distance is calculated as the sum of absolute values of differences between pairs of entries which are at the same position in both matrices. Red colored entries in labels pairwise distance matrix shown in (b)/(c) differ from the corresponding entries in matrix for true tree shown in (a) and therefore contribute to the overall distance. PMSPD assigns the same score to 'Inferred tree 1' and 'Inferred tree 2', despite the fact that 'Inferred tree 2' is, from the perspective of interpreting tumor evolution, much closer to 'True tree'.



(a) Tree for TNBC dataset inferred by SiFit (b) Tree for TNBC dataset inferred by PhISCS

Figure 4 Clonal trees of tumor evolution, inferred by SiFit and PhISCS, for triple-negative breast cancer (TNBC) dataset originally published in [36] and consisting of the binary presence/absence profile of 22 mutations across 16 single cells. Names of the clones are assumed not to be included as part of the node label. Trees are very similar to each other in placement of the vast majority of mutations: (i) Clone 1 in the SiFit tree is almost identical (with respect to the set of mutations assigned to its label) to Clone 1 in PhISCS tree (ii) Clone 2 in SiFit tree is split into two adjacent clones, namely Clone 2 and Clone 3, in PhISCS tree. Analogous applies to Clone 7. (iii) The order of mutations in genes CHRM5 and TGFB2, as well as in most other pairs of mutations (including the pairs where both mutations are at the same node), is same among the trees. Notable exceptions leading to some dissimilarities between the trees include mutations in genes MAP3K4 and ECM1. In addition, mutations in genes CBX4 and TNC are absent in tree reported by SiFit. Removing these four mutations and their corresponding nodes from each tree (if present) and assigning each of the Clone 4 and Clone 7 in SiFit tree as child of Clone 2, and Clone 7 as child of Clone 3 in PhISCS tree, we obtain trees which are same up to the existence of splits of single into two adjacent clones belonging to the same lineage (see (ii) from above). MLTED-normalized score for the two trees equals 0.82, which well reflects the overall high topological similarity and concordance in ordering pairs of mutations.

,

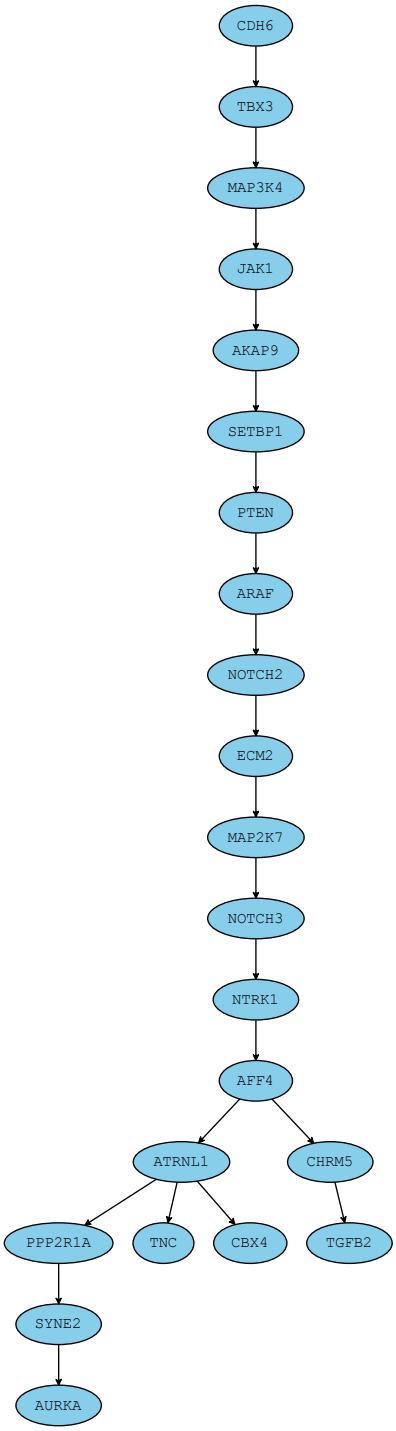
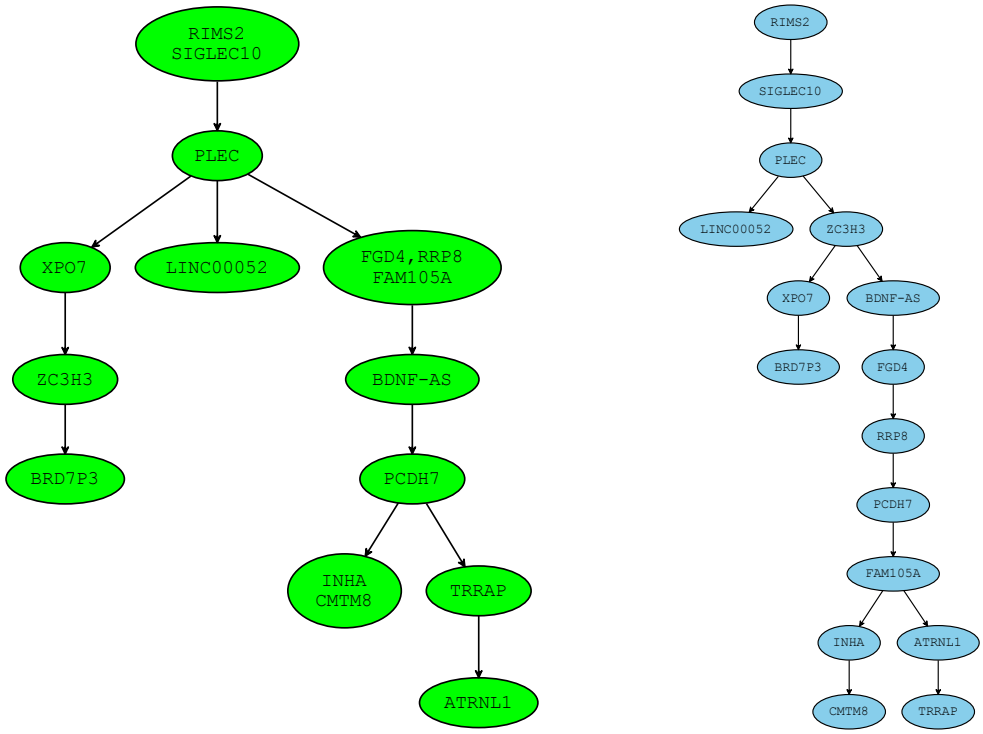


Figure 5 Mutation tree for TNBC dataset (see Figure 4 for details) inferred by SCITE. This tree can be obtained from PhISCS tree by expanding nodes having more than one label, hence MLTED-normalized score between the two trees is maximum possible (i.e. equals 1). Compared with tree inferred by SiFit, SCITE tree has analogous topological similarities and differences as tree inferred by PhISCS, and MLTED-normalized score for these two trees is also equal to 0.82.

,



(a) Tree for ALL dataset inferred by SiFit (b) Tree for ALL dataset inferred by SCITE

Figure 6 Trees inferred by SCITE and SiFit for acute lymphoblastic leukemia (ALL) patient dataset from [7] consisting of 115 single cells and 16 mutations. Unsurprisingly, due to large number of single-cells in this dataset, sequencing noise and similarities in the scoring schemes used in PhISCS and SCITE (see Section A) both methods report the same mutation tree so we only focus on SCITE in this discsussion. The most notable difference among the two trees is in the placement and ordering of mutations in genes ZC3H3, XPO7 and BRD7P3 as well as in the ordering of mutations in genes FGD, RRP8, FAM105A, BDNF-AS and PCDH7. Furthermore, the relative order also differs for mutations in genes TRRAP and ATRNL1. However, in contrast to these important differences, the trees still share most of the major branching events in tumor evolution and have consistent ancestor-descendant order for most of the pairs of mutations. All these are reflected in MLTED-normalized score of 0.69 assigned to this pair of trees.

,

A Details of obtaining trees of tumor evolution for the real data sets

A.1 Summary of methods used for inferring trees of tumor evolution

In this work, we inferred trees of tumor evolution by the use of SCITE [14], SiFit [10] and PhISCS[3]. Each of the methods takes as the input single-cell sequencing (SCS) data matrix and estimated noise rates of SCS experiment. The underlying scoring used in PhISCS is analogous to that in SCITE and the major difference among the two methods is in the type of tree returned in the output. While SCITE searches for the maximum likelihood mutation tree, PhISCS reports the maximum likelihood clonal tree. Due to the equivalence in tree scoring, assuming that both methods find the optimal solution, clonal tree reported by PhISCS is expected to represent compressed version of mutation tree reported by SCITE (i.e. we expect that tree reported by SCITE belongs to the set of mutation trees which can be obtained from clonal tree reported by PhISCS). Similarly as PhISCS, SiFit also returns clonal tree of tumor evolution but uses different tree search methodology and does not necessarily yield the same output as PhISCS nor SCITE (as demonstrated in [10]).

A.2 Details of obtaining input data and running SCITE, SiFit and PhISCS

We obtained binary SCS data mutation matrix for TNBC patient sample from [28] and for ALL patient sample from [7]. For each sample, false positive and false negative rates of sequencing experiment were estimated in the original studies [36, 7] and provided as the input to the methods used in the analysis. In order to obtain better convergence, we run MCMC based methods SiFit and SCITE for very large number of iterations. For SiFit, we set number of iterations to 5,000,000. For SCITE we set number of repetitions of the MCMC to 3 and chain length of each MCMC repetition to 1,000,000. PhISCS is combinatorial optimization based method which provided guarantee of the optimality for each solution.

[3] Available at `https://github.com/haghshenas/PhISCS`

Heuristic Algorithms for the Maximum Colorful Subtree Problem

Kai Dührkop
Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany
kai.duehrkop@uni-jena.de

Marie A. Lataretu
Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany
marie.lataretu@uni-jena.de

W. Timothy J. White
Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany, and
Berlin Institute of Health, Berlin, Germany
tim.white@bihealth.de
https://orcid.org/0000-0002-1997-0176

Sebastian Böcker
Chair for Bioinformatics, Friedrich-Schiller-University, Jena, Germany
sebastian.boecker@uni-jena.de
https://orcid.org/0000-0002-9304-8091

Abstract

In metabolomics, small molecules are structurally elucidated using tandem mass spectrometry (MS/MS); this computational task can be formulated as the Maximum Colorful Subtree problem, which is NP-hard. Unfortunately, data from a single metabolite requires us to solve hundreds or thousands of instances of this problem – and in a single Liquid Chromatography MS/MS run, hundreds or thousands of metabolites are measured.

Here, we comprehensively evaluate the performance of several heuristic algorithms for the problem. Unfortunately, as is often the case in bioinformatics, the structure of the (chemically) true solution is not known to us; therefore we can only evaluate against the optimal solution of an instance. Evaluating the quality of a heuristic based on scores can be misleading: Even a slightly suboptimal solution can be structurally very different from the optimal solution, but it is the structure of a solution and not its score that is relevant for the downstream analysis. To this end, we propose a different evaluation setup: Given a set of candidate instances of which exactly one is known to be correct, the heuristic in question solves each instance to the best of its ability, producing a score for each instance, which is then used to rank the instances. We then evaluate whether the correct instance is ranked highly by the heuristic.

We find that one particular heuristic consistently ranks the correct instance in a top position. We also find that the scores of the best heuristic solutions are very close to the optimal score; in contrast, the structure of the solutions can deviate significantly from the optimal structures. Integrating the heuristic allowed us to speed up computations in practice by a factor of 100-fold.

2012 ACM Subject Classification Applied computing → Computational biology, Applied computing → Metabolomics / metabonomics

Keywords and phrases Fragmentation trees, Computational mass spectrometry

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.23

Funding WTJW funded by Deutsche Forschungsgemeinschaft (grant BO 1910/9).

1 Introduction

Metabolomics characterizes the collection of all metabolites in a biological cell, tissue, organ or organism using high-throughput techniques [10]. Liquid Chromatography Mass Spectrometry (LC-MS) is one of the predominant experimental platforms for this task [1] and can detect compounds at the attogram level [8]. Today, a major challenge is to determine the identities of the thousands of metabolites detected in one LC-MS run. This is also true for related fields such as natural products research [16,21], biomarker discovery [21], environmental science [13], or food science. Tandem mass spectrometry (MS/MS) is used to derive information about the structure of a metabolite by fragmenting the molecule and recording the masses of its fragments. Interpretation of the hundreds to thousands of MS/MS spectra generated in a single LC-MS run remains a bottleneck in the analytical pipeline [19]. MS/MS data is usually searched against spectral libraries [17], but only a small number of metabolites (around 2 %) can be identified in this manner [4].

Fragmentation trees were introduced in 2008 [3] and were initially targeted towards identifying the molecular formula of the unknown small molecule. Later, it was shown that the structure of fragmentation trees contains valuable information for structural elucidation of the underlying molecule [11]. In particular, CSI:FingerID combines fragmentation tree computation with multiple kernel learning on these trees [6, 15], and is the currently best-performing method for searching MS/MS data in *structure* databases [14]. Computing an optimum fragmentation tree naturally leads to the Maximum Colorful Subtree problem [2,3]. Unfortunately, this problem is NP-hard and also hard to approximate [7, 12]. Algorithms exist to solve the problem either heuristically [12] or exactly [3, 12, 20]. The problem is a variant of the well-studied Graph Motif problem [5, 9].

Optimization problems in bioinformatics research are designed so that the optimal solution (with regards to the objective function) is "similar" to the true solution, such as the biologically correct phylogenetic tree, the "true" sequence alignment, etc. Unfortunately, many of these optimization problems are NP-hard; furthermore, the true solution is often not known to us. Approximation algorithms are algorithms for (usually) NP-hard problems with provable guarantees on the distance of the returned solution to the optimal one. But this provable guarantee is only for the objective function; in bioinformatics research, we are rarely interested in the objective function value beyond using it to find the optimal solution. Heuristics in bioinformatics are usually designed to find solutions structurally similar to the optimum solution or, even better, the true solution. This makes it intrinsically difficult to evaluate the performance of these heuristics, as we have to define a measure on the structural similarity between the heuristic solution and the true solution; furthermore, the true solution has to be known, which is often not the case.

We will use an alternative approach to evaluate the performance of a heuristic: For many applications, one biological instance results in many computational instances, corresponding to candidates or hypotheses; the score of the solution to each instance is used to rank its corresponding hypothesis. Although the true solution may not be known, we may have information regarding the correct candidate or hypothesis. To this end, we can evaluate a heuristic based on its ability to top-rank the correct candidate.

We propose several heuristics for the Maximum Colorful Subtree problem, and evaluate these heuristics with regards to their ranking quality. We find that one particular heuristic allows us to quickly shrink the set of plausible candidates (molecular formulas of the precursor molecule). This can be used as a filter, such that optimum solutions have to be sought only for a (preferably small) subset of candidates. We also evaluate whether the structure of the constructed solutions is similar to the optimum solution.

2 Fragmentation trees and the Maximum Colorful Subtree problem

Tandem mass spectrometry selects ions with a particular mass, fragments these ions by collision with neutral molecules such as argon or nitrogen, then records the masses of the resulting fragments. Assuming that only identical copies with identical structure are selected, we say that one *precursor ion* with some *precursor mass* was selected. Mass spectrometry measures mass-to-charge instead of mass, but we may assume that all small molecules carry a single charge. Masses of fragments of the molecule are recorded as *peaks* in the MS/MS spectrum.

Details of how to transform the MS/MS spectrum of an (unknown) compound into one or more instances of the Maximum Colorful Subtree problem have been published elsewhere [2, 3]; we briefly recapitulate the process. We consider all molecular formulas from some ground set, such as, all molecular formulas built from the elements CHNOPS. We *decompose* the precursor mass into all possible candidate molecular formulas from this ground set; each candidate molecular formula generates one instance (graph). For each instance, we decompose the fragment peaks in the MS/MS spectrum, ensuring that each fragment molecular formula is a subformula of the candidate molecular formula for the precursor mass. These molecular formulas constitute the nodes of a graph; each node is colored by the peak it stems from. An edge, representing a possible fragmentation event, is present between molecular formulas u, v if and only if v is a proper subformula of u. Now, both nodes and edges receive a certain weight [2], based both on prior knowledge (e.g., distribution of loss masses) and the data (e.g., mass difference between a peak and its hypothetical molecular formula); but as pointed out in [3], we may encode both kinds of weights using only edge weights. SIRIUS 4 default weights are used, see [2]. The maximum colorful subtree within each instance is computed, and its weight is used to rank the corresponding candidate molecular formula of the precursor peak.

We can formulate fragmentation tree computation as a stochastic optimization problem, where we try to "best explain" the observed data (the fragmentation spectrum of a small molecule obtained through tandem mass spectrometry) under a hypothesis (the molecular formula of the small molecule, represented by the root vertex) using a maximum a posteriori estimator. The likelihood function rewards hypotheses that explain high-intensity peaks. Each peak in the fragmentation spectrum must be explained at most once, in accordance with the parsimony principle. Böcker and Dührkop [2] then show that one can find the maximum a posteriori hypothesis by solving a particular Maximum Colorful Subtree instance, where edge weights are logs of probabilities of fragmentation events under the model.

We now describe the formal Maximum Colorful Subtree optimization problem. Let $G = (V, E)$ be a node-colored, rooted, directed acyclic graph (DAG) with root $r \in V$ and edge weights $w : E \to \mathbb{R}$. Let $C(G)$ be the set of colors used in G, let $c(v) \in C(G)$ be the color assigned to node $v \in V$, and let $c(U) := \{c(v) \mid v \in U\}$ for $U \subseteq V$. We will consider subtrees $T = (V_T, E_T)$ of G rooted at r. We say that T is *colorful* if all of its nodes have different colors. The Maximum Colorful Subtree problem asks to find a colorful r-rooted subtree T of G of maximum weight. We may assume that G is (weakly) connected and that r is the unique source of G, as we can remove all nodes from the graph which cannot be reached by a path from the root r, without changing the optimal solution. We say that a subgraph $G' \subseteq G$ is *full* if $v \in G'$ implies that every edge and vertex reachable from v in G is also in G'.

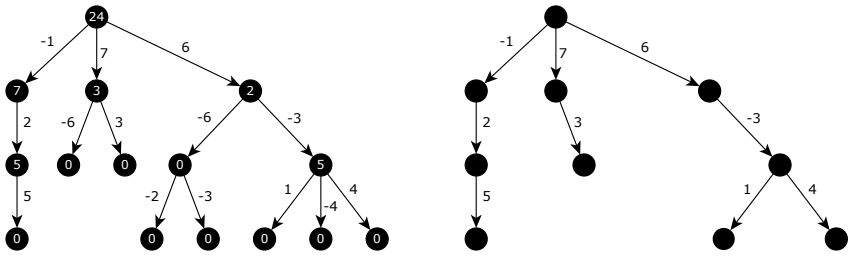


Figure 1 Illustration of the Remove Dangling Subtrees (RDS) postprocessing. Left: Input tree, where each node v is labeled by its score $D[v]$. Right: Output tree of weight 24.

We note that previous work on the problem also makes the assumption of a single source, albeit usually implicitly [3, 11, 12, 20]. From an algorithmic standpoint, problem variants with or without a given root are "basically equivalent": Given an algorithm that does not assume a fixed root r, we can solve an instance of the problem variant with root r by introducing a superroot r^* connected solely to r, sufficiently large edge weight $w(r^*, r)$ and a new color for r^*. For the reverse direction, we solve the problem for every $r \in V$, then choose the best solution.

There are two peculiarities when computing fragmentation trees that are different from the general problem, and that we will make use of here. First, any DAG used for fragmentation tree computation is *transitive*: That is, $uv \in E$ and $vw \in E$ implies $uw \in E$. Second, a coloring $c : V \to C(G)$ of DAG $G = (V, E)$ is *order-preserving* if there is an ordering '$\prec$' on the colors $C(G)$ such that $c(u) \prec c(v)$ holds for every edge uv of G [7]. Computing fragmentation trees naturally results in order-preserving colors, as nodes can be colored by the fragment mass that is responsible for this node, and edges exist only between nodes from larger to smaller masses. The Maximum Colorful Arborescence problem [7] asks to find a rooted colorful subtree T of G of maximum weight, where G is a DAG with order-preserving colors and edge weights. See [7] for numerous complexity results. Here, we will stick with the name "Maximum Colorful Subtree problem", but nevertheless assume that the coloring is order-preserving, unless indicated otherwise.

3 Heuristics for the Maximum Colorful Subtree problem

The following postprocessing methods can be applied to a tree $T = (V_T, E_T)$ *after* any heuristic: The **Remove Dangling Edges** (RDE) postprocessing iteratively removes edges uv from T, where v is a leaf and $w(uv) < 0$; this is repeated until no more such edges are found. In contrast, the **Remove Dangling Subtrees** (RDS) postprocessing does not consider a single edge at a time, but rather full subtrees: Each node $u \in V_T$ is scored by the maximum weight of any full subtree rooted in u. Score $D[u]$ can be computed using dynamic programming:

$$D[u] := \sum\nolimits_{uv \in E_T} \max\{0, w(u, v) + D[v]\}$$

Clearly, $D[u] \geq 0$. For each edge uv with $w(u, v) + D[v] < 0$ we remove uv and the subtree below it. Both postprocessings can be computed in $O(|V_T|)$ time using a tree traversal, as every edge is considered once and $|E_T| = |V_T| - 1$. Figure 1 shows an example of the RDS postprocessing.

We now present heuristics for finding a colorful subtree with root r in a transitive DAG with order-preserving coloring and unique source r.

- *Kruskal-style.* This heuristic sorts all edges of the graph by decreasing edge weight, then iteratively adds edges from the sorted list, ensuring that the growing subgraph is colorful and that each node has at most one incoming edge. Since r is the unique source of G, and since G is transitive, this will ultimately result in a colorful subtree of G. This heuristic is similar to Kruskal's algorithm for computing a minimum spanning tree; it was called "greedy heuristic" in [3].
- *Prim-style.* This heuristic progresses similarly to Prim's algorithm for calculating a minimum spanning tree: The tree $T = (V_T, E_T)$ initially contains only the root r of G. In every step, we consider all edges uv with $u \in V_T$ and $v \notin V_T$ such that $c(v) \notin c(V_T)$; among these, we choose the edge with maximum weight and add it to the tree. We repeat until all colors in the graph are used in the tree; recall that G is transitive, so $rv \in E$ for each $v \neq r$. We explicitly do not quit when adding the first negative-weight edge uv, as the newly reached node v may allow us to later add other edges with positive weight. The Prim-style heuristic will usually result in a different tree than the Kruskal-style heuristic, due to the colorfulness constraint.
- *Insertion.* This heuristic is a modification of the "insertion heuristic" from [12]. We again start with a tree $T = (V_T, E_T)$ containing only the root r of G. The heuristic greedily attaches nodes labeled with unused colors. For every node v with $c(v) = c'$ unused, and every node u already part of the solution, we calculate how much we gain by attaching v to u. To calculate this gain $I(u, v)$, we take into account the score of the edge uv as well as the possibility of rerouting other outgoing edges of u through v:

$$I(u,v) := w(uv) + \sum_{x \in V_T, w(vx) > w(ux)} \big(w(vx) - w(ux)\big)$$

  where we assume $w(uv) = -\infty$ if $uv \notin E$. The node with maximum gain is then attached to the partial solution, and edges are rerouted as required. See [12] for details; different from there, we do not iterate over colors in some fixed order but instead, consider all unused colors in every step.
- *Top-down.* The top-down heuristic [3] is also greedy, but always adds paths beginning at the root to the partial solution. The partial solution initially contains only the root r of G. The heuristic greedily constructs a path starting at the root which is added to the partial solution; the next node of the path is chosen so that it maximizes the weight of the added edge, simultaneously ensuring that the partial solution remains colorful and does not violate the tree property. If no such edge exists, the algorithm restarts at the root, and searches for another path. It terminates if no edge at the root can be selected. In the resulting tree, all internal nodes but the root have exactly one child. This heuristic extends even simpler heuristics that attach all nodes to the root, which have been in frequent use for molecular formula determination from MS/MS data.
- *Critical Path*[1]. Again, we iteratively build a tree $T = (V_T, E_T)$; initially, the partial solution T contains only the root r of G. The score $S[u]$ of a node $u \in V$ is the maximum weight of a path p from u to any node v, such that $c(p) \cap c(V_T) \subseteq \{c(u)\}$; that is, the path does not use nodes with colors already present in the tree, except for the color of the starting node. We can compute $S[u]$ using the recurrence

$$S[u] := \max_{uv \in E, c(v) \notin c(V_T)} \{0,\ S[v] + w(uv)\} \tag{1}$$

  where we use that the coloring of G is order-preserving, since in that case no two nodes of the path have the same color. We assume $\max \emptyset = 0$ when computing (1). We iterate over the ordered colors c in reverse order, computing $S[u]$ for all nodes u of color c. The

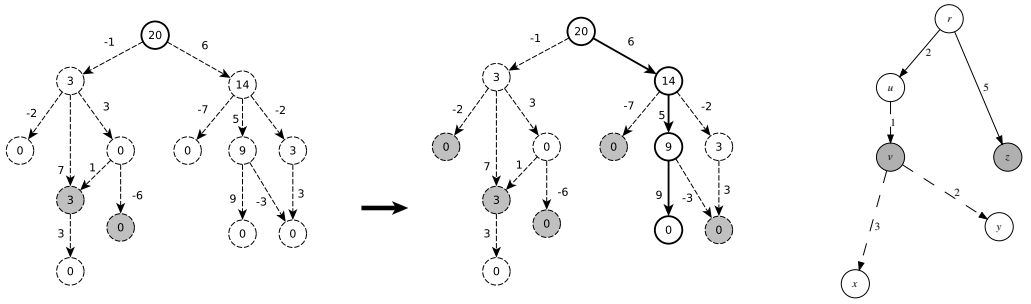


Figure 2 Left: Example for the *Critical Path*[1] heuristic. Nodes are labeled by score, solid lines show the tree, dashed lines the rest of the graph. Grayed-out nodes have colors already used in the subtree. Right: An example input graph for which Critical Path[1] produces a better tree than Critical Path[2]. Nodes v and z are the same color; all other nodes have distinct colors. The two solid edges are the suboptimal tree output by Critical Path[2]. Critical Path[1] initially chooses the path $ruvx$ for a score of 6, then adds vy for a total of 8. Critical Path[2] begins in the same way by choosing the first edge ru of the heaviest path for a score of 2, but in its second step it chooses the weight-5 edge rz, as the heaviest path starting with uv has weight 4. No further edges can be added, so the total weight is 7.

critical path p of maximum score can be found by backtracing from the maximum entry $S[u]$ with $u \in V_T$. We add p to T, then iterate, recomputing S for the new set of used colors $c(V_t)$. See Figure 2 (left) for an example.

- *Critical Path*[2]. This heuristic also relies on critical paths, but adds, in each iteration, only the first edge of the critical path to the partial solution. We note that this heuristic does not dominate the Critical Path heuristic, meaning that in certain cases, the subtree computed by this heuristic has smaller weight than that computed by the Critical Path heuristic; see Figure 2 (right) for an example.
- *Critical Path*[3]. This heuristic combines the Insertion heuristic with the Critical Path heuristic: In each step the heuristic chooses the edge uv with $u \in V_T$ that maximizes the sum of critical path score and insertion score $S[u] + I(u, v)$.
- *Maximum.* All heuristics compute lower bounds of the maximum score; therefore the maximum score over all heuristic solutions is also a lower bound.

3.1 Time complexity of the heuristics

Let $n := |V|$, $m := |E|$, and $k := |C(G)|$. Clearly, $k \leq n$ and in applications, we usually have $k \ll n$. Furthermore, $|V_T| \leq k$ holds for the returned subtree $T = (V_T, E_T)$.

- It is easy to check that the Kruskal-style heuristic has time complexity $O(m \log n)$ for sorting all edges according to weight. Connectivity testing can then be performed in sub-logarithmic time per considered edge using a union-find data structure [18]; checking for colorfulness is easily accommodated by initially placing all nodes of the same color in the same component. The overall time complexity thus remains $O(m \log n)$. Similarly, the Prim-style heuristic requires $O(m \log n)$ time.
- For the Insertion heuristic, computing the gains $I(u, v)$ for a particular v and for all u requires $O(k^2)$ time, since $u, x \in V_T$. Hence, choosing one v to attach to the growing tree requires $O(k^2 n)$ time, resulting in $O(k^3 n)$ total running time.
  However there exists a more complicated yet faster implementation for this heuristic: For each $v \in V$, we maintain two scores, $in(v)$ and $out(v)$, which correspond to the two terms on the RHS of the definition of $I(u, v)$. Specifically, $in(v) = \max_{u \in V_T} w(uv)$, and

$out(v) = \sum_{x \in V_T} \max\{0, w(vx) - w(p_T(x), x)\}$, where $p_T(x)$ is the parent of x in T for all $x \in T$. To choose the next node to insert, we look for the node $v \in V$ maximizing $in(v) + out(v)$, ignoring nodes of already-used colors, which takes $O(n)$ time (and could in practice often finish early if we search in decreasing order of one of these terms, and know an upper bound on the other). We then perform a single $O(k)$-time scan to find its optimal parent in the tree, and then make two further updates: First, for all $u \in V$, set $in(u) \leftarrow \max\{in(u), w(vu)\}$ and $out(u) \leftarrow out(u) + \max\{0, w(uv) - w(p_T(v), v)\}$. Second, for all $x \in V_T$, check whether the incoming edge yx (i.e., $y = p_T(x)$) can be improved by rerouting via v; if so, delete yx, insert vx and for all $u \in V$ such that $w(yx) \leq w(ux)$, set $out(u) \leftarrow \max\{0, out(u) - (w(vx) - w(yx))\}$. The second update, which dominates, needs $O(kn)$ time per inserted node, for $O(k^2 n)$ overall.

- The Top-down heuristic searches at most k times for the maximum weight edge leaving a node; since there are $O(n)$ such edges, the running time is $O(kn)$.
- For the Critical Path[1] heuristic, we need $O(m)$ time to compute the $S[u]$ values and to identify the path of maximum weight. This is repeated at most $k-1$ times, resulting in a total running time of $O(km)$. The same holds true for Critical Path[2]. For Critical Path[3] we can again maintain an $out(v)$ table that contains the score bonus we get for attaching a node in the intermediate tree as a child of v and deleting its previous incoming edge. After each insertion of an edge into the intermediate tree we have to perform the two update operations on out which takes $O(kn)$ time per insertion. In total we need $O(k^2 n + km)$ time to compute Critical Path[3]. In applications, k is very small and $n \ll m$, so the $O(km)$ part for calculating the critical paths requires most of the computation time.

3.2 Computing the k-best fragmentation trees exactly

Even if we do not trust the structural quality of the heuristic solution, the above heuristics allow us to speed up fragmentation tree computation: We first select a single candidate (molecular formula of the precursor) using one of the heuristics, then compute the optimal solution for this instance using an exact method [3, 12, 20]. In practice, this approach has two shortcomings: Even though certain heuristics show a very good performance in selecting the correct molecular formula (see below), this correct answer is not known to us in application; but we will observe that the computed fragmentation tree will often not be the optimum fragmentation tree, if we also consider other molecular formula candidates and corresponding instances.

Even worse, it is usually not sufficient in application to select a single best candidate using the heuristic, then re-compute the fragmentation tree for the corresponding instance. Instead, we usually want to know optimal fragmentation trees for the k best-scoring candidates. This is independent of whether results are reported to the user, who wants to use fragmentation tree structure to survey if computations and, hence, the assigned precursor molecular formula are trustworthy; or, if we perform some downstream computational analysis based on fragmentation tree structure, such as CSI:FingerID [6]. In particular for "large" metabolites with mass beyond 600 Dalton, this is necessary because neither the heuristics nor the exact method will always allow us to find the correct candidate; only by considering several candidates can we be sufficiently sure that the correct answer is present.

We propose the following heuristic to compute optimum fragmentation trees for the k-best molecular formula candidates: First, we compute heuristic solutions for all candidates, and order molecular formula candidates according to the heuristic score. Next, we compute

optimum fragmentation trees for the k best candidates; for small k, we can instead choose some fixed parameter, such as 10 candidates. We estimate the maximum $\Delta \geq 0$ of differences between the score of the optimum solution and the corresponding heuristic solution, using those candidates for which we know the exact solution. We now assume that the score difference is upper-bounded by Δ for all candidates. We continue to process candidates and compute optimum fragmentation trees from the sorted list, updating the k-best candidates and the corresponding score threshold; we stop computations when the heuristic score of a candidate plus Δ is smaller than the current score threshold. Clearly, our assumption made above may be violated for certain inputs, making this method a heuristic.

4 Data and Instances

To evaluate whether a heuristic is capable of ranking the correct molecular formula in the top position, we have to use reference data where the true compound structure is known for each MS/MS measurement. We use reference compounds from GNPS [19]; each reference compound is one instance, corresponding to several graphs (for the different molecular formula candidates of the precursor mass) we have to solve. We then filter instances: For example, we require a mass accuracy of 10 ppm (parts per million), and discard compounds where the precursor mass is missing or outside this mass range. All details can be found in [2]. This leaves us with 4 050 compounds, each of which is then transformed into typically many instances of the Maximum Colorful Subtree problem. Each reference compound resulted in between 1 and 21 748 candidate molecular formulas, with median 53 and average 263.8 candidates. To avoid proliferating running times, we consider only the 60 most intense peaks in a MS/MS spectrum that can be decomposed, which is again SIRIUS 4 default behavior. We fix the SIRIUS tree size parameter, which is usually adapted at runtime, at -0.5. In addition, we switch off SIRIUS's spectral recalibration.

5 Results

We applied all but the Critical Path heuristics using the RDS postprocessing. We do not evaluate the RDE postprocessing, as it is dominated by RDS (that is, the score is at least as good, in all cases) which, in turn, dominates solutions without postprocessing. Furthermore, both kinds of postprocessing are very fast in practice. For the Critical Path heuristics, RDS cannot improve a solution for variants 1 and 2; while it is in principle possible that it could improve a solution for variant 3, this is very unlikely, and to keep results of the Critical Path heuristics consistent, we disabled the RDS postprocessing for all 3 variants. All heuristics were implemented in Java 8. For the exact method, we use the Integer Linear Program (ILP) from [12] with the CPLEX solver 12.7.1 (IBM, `https://www.ibm.com/products/ilog-cplex-optimization-studio`).

First, we evaluated the power of the different heuristics to rank the correct answer (molecular formula) at the top position; see Fig. 3 (left). We also compared against the exact solution. We observe similar identification rates for the Critical Path heuristics, the Maximum heuristic and the exact method. To test whether this trend is true not only for the top rank, but also for the top k ranks, we also evaluated how often any method is capable to rank the correct answer in its top k, for varying k; see Fig. 3 (right). Identification rates differ much more strongly when varying k for one method than for different methods and one k; to ease interpretation, we normalize identification rates by subtracting the identification rate of the exact method. We see that all heuristics but for the three Critical Path variants result

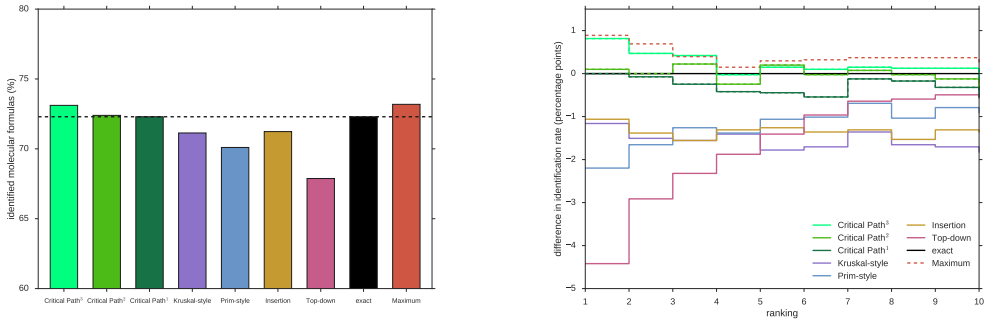


Figure 3 Performance evaluation, finding the correct molecular formula. Left: *Percentage* of compounds where the correct molecular formula received the highest score. Note the zoom of the y axis. Right: *Percentage point difference* against exact computations; how often is the correct answer part of the top k output of each method? Note that "Maximum" is one of the heuristics.

in inferior rankings, losing one or more percentage points for most ranks. In contrast, the Critical Path heuristics rank solutions with power comparable to the exact method, and the latter two variants often outperform the exact method. Somewhat surprisingly, the maximum over all heuristics performs even better than the best heuristic.

Second, we compared running times of the different methods; see Fig. 4. Running times were measured using a single thread on an Intel E5-2630v3 at 2.40 GHz with 64 GB RAM. The total running time for the exact methods over all instances is almost one month, underlining the importance of speeding up computations. But also note that solving all instances exactly requires only about 100-fold the time required for constructing the instance graphs. For each method, we sorted all instances by running time; we then determined how much time is required to solve the "easiest" $x\,\%$ of instances for that method, for each $0 \leq x \leq 100$. Generally, the ordering of instances is different for each method. For all methods, we observe that the "hardest" $5\,\%$ of the instances are responsible for most of the total running time; this has been observed before [2, 12]. In comparison to the exact method, all heuristics are very fast – at least two orders of magnitude faster. In particular, *each* heuristic is faster than the method for constructing the instance graphs; running all heuristics, as required for the Maximum heuristic, requires about the same time as the graph construction. Comparing heuristics' running times, we see that the Kruskal-style heuristic is slowest, and that the first variant of the Critical Path heuristic is faster in practice than variants 2 and 3, but not significantly.

Third, we compared the scores reached by the different heuristics to the scores of the exact solutions; see Fig. 5 (left). For each compound, we only considered the instance of the Maximum Colorful Subtree that corresponds to the true candidate molecular formula. We report scores relative to the exact solution (at $100\,\%$), and sorted instances with respect to this relative score. In the resulting plot, it is not obvious which of the heuristics "Insertion", "Kruskal-style" and "Critical Path^1" should be preferred. We see that Critical Path^3 heuristic and, hence, the maximum of all methods are able to compute (almost) optimal solutions for about $80\,\%$ of the instances. In turn, this means that even for these methods which perform extremely well in ranking the correct answer, we miss the optimal solution in about $20\,\%$ of the instances. In addition, we compared scores of the Critical Path^3 heuristic against the exact method in more detail, see Fig. 5 (right): We see that for instances where the heuristic does not find the optimal solution, the computed solution is only "slightly suboptimal" with

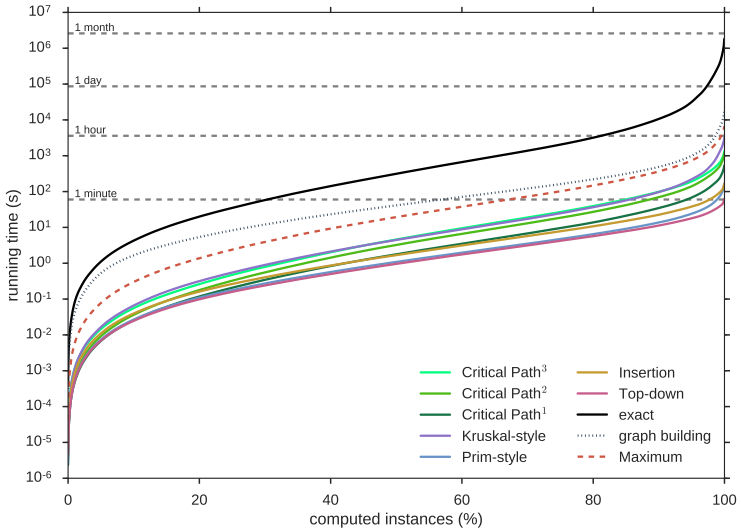


Figure 4 Running times of the different methods. One instances consists of all graphs generated for one compound in the dataset, considering all decompositions of the precursor mass. For each method, instances are sorted with respect to running time, and we report amortized running times. We also report running times for constructing the instance DAGs and for the exact method.

respect to its score. In fact, Pearson correlation between the two measures is +1.00.

Fourth, we evaluate the solution structure quality of the Critical Path^3 heuristic. Unfortunately, the "true fragmentation tree" cannot be determined experimentally [11]. To this end, we compare heuristic tree structures against tree structures computed using the exact method. For each compound, we restrict the comparison to the true candidate molecular formula; for other candidate molecular formulas, the optimal tree cannot possibly be the "'true fragmentation tree". See Fig. 6. For tree sizes, we observe rather large deviations between heuristic and optimal trees; in contrast, the overall distribution of tree sizes is highly similar. But if we compare tree structures, we observe much larger differences between the Critical Path^3 heuristic and the exact method: We measure structural similarity comparing either the set of node labels (fragments) or the set of edge labels (losses) of the two trees. We estimate the similarity of two (finite) sets A, B using the *Jaccard similarity coefficient* $J(A,B) = |A \cap B| \,/\, |A \cup B| \in [0,1]$. We observe that more than 20 % of the heuristic trees differ from the corresponding optimal tree; for at least 10 %, this difference is significant.

6 Conclusion

We have presented heuristics for the Maximum Colorful Subtree problem. Our evaluation shows that the Critical Path^3 heuristic is well-suited for choosing the correct candidate molecular formula, when applied to tandem mass spectrometry data of small molecules. Our evaluation sidesteps the catch-22 that we want to evaluate solutions based on structure and not score when, at the same time, the correct solution structure is not known. Even when the heuristic returns a suboptimal solution, the score is usually very close to the optimal score. Furthermore, the heuristics allow us to rank candidates, and to restrict computation of exact solutions to the top-ranked candidates (Sec. 3.2). In application, this combination resulted in significant speedups without sacrificing fragmentation tree quality.

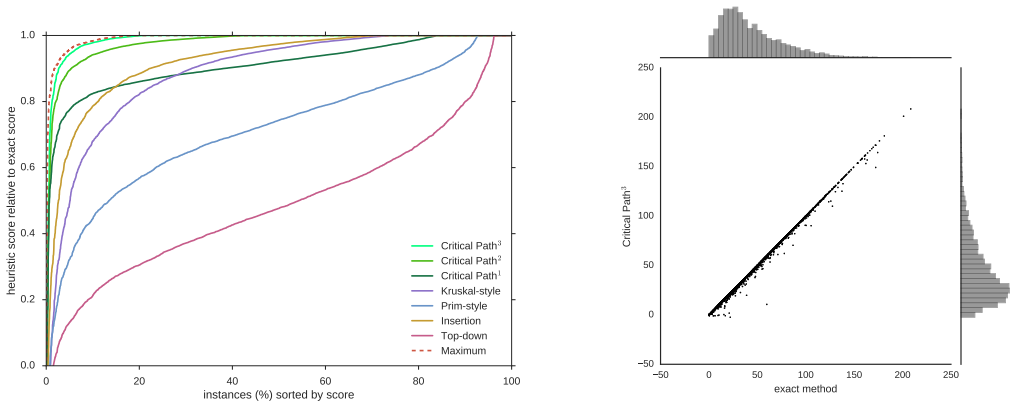


Figure 5 Left: Relative scores of the heuristics. For each Maximum Colorful Subtree instance, we consider the relative score in comparison to the exact method at 100 %. For each method, instances are sorted with respect to relative score. Right: Comparison of the score of the Critical Path^3 heuristic in comparison to the optimal score of the instance. For each compound, we consider only the true molecular formula candidate.

The structure of the heuristic tree deviates significantly from the optimal tree for more than 20 % of the instances. We therefore argue against using this tree for downstream analysis, such as machine learning [6, 15]. A back-of-the-envelope calculation indicates the problem: If we assume that 20 % of the heuristic trees are "structurally faulty", then a pairwise comparison of trees will result in 36 % tree pairs where at least one of the trees is "structurally faulty".

Building an instance DAG requires more time than running any of the presented heuristics. We conjecture that there is only limited potential for speeding up the graph building phase. To this end, whereas searching for better (and not significantly slower) heuristics is still a valid undertaking, faster heuristics are of little practical use. It is worth mentioning that computing exact solutions for the NP-hard Maximum Colorful Subtree problem takes only about 100-fold the time needed for constructing the graph instances; further speed-up is possible using data reductions and a stronger ILP formulation of the problem from [20].

Even elaborate heuristics for a bioinformatics problem, which are capable of finding solutions with objective function value very close to the optimum, can result in solutions which are structurally very dissimilar from the optimum structure. We showed that this is not only a theoretical possibility, but happens regularly for real-world instances. This underlines the importance of finding exact solutions for bioinformatics problems; the structure of solutions found by heuristic, including local search heuristics such as Markov chain Monte Carlo, may deviate significantly from the optimal solution.

Availability

The Critical Path^3 heuristic and the exact method are available through the SIRIUS 4 software (`https://bio.informatik.uni-jena.de/software/sirius/`) and also from GitHub (`https://github.com/boecker-lab/sirius`). Source code for all other heuristics will be made available upon request. Instances are available from `https://bio.informatik.uni-jena.de/data/`.

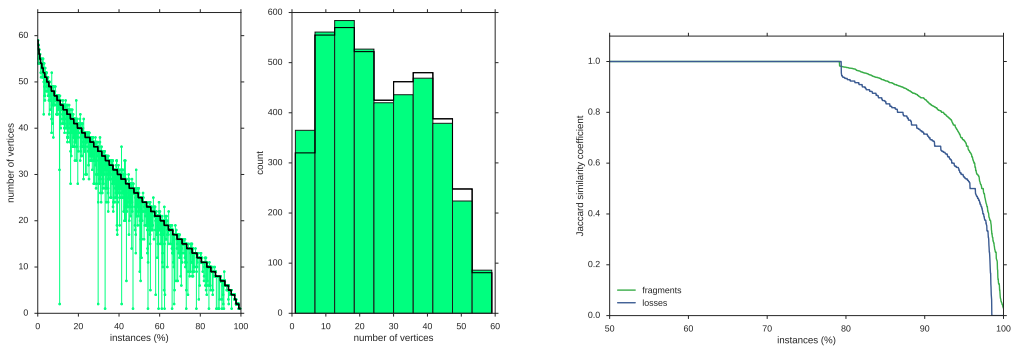


Figure 6 Left: Size of the fragmentation tree. Instances are sorted with respect to the size of the optimal fragmentation tree (black); green bars indicate the corresponding tree sizes for the Critical Path^3 heuristic. Middle: Distribution of tree sizes for the exact method (black) and the Critical Path^3 heuristic (green). Right: Comparison of the fragmentation tree structure, optimal tree vs. the tree computed by the Critical Path^3 heuristic. Note the zoom of the x axis. In all three cases, we consider only the true molecular formula candidate for each compound.

References

1 Alexander A Aksenov, Ricardo da Silva, Rob Knight, Norberto P Lopes, and Pieter C Dorrestein. Global chemical analysis of biology by mass spectrometry. *Nature Reviews Chemistry*, 1(7):0054, 2017.

2 Sebastian Böcker and Kai Dührkop. Fragmentation trees reloaded. *J Cheminform*, 8:5, 2016. doi:10.1186/s13321-016-0116-8.

3 Sebastian Böcker and Florian Rasche. Towards de novo identification of metabolites by analyzing tandem mass spectra. *Bioinformatics*, 24:I49–I55, 2008. Proc. of *European Conference on Computational Biology* (ECCB 2008). doi:10.1093/bioinformatics/btn270.

4 Ricardo R. da Silva, Pieter C. Dorrestein, and Robert A. Quinn. Illuminating the dark matter in metabolomics. *Proc Natl Acad Sci U S A*, 112(41):12549–12550, 2015. doi:10.1073/pnas.1516878112.

5 Riccardo Dondi, Guillaume Fertin, and Stéphane Vialette. Complexity issues in vertex-colored graph pattern matching. *J Discrete Algorithms*, 9(1):82–99, 2011. doi:doi:10.1016/j.jda.2010.09.002.

6 Kai Dührkop, Huibin Shen, Marvin Meusel, Juho Rousu, and Sebastian Böcker. Searching molecular structure databases with tandem mass spectra using CSI:FingerID. *Proc Natl Acad Sci U S A*, 112(41):12580–12585, 2015. doi:10.1073/pnas.1509788112.

7 Guillaume Fertin, Julien Fradin, and Géraldine Jean. Algorithmic aspects of the maximum colorful arborescence problem. In *Proc. of Theory and Applications of Models of Computation (TAMC 2017)*, volume 10185 of *Lect Notes Comput Sci*, pages 216–230, 2017. doi:10.1007/978-3-319-55911-7_16.

8 Alaa Khedr, Soad S Abd El-Hay, and Ahmed K Kammoun. Liquid chromatography-tandem mass spectrometric determination of propofol in rat serum and hair at attogram level after derivatization with 3-bromomethyl-propyphenazone. *Journal of pharmaceutical and biomedical analysis*, 134:195–202, 2017. doi:10.1016/j.jpba.2016.11.051.

9 Vincent Lacroix, Cristina G. Fernandes, and Marie-France Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans Comput Biology Bioinform*, 3(4):360–368, 2006.

10 Gary J. Patti, Oscar Yanes, and Gary Siuzdak. Metabolomics: The apogee of the omics trilogy. *Nat Rev Mol Cell Biol*, 13(4):263–269, 2012. doi:10.1038/nrm3314.

11 Florian Rasche, Aleš Svatoš, Ravi Kumar Maddula, Christoph Böttcher, and Sebastian Böcker. Computing fragmentation trees from tandem mass spectrometry data. *Anal Chem*, 83(4):1243–1251, 2011. doi:10.1021/ac101825k.

12 Imran Rauf, Florian Rasche, François Nicolas, and Sebastian Böcker. Finding maximum colorful subtrees in practice. *J Comput Biol*, 20(4):1–11, 2013. doi:10.1089/cmb.2012.0083.

13 Emma L Schymanski, Heinz P Singer, Jaroslav Slobodnik, Ildiko M Ipolyi, Peter Oswald, Martin Krauss, Tobias Schulze, Peter Haglund, Thomas Letzel, Sylvia Grosse, Nikolaos S Thomaidis, Anna Bletsou, Christian Zwiener, María Ibáñez, Tania Portolés, Ronald de Boer, Malcolm J Reid, Matthias Onghena, Uwe Kunkel, Wolfgang Schulz, Amélie Guillon, Naïke Noyon, Gaëla Leroy, Philippe Bados, Sara Bogialli, Draženka Stipaničev, Pawel Rostkowski, and Juliane Hollender. Non-target screening with high-resolution mass spectrometry: critical review using a collaborative trial on water analysis. *Analytical and bioanalytical chemistry*, 407:6237–6255, 2015. doi:10.1007/s00216-015-8681-7.

14 Emma Louise Schymanski, Christoph Ruttkies, Martin Krauss, Céline Brouard, Tobias Kind, Kai Dührkop, Felicity Ruth Allen, Arpana Vaniya, Dries Verdegem, Sebastian Böcker, Juho Rousu, Huibin Shen, Hiroshi Tsugawa, Tanvir Sajed, Oliver Fiehn, Bart Ghesquière, and Steffen Neumann. Critical Assessment of Small Molecule Identification 2016: Automated methods. *J Cheminf*, 9:22, 2017. doi:10.1186/s13321-017-0207-1.

15 Huibin Shen, Kai Dührkop, Sebastian Böcker, and Juho Rousu. Metabolite identification through multiple kernel learning on fragmentation trees. *Bioinformatics*, 30(12):i157–i164, 2014. Proc. of *Intelligent Systems for Molecular Biology* (ISMB 2014). doi:10.1093/bioinformatics/btu275.

16 Aleksandra Skirycz, Sylwia Kierszniowska, Michaël Méret, Lothar Willmitzer, and George Tzotzos. Medicinal bioprospecting of the amazon rainforest: A modern eldorado? *Trends in biotechnology*, 34:781–790, 2016. doi:10.1016/j.tibtech.2016.03.006.

17 Stephen E. Stein. Mass spectral reference libraries: An ever-expanding resource for chemical identification. *Anal Chem*, 84(17):7274–7282, 2012. doi:10.1021/ac301205z.

18 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J Comput System Sci*, 18(2):110–127, 1979.

19 Mingxun Wang, Jeremy J. Carver, Vanessa V. Phelan, Laura M. Sanchez, Neha Garg, Yao Peng, Don Duy Nguyen, Jeramie Watrous, Clifford A. Kapono, Tal Luzzatto-Knaan, Carla Porto, Amina Bouslimani, Alexey V. Melnik, Michael J. Meehan, Wei-Ting Liu, Max Crüsemann, Paul D. Boudreau, Eduardo Esquenazi, Mario Sandoval-Calderón, Roland D. Kersten, Laura A. Pace, Robert A. Quinn, Katherine R. Duncan, Cheng-Chih Hsu, Dimitrios J. Floros, Ronnie G. Gavilan, Karin Kleigrewe, Trent Northen, Rachel J. Dutton, Delphine Parrot, Erin E. Carlson, Bertrand Aigle, Charlotte F. Michelsen, Lars Jelsbak, Christian Sohlenkamp, Pavel Pevzner, Anna Edlund, Jeffrey McLean, Jörn Piel, Brian T. Murphy, Lena Gerwick, Chih-Chuang Liaw, Yu-Liang Yang, Hans-Ulrich Humpf, Maria Maansson, Robert A. Keyzers, Amy C. Sims, Andrew R. Johnson, Ashley M. Sidebottom, Brian E. Sedio, Andreas Klitgaard, Charles B. Larson, Cristopher A. Boya P, Daniel Torres-Mendoza, David J. Gonzalez, Denise B. Silva, Lucas M. Marques, Daniel P. Demarque, Egle Pociute, Ellis C. O'Neill, Enora Briand, Eric J N. Helfrich, Eve A. Granatosky, Evgenia Glukhov, Florian Ryffel, Hailey Houson, Hosein Mohimani, Jenan J. Kharbush, Yi Zeng, Julia A. Vorholt, Kenji L. Kurita, Pep Charusanti, Kerry L. McPhail, Kristian Fog Nielsen, Lisa Vuong, Maryam Elfeki, Matthew F. Traxler, Niclas Engene, Nobuhiro Koyama, Oliver B. Vining, Ralph Baric, Ricardo R. Silva, Samantha J. Mascuch, Sophie Tomasi, Stefan Jenkins, Venkat Macherla, Thomas Hoffman, Vinayak Agarwal, Philip G. Williams,

Jingqui Dai, Ram Neupane, Joshua Gurr, Andrés M C. Rodríguez, Anne Lamsa, Chen Zhang, Kathleen Dorrestein, Brendan M. Duggan, Jehad Almaliti, Pierre-Marie Allard, Prasad Phapale, Louis-Felix Nothias, Theodore Alexandrov, Marc Litaudon, Jean-Luc Wolfender, Jennifer E. Kyle, Thomas O. Metz, Tyler Peryea, Dac-Trung Nguyen, Danielle VanLeer, Paul Shinn, Ajit Jadhav, Rolf Müller, Katrina M. Waters, Wenyuan Shi, Xueting Liu, Lixin Zhang, Rob Knight, Paul R. Jensen, Bernhard Ø. Palsson, Kit Pogliano, Roger G. Linington, Marcelino Gutiérrez, Norberto P. Lopes, William H. Gerwick, Bradley S. Moore, Pieter C. Dorrestein, and Nuno Bandeira. Sharing and community curation of mass spectrometry data with Global Natural Products Social molecular networking. *Nat Biotechnol*, 34(8):828–837, 2016. doi:10.1038/nbt.3597.

20 W. Timothy J. White, Stephan Beyer, Kai Dührkop, Markus Chimani, and Sebastian Böcker. Speedy colorful subtrees. In *Proc. of Computing and Combinatorics Conference (COCOON 2015)*, volume 9198 of *Lect Notes Comput Sci*, pages 310–322. Springer, Berlin, 2015. doi:10.1007/978-3-319-21398-9_25.

21 David S Wishart. Emerging applications of metabolomics in drug discovery and precision medicine. *Nature reviews. Drug discovery*, 15:473–484, 2016. doi:10.1038/nrd.2016.32.

Parsimonious Migration History Problem: Complexity and Algorithms

Mohammed El-Kebir
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL
melkebir@illinois.edu
https://orcid.org/0000-0002-1468-2407

Abstract

In many evolutionary processes we observe extant taxa in different geographical or anatomical locations. To reconstruct the migration history from a given phylogenetic tree T, one can model locations using an additional character and apply parsimony criteria to assign a location to each internal vertex of T. The *migration criterion* assumes that migrations are independent events. This assumption does not hold for evolutionary processes where distinct taxa from different lineages comigrate from one location to another in a single event, as is the case in metastasis and in certain infectious diseases. To account for such cases, the *comigration criterion* was recently introduced, and used as an optimization criterion in the Parsimonious Migration History (PMH) problem. In this work, we show that PMH is NP-hard. In addition, we show that a variant of PMH is fixed parameter tractable (FPT) in the number of locations. On simulated instances of practical size, we demonstrate that our FPT algorithm outperforms a previous integer linear program in terms of running time.

2012 ACM Subject Classification Applied computing → Molecular evolution, Mathematics of computing → Trees, Mathematics of computing → Combinatorial optimization

Keywords and phrases Reconciliation, maximum parsimony, metastasis, infection, phylogenetics, phylogeography, fixed parameter tractability

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.24

1 Introduction

A phylogenetic tree, or phylogeny for short, models an evolutionary process such as those that arise in the development of cancer, species, pathogens and languages. In a character-based phylogeny, taxa are described by the same set of traits, where each trait is modeled by a single character with discrete states. Mathematically, a *character-based phylogeny* is a tree T whose vertices are taxa labeled by a vector of character states, assigning a single state to each character in each taxon. While the leaves of T correspond to extant taxa with observed character states, the internal vertices and edges of T are typically inferred algorithmically using different criteria such as maximum parsimony [4] or maximum likelihood [3].

In some evolutionary processes, extant taxa occur in different geographical or anatomical locations and one wishes to reconstruct the locations of ancestral taxa. Slatkin and Maddison [10] noted that locations can be modeled by an additional character and introduced the *migration criterion*. That is, given a phylogeny T with a location $\ell(u)$ assigned to each leaf u, the authors proposed to assign a location $\ell(v)$ to each internal vertex v of T such that the number $\mu(T, \ell)$ of migrations, i.e. edges (v, w) where $\ell(v) \neq \ell(w)$, is minimized (Fig. 1). Slatkin and Maddison [10] noted that this is an instance of the small phylogeny problem and can be solved in polynomial time [4, 9]. Later, McPherson et al. [7] used the migration criterion to study the migration history in metastatic ovarian cancers, where locations are distinct anatomical locations with metastasis that occur within the same patient.

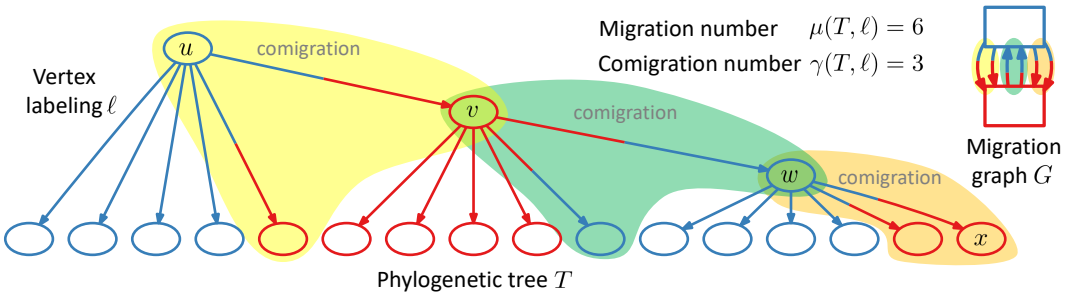


Figure 1 Migration histories are labelings of the vertices of a phylogenetic tree by locations. Vertex labeling ℓ assigns to each vertex u a location $\ell(u)$. A migration is an edge (u, v) where $\ell(u) \neq \ell(v)$. Here, vertex labeling ℓ of T incurs 6 migrations (bichromatic edges) and thus has migration number $\mu(T, \ell) = 6$. Although migrations (u, v) and (w, x) have identical source and target locations, they could not have happened simultaneously as taxon w is a descendant of taxon v. A comigration is a set of migrations between the same pair of locations that occur on distinct branches of the phylogenetic tree T. The comigration number is the smallest partition of migrations into comigrations. Here, vertex labeling ℓ of T has comigration number $\gamma(T, \ell) = 3$. The set of migrations determines the migration graph G, a multi-graph whose vertices are locations.

Underlying the migration criterion is the assumption that migrations are independent events. While this assumption is justified for evolutionary processes where shared location of taxa is uninformative, in certain evolutionary processes migrations of distinct taxa between the same locations are not independent events. For instance, in cancer, metastases may be seeded by groups of tumor cells from different clones (taxa) that migrate together in the bloodstream or lymphatics. Moreover, in certain infectious diseases, such as those caused by Hepatitis-B and C, HIV and Ebola, different strains of the pathogen can infect a person through a single transmission event. As such, the migration criterion does not always apply.

Recently, El-Kebir et al. [2] introduced the *comigration criterion*, where multiple migrations between the same pair of locations are counted as a single event (Fig. 1). The authors showed that the problem of assigning locations to ancestral taxa under the migration and comigration criteria is a multi-objective optimization problem, with tradeoffs between both criteria and the topology of the *migration graph*, a directed multi-graph capturing all migrations between locations (Fig. 2). This problem was called the Parsimonious Migration History (PMH) and was solved using an integer linear program (ILP). The hardness of PMH was an open problem.

In this work, we show that PMH is NP-hard. On the positive side, we show that when the migration graph is restricted to a tree, PMH is fixed parameter tractable (FPT) in the number of locations. Using simulated data of metastatic cancers, we show that for practical PMH instances with a small number of locations, the FPT algorithm outperforms the ILP in terms of running time.

2 Preliminaries

Let T be a tree rooted at vertex $r(T)$. We denote the edge set by $E(T)$, the vertex set by $V(T)$ and the leaf set by $L(T)$. We denote by T_v the subtree of T rooted at vertex v. We denote the parent vertex of a non-root vertex $v \neq r(T)$ by $\pi(v)$. We write $u \preceq_T v$ if and only if vertex u occurs on the unique path from $r(T)$ to v. Note that $\preceq_T$ is reflexive, i.e. $v \preceq_T v$ for all vertices $v \in V(T)$. We write $u \prec_T v$ if and only if $u \preceq_T v$ and $u \neq v$. We denote the lowest common ancestor of a vertex subset $U \subseteq V(T)$ by $\text{LCA}_T(U)$. We say that two vertices

u, v are *incomparable* or occur on *distinct branches* if and only if $u \not\preceq_T v$ and $v \not\preceq_T u$. Note that $u \not\preceq_T v$ and $v \not\preceq_T u$ if and only if $\text{LCA}_T(\{u, v\}) \notin \{u, v\}$.

A *phylogenetic tree* T is a rooted tree, whose leaves correspond to taxa that are labeled by different locations. We denote the set of locations by Σ. Since the leaf set $L(T)$ is composed of taxa observed at the present time, we know their locations and are thus given the function $\hat{\ell} : L(T) \to \Sigma$, assigning location $\hat{\ell}(v)$ to each leaf $v \in L(T)$. We use $\hat{\ell}(L(T_u))$ where $u \in V(T)$ as a shorthand for $\{\hat{\ell}(v) \mid v \in L(T_u)\}$. Typically, the edges of a phylogenetic tree are labeled by mutations. As the problem that we consider does not use mutations, they are omitted.

3 Problem Statement

Given a phylogenetic tree T with leaf labeling $\hat{\ell} : L(T) \to \Sigma$, the task is to reconstruct the migration history by inferring the location of origin of each ancestral taxon. In other words, we wish to extend the given leaf labeling $\hat{\ell}$ to a *vertex labeling* $\ell : V(T) \to \Sigma$ such that $\ell(v) = \hat{\ell}(v)$ for each leaf $v \in L(T)$. To distinguish different vertex labelings ℓ of a phylogenetic tree T, we use two different optimization criteria.

The first criterion was introduced by Slatkin and Maddison [10] and considers migrations, which are defined as follows.

▶ **Definition 1** ([2, 10]). A *migration* is an edge $(u, v) \in E(T)$ that connects two vertices with different locations, i.e. $\ell(u) \neq \ell(v)$.

The *migration number* $\mu(T, \ell)$ is the number of migrations incurred by ℓ. Slatkin and Maddison [10] noted that the problem of finding a vertex labeling with minimum migration number is an instance of the small phylogeny maximum parsimony problem with a single multi-state character (with states Σ). As such, this problem can be solved in polynomial time using either the Fitch [4] or the Sankoff algorithm [9].

The second criterion, which was recently introduced in [2], allows for the simultaneous migration, or *comigration*, of individuals from different (ancestral) taxa between the same locations. This criterion is applicable to evolutionary processes where locations are seeded by individuals from distinct taxa through a single event. Two migrations $(u, v) \neq (w, x)$, where $v \preceq_T w$, between the same pair of locations, i.e. $\ell(u) = \ell(w)$ and $\ell(v) = \ell(x)$, could never have occurred simultaneously. This is because taxon w is a descendant of taxon v, and thus migration (u, v) must have occurred prior to migration (w, x) (Fig. 1). To account for such scenarios, we define comigrations as follows.

▶ **Definition 2** ([2]). A *comigration* is a subset X of pairwise incomparable migrations between the same locations. That is, for all distinct pairs $(u, v), (u', v') \in X$ of migrations it holds that (i) $\ell(u) \neq \ell(v)$, (ii) $\ell(u) = \ell(u')$, (iii) $\ell(v) = \ell(v')$, (iv) $v \not\preceq_T v'$ and (v) $v' \not\preceq_T v$.

Employing the principle of parsimony, we define the *comigration number* $\gamma(T, \ell)$ as the smallest partition of migrations into comigrations [2]. As shown in [2], we have that

$$\gamma(T, \ell) = \sum_{s,t \in \Sigma \,:\, s \neq t} \gamma(T, \ell, s, t) \tag{1}$$

where $\gamma(T, \ell, s, t)$ is the maximum number of edges $(u, v) \in E(T)$ with $\ell(u) = s$ and $\ell(v) = t$ that are on the same path of T starting from the root $r(T)$.

The set of migrations incurred by a vertex labeling ℓ of T determines the *migration graph* G. More formally, the vertices of G are locations, and there is a directed edge (s, t) for each edge (u, v) in T where (i) $s \neq t$, (ii) $\ell(u) = s$ and (iii) $\ell(v) = t$. The migration

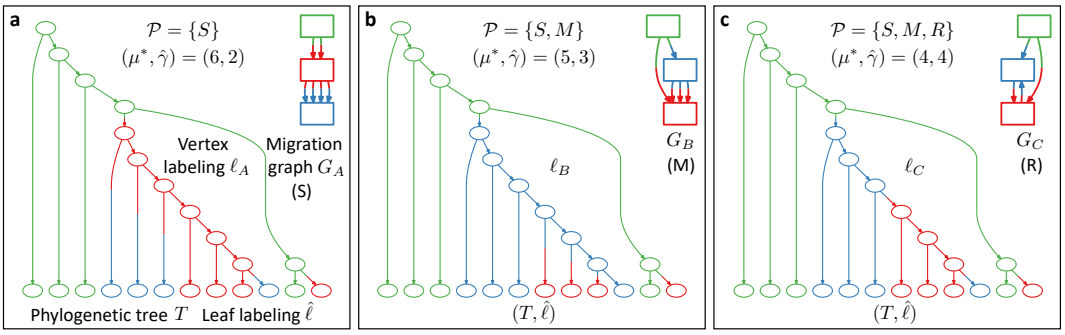


Figure 2 The Parsimonious Migration History (PMH) problem is a constrained multi-objective optimization problem, with tradeoffs between the migration number, comigration number and the migration pattern. Given a phylogenetic tree T whose leaves are labeled by locations via $\hat{\ell}$ and a set $\mathcal{P}$ of allowed migration patterns, the task is to extend $\hat{\ell}$ to a vertex labeling ℓ such that: (i) the resulting migration graph G adheres to $\mathcal{P}$, (ii) ℓ has the minimum migration number $\mu^*(T)$ and (iii) subsequently smallest comigration number $\hat{\gamma}(T)$. (a) In the most restrictive case, $\mathcal{P} = \{S\}$, the migration graph is required to be a tree, yielding vertex labeling ℓ_A with $\mu(T, \ell_A) = 6$ and $\gamma(T, \ell_A) = 2$. The resulting migration graph G_A has a single-source seeding (S) pattern. (b) In the case $\mathcal{P} = \{S, M\}$, the migration graph is required to not contain a directed cycle, yielding vertex labeling ℓ_B with $\mu(T, \ell_B) = 5$, $\gamma(T, \ell_B) = 3$, and migration graph G_B with a multi-source seeding (M) pattern. (c) In the case $\mathcal{P} = \{S, M, R\}$, the migration graph is left unrestricted, yielding vertex labeling ℓ_C with minimum migration number $\mu(T, \ell_C) = 4$, smallest comigration number $\gamma(T, \ell_C) = 4$, and migration graph G_C with a reseeding (R) pattern.

graph G is a multi-graph because there may exist multiple directed edges between the same pair of locations. Different topologies of G correspond to different *migration patterns*. We distinguish three migration patterns: (i) in *single-source seeding* (S), the migration graph G is a tree (Fig. 2a); (ii) in *multi-source seeding* (M), some locations are the target of migrations from distinct source locations but G itself does not have a directed cycle (Fig. 2b); and (iii) in *reseeding* (R), G has a directed cycle (Fig. 2c).

There are tradeoffs between the migration number, the comigration number and the migration pattern, as shown in [2] and briefly reviewed in the following. Let T be a phylogenetic tree whose leaves are labeled by $m = |\Sigma|$ locations. Any vertex labeling ℓ of T has migration number $\mu(T, \ell) \geq m - 1$ and comigration number $\gamma(T, \ell) \geq m - 1$. While there may not exist a vertex labeling ℓ with migration number $\mu(T, \ell) = m - 1$, there always exists a vertex labeling ℓ with comigration number $\gamma(T, \ell) = m - 1$ for any leaf-labeled tree T. For instance, a labeling ℓ that assigns the same location to all internal vertices has comigration number $\gamma(T, \ell) = m - 1$. Moreover, a vertex labeling ℓ incurs a single-source seeding (S) pattern if and only if $\gamma(T, \ell) = m - 1$. Let $\mu^*(T) = \min_\ell \mu(T, \ell)$ be the minimum migration number of T. In general, there may not exist a vertex labeling ℓ of T that simultaneously achieves the minimum migration number $\mu(T, \ell) = \mu^*(T)$ and minimum comigration number $\gamma(T, \ell) = m - 1$ (Fig. 2). To examine these tradeoffs, El-Kebir et al. [2] introduced the following constrained multi-objective optimization problem that considered three different sets $\mathcal{P}$ of allowed migration patterns: (i) $\mathcal{P} = \{S\}$, requiring the migration graph G to be an S pattern (Fig. 2a); (ii) $\mathcal{P} = \{S, M\}$, requiring the migration graph to be either an S or M pattern (Fig. 2b); (iii) $\mathcal{P} = \{S, M, R\}$ meaning that G is unrestricted (Fig. 2c).

▶ **Problem 1** (Parsimonious Migration History (PMH) [2]). *Given a phylogenetic tree T with leaf labeling $\hat{\ell} : L(T) \to \Sigma$ and a set $\mathcal{P}$ of allowed migration patterns, find a vertex labeling ℓ such that: (i) $\ell(v) = \hat{\ell}(v)$ for each leaf $v \in L(T)$; (ii) ℓ has the minimum migration*

number $\mu(T,\ell) = \mu^*(T) = \min_{\ell'} \mu(T,\ell')$ *and subsequently the smallest comigration number* $\gamma(T,\ell) = \hat{\gamma}(T) = \min_{\ell':\mu(T,\ell')=\mu^*(T)} \gamma(T,\ell')$*; and (iii) the resulting migration graph* G *is a tree if* $\mathcal{P} = \{S\}$*, a directed acyclic graph if* $\mathcal{P} = \{S, M\}$ *or unrestricted if* $\mathcal{P} = \{S, M, R\}$*.*

4 Results

We have the following two results for the case where the migration graph G is restricted to a tree (i.e. $\mathcal{P} = \{S\}$).

▶ **Theorem 3.** PMH *is NP-hard when* $\mathcal{P} = \{S\}$*.*

▶ **Theorem 4.** PMH *is fixed parameter tractable in* $|\Sigma|$ *when* $\mathcal{P} = \{S\}$*.*

Both theorems rely on the following important proposition that we prove first.

▶ **Proposition 5.** *Let* T *be a phylogenetic tree, and let* ℓ *be a vertex labeling of* T *such that the migration graph* G *is a tree. Then,* $\ell(u) \preceq_G \text{LCA}_G(\hat{\ell}(L(T_u)))$ *for any vertex* $u \in V(T)$*.*

Proof. Suppose for a contradiction that there exists a vertex labeling ℓ of T such that $\ell(u) \not\preceq_G \text{LCA}_G(\hat{\ell}(L(T_u)))$ for a vertex u of T. For brevity, we define $s = \text{LCA}_G(\hat{\ell}(L(T_u)))$. By our premise, there exists a leaf $v \in L(T_u)$ such that $\hat{\ell}(v) \notin V(G_{\ell(u)})$. By definition of LCA_G, there exists a unique path from s to $\hat{\ell}(v)$ in G. Moreover, since v is reachable from u in T, there exists a path from $\ell(u)$ to $\hat{\ell}(v)$ in G. We distinguish two cases. First, $\hat{\ell}(v) \preceq_G \pi(\ell(u))$. This case contradicts that G is a tree, as there would be a directed cycle between $\ell(u)$ and $\hat{\ell}(v)$ in G. Second, $\hat{\ell}(v)$ and $\ell(u)$ are incomparable in G. Thus, the path from s to $\hat{\ell}(v)$ does not contain $\ell(u)$. This, however, contradicts that there exists a path from $\ell(u)$ to $\hat{\ell}(v)$. ◀

4.1 NP-hardness

We show NP-hardness of the PMH problem in the case where $\mathcal{P} = \{S\}$ by reduction from 3-Satisfiability (3-SAT). In 3-SAT, we are given a Boolean formula $\phi = \bigwedge_{i=1}^{k} (y_{i,1} \vee y_{i,2} \vee y_{i,3})$ in 3-conjunctive normal form (3-CNF) with n variables and k clauses, and wish to decide whether there exists a truth assignment $\theta : [n] \to \{0,1\}$ that satisfies all the clauses of ϕ. We define $\psi(y_{i,j}) = 1$ if literal $y_{i,j}$ is of the form x, and define $\psi(y_{i,j}) = 0$ if literal $y_{i,j}$ is of the form $\neg x$. Truth assignment θ satisfies clause $(y_{i,1} \vee y_{i,2} \vee y_{i,3})$ provided there exists a $j \in \{1,2,3\}$ such that $\theta(x) = \psi(y_{i,j})$ where x is the variable corresponding to literal $y_{i,j}$. Without loss of generality, we may assume that each clause of ϕ consists of three distinct variables. We denote the n variables of ϕ by $x_1, \ldots, x_n$ and the k clauses of ϕ by $c_1, \ldots, c_k$. 3-SAT is among the 21 problems proven to be NP-hard by Karp [6].

We construct a tree T with location set $\Sigma = \{\bot, x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n, c_1, \ldots, c_k\}$ such that the migration graph G^* of any optimal vertex labeling ℓ^* models a truth assignment θ. More specifically, we want to ensure that: (i) ℓ^* labels the root of T by $\bot$; (ii) for each variable x of ϕ, either $\{(\bot, x), (x, \neg x)\} \subseteq E(G^*)$ if $\theta(x) = 1$, or $\{(\bot, \neg x), (\neg x, x)\} \subseteq E(G^*)$ if $\theta(x) = 0$; and (iii) θ is satisfiable if and only if ℓ^* encodes a truth assignment that satisfies all clauses of θ. We accomplish these three requirements using three types of gadgets that form subtrees of T (Fig. 3a): (i) n variable gadgets $T[x_1], \ldots, T[x_n]$, each with $2B$ leaves (Fig. 3b); (ii) a single root gadget $T[\bot]$ with A leaves (Fig. 3c); and (iii) $3k$ clause literal gadgets $T[y_{1,1}], \ldots, T[y_{k,3}]$, each with 9 leaves (Fig. 3d). Fig. 4 shows an example reduction.

Let ℓ^* be an optimal vertex labeling of T under the restriction $\mathcal{P} = \{S\}$, and let G^* be the resulting migration graph. By setting $B > 10k + 1$ and $A > 2Bn + 27k$, we accomplish the first two requirements, as shown by the following two lemmas.

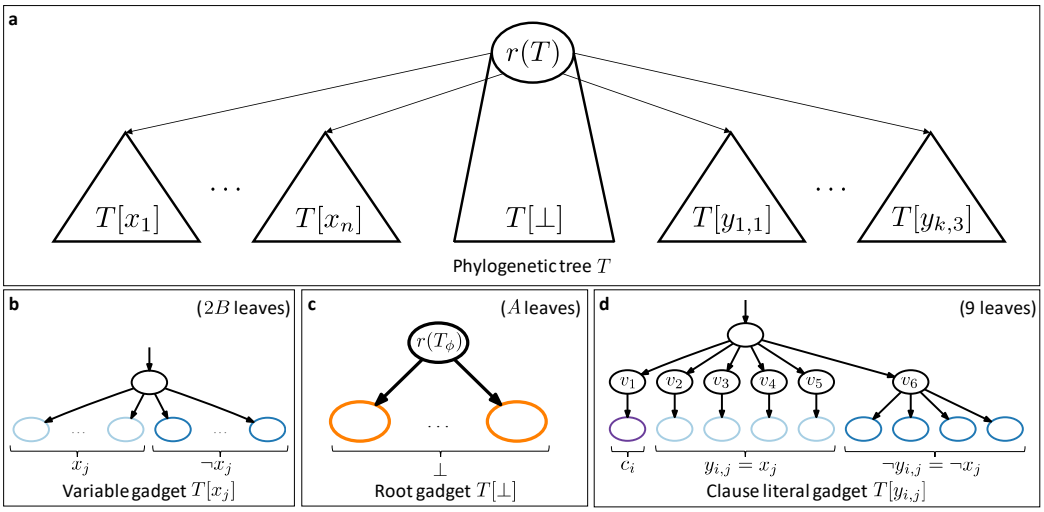


Figure 3 Given a Boolean formula $\phi = \bigwedge_{i=1}^{k}(y_{i,1} \vee y_{i,2} \vee y_{i,3})$ in 3-conjunctive normal form, we construct the phylogenetic tree T composed of three types of gadgets. (a) Tree T has n variable gadgets (panel b), a single root gadget (panel c) and $3k$ clause literal gadgets (panel d). The location set Σ contains a location corresponding to a positive and negative literal of each variable, a location corresponding to each clause and a special location $\bot$. That is, $\Sigma = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n, c_1, \ldots, c_k, \bot\}$. (b) For each variable x_j, the corresponding variable gadget $T[x_j]$ is composed of $2B$ leaves where $B = 10k + 2$. This gadget enforces that either $(x_j, \neg x_j) \in E(G^*)$ or $(\neg x_j, x_j) \in E(G^*)$. (c) The root gadget $T[\bot]$ is composed of $A = 2Bn + 27k + 1$ leaves and enforces that $\ell^*(r(T)) = \bot$. (d) For each literal $y_{i,j}$, the corresponding clause literal gadget $T[y_{i,j}]$ has 9 leaves, and links variables to clauses. We refer to Fig. 4 for an example.

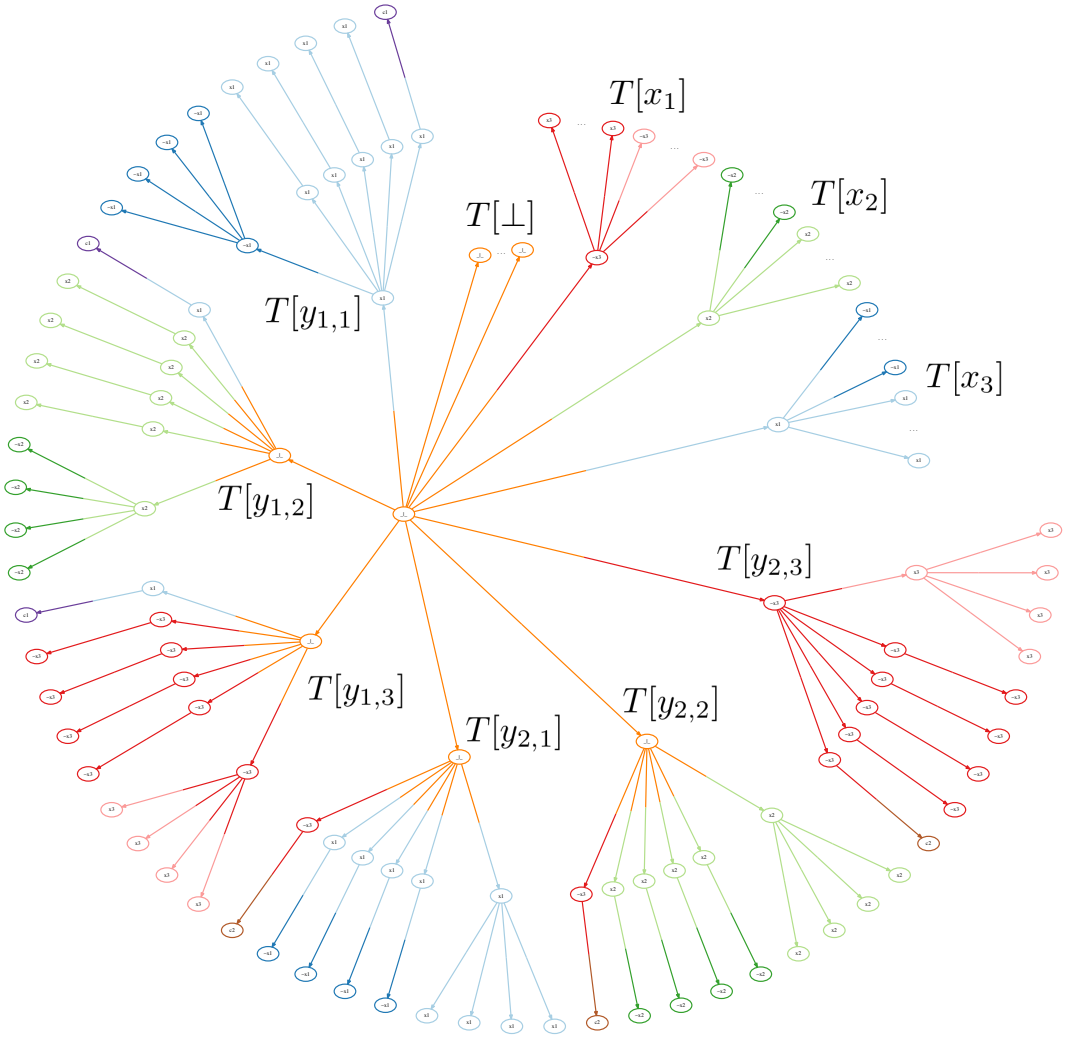
▶ **Lemma 6.** *The root vertex $r(T)$ has location $\ell^*(r(T)) = \bot$.*

Proof. Suppose for a contradiction that $\ell^*(r(T)) \neq \bot$. Thus, we have that each of the incoming edges to the leaves of the subtree $T[\bot]$ is a migration. Hence, $\mu(T, \ell^*) \geq A > 2Bn + 27k$. Consider the vertex labeling ℓ of T where $\ell(v) = \bot$ for each vertex v. Observe that the resulting migration graph of ℓ has an S pattern. Only the leaves of subtree $T[\bot]$ are labeled by $\bot$; the remaining $2Bn + 27k$ have leaf labels that differ from $\bot$. We thus have $\mu(T, \ell) = 2Bn + 27k$. Therefore, $\mu(T, \ell) < \mu(T, \ell^*)$, which contradicts the premise that ℓ^* is an optimal vertex labeling. Hence, $\ell^*(r(T)) = \bot$. ◀

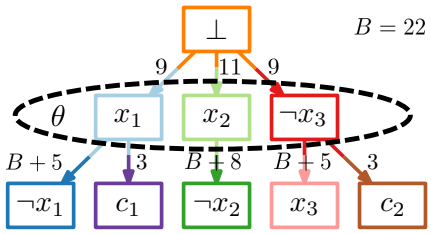
▶ **Lemma 7.** *For all variables x, $\{(\bot, x), (x, \neg x)\} \subseteq E(G^*)$ or $\{(\bot, \neg x), (\neg x, x)\} \subseteq E(G^*)$.*

Proof. Suppose for a contradiction that there exists a variable x of ϕ such that neither $\{(\bot, x), (x, \neg x)\} \subseteq E(G^*)$ nor $\{(\bot, \neg x), (\neg x, x)\} \subseteq E(G^*)$. Let $T[x]$ be the subtree of T corresponding to the variable gadget of x. Let r_x be the root vertex of $T[x]$. By Lemma 6 and the premise, we have that $\ell^*(r_x) \notin \{x, \neg x\}$. Therefore, the $2B$ edges incoming to the leaves of $T[x]$ are migrations in ℓ^*.

We construct a vertex labeling ℓ with fewer migrations than ℓ^*. Intuitively, vertex labeling ℓ corresponds to a truth assignment θ where $\theta(x) = 1$ for all variables x. Initially, we set $\ell = \ell^*$. Next, we set $\ell(r_x) = x$. By definition of 3-SAT, each clause c_i of ϕ contains at most one literal $y_{i,j}$ of the form x or $\neg x$. Let c_i be such a clause with literal $y_{i,j}$ of the form x or $\neg x$. Let $T[y_{i,j}]$ be the subtree of T corresponding to the clause literal gadget of $y_{i,j}$, and let $r_{i,j}$ be the root vertex of $T[y_{i,j}]$. We set $\ell(r_{i,j}) = \bot$, $\ell(v_1) = \bot$, $\ell(v_2) = \ell(v_3) = \ell(v_4) = \ell(v_5) = x$ and $\ell(v_6) = x$ (where $v_1, \ldots, v_6$ are defined in Fig. 3d). We distinguish two cases. First,



(a) Phylogenetic tree T and optimal vertex labeling ℓ^*.



(b) Migration graph G^*.

Figure 4 Example reduction. Consider the Boolean formula $\phi = (y_{1,1} \vee y_{1,2} \vee y_{1,3}) \wedge (y_{2,1} \vee y_{2,2} \vee y_{2,3}) = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ with $k = 2$ clauses and $n = 3$ variables. Truth assignment $\theta(x_1) = 1, \theta(x_2) = 1, \theta(x_3) = 0$ satisfies ϕ. (a) The corresponding tree T has location set $\Sigma = \{x_1, x_2, x_3, \neg x_1, \neg x_2, \neg x_3, c_1, c_2, \bot\}$. The variable gadgets $T[x_1], T[x_2], T[x_3]$ each have $2B = 2(10k+2) = 44$ leaves. The root gadget $T[\bot]$ has $A = 2Bn+27k+1 = 187$ leaves. The variable literal gadgets $T[y_{1,1}], \ldots, T[y_{2,3}]$ each have 9 leaves. The shown vertex labeling ℓ^* corresponds to truth assignment θ and has migration number $\mu(T, \ell^*) = (B+1)n + 25k = (22+1) \cdot 3 + 25 \cdot 2 = 119$. Thus, by Lemma 10, the truth assignment θ encoded by ℓ^* satisfies ϕ. (b) The corresponding migration graph G^*, where each edge is labeled by its multiplicity incurred by ℓ^*.

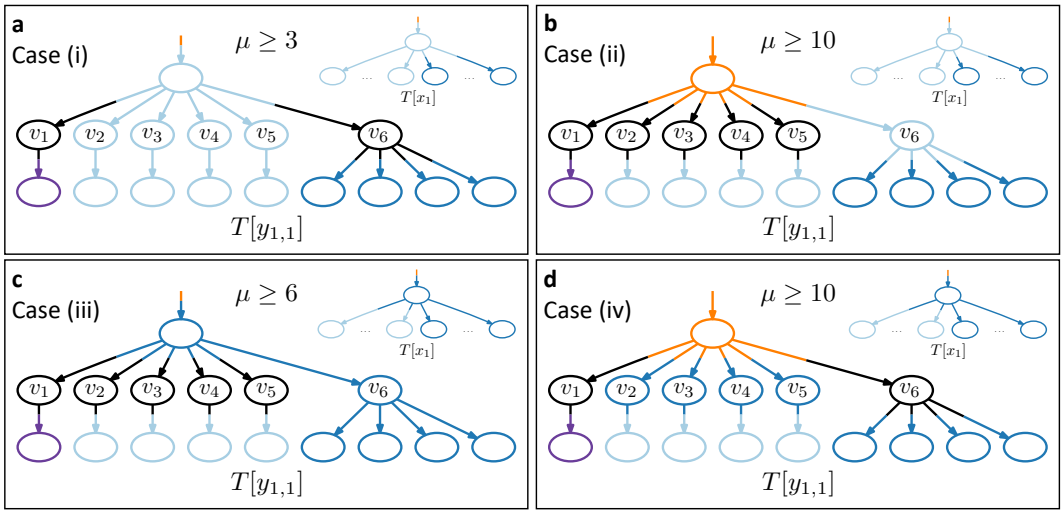


Figure 5 A clause literal gadget $T[y_{i,j}]$ imposes four different sets of constraints on optimal vertex labelings ℓ^*, as shown in Lemma 8. The figure reproduces clause literal gadget $T[y_{1,1}]$, where $y_{1,1} = x_1$, depicted in Fig. 4a. (a) If $\ell^*(r(T[x_1])) = x_1$ and $\ell^*(r(T[y_{1,1}])) = x_1$, vertices $v_2, \ldots, v_5$ must be labeled by x_1 and the migration number is at least 3. (b) If $\ell^*(r(T[x_1])) = x_1$ and $\ell^*(r(T[y_{1,1}])) = \bot$, vertex v_6 must be labeled by x_1 and the migration number is at least 10. (c) If $\ell^*(r(T[x_1])) = \neg x_1$ and $\ell^*(r(T[y_{1,1}])) = \neg x_1$, vertex v_6 must be labeled by $\neg x_1$ and the migration number is at least 6. (d) If $\ell^*(r(T[x_1])) = \neg x_1$ and $\ell^*(r(T[y_{1,1}])) = \bot$, vertices $v_2, \ldots, v_5$ must be labeled by $\neg x_1$ and the migration number is at least 10.

$y_{i,j} = x$. In this case, the outgoing edge of v_1 is a migration. The incoming edges of v_2, v_3, v_4, v_5, v_6 are migrations. The four outgoing edges of v_6 are migrations. Thus, the number of migrations incurred by ℓ in $T[y_{i,j}]$ is 10. Second, $y_{i,j} = \neg x$. In this case, the outgoing edge of v_1 is a migration. The incoming edges of v_2, v_3, v_4, v_5, v_6 are migrations. The outgoing edges of v_2, v_3, v_4, v_5 are migrations. Thus, the number of migrations incurred by ℓ in $T[y_{i,j}]$ is 10.

In both cases, we have that $\mu(T[x], \ell^*) > 2B$. On the other hand, $\mu(T[x], \ell) = B + 1$, and $\mu(T[y_{i,j}], \ell) = 10$ for each literal $y_{i,j}$ of the form $\{x, \neg x\}$. As there at most k literals of the form $\{x, \neg x\}$, we have that there are at most $B + 1 + 10k$ migrations incurred by ℓ in subtree $T[x]$ and the corresponding clause literal subtrees $T[y_{i,j}]$. Since $B > 1 + 10k$, we have that $2B > B + 1 + 10k$. Thus,

$$\mu(T[x], \ell^*) > 2B > B + 1 + 10k > \mu(T[x], \ell) + \sum_{y_{i,j} \text{ of form } \{x, \neg x\}} \mu(T[y_{i,j}], \ell).$$

Compared to ℓ^*, the new vertex labeling ℓ differs in subtree $T[x]$ and in at most k subtrees $T[y_{i,j}]$. Hence, $\mu(T, \ell^*) > \mu(T, \ell)$, yielding a contradiction. The lemma now follows. ◀

Thus, our reduction enables us to encode a truth assignment. We now focus on the third requirement that links literals to clauses. We start by showing that the clause literal gadget, when combined with the variable and root gadgets, imposes very specific constraints on ℓ^*.

▶ **Lemma 8.** *Let x be a variable of ϕ and let $y_{i,j}$ be a literal corresponding to x. Then, one of the following four cases must hold, where $C = \{c_i \mid i \in [k]\}$ and $X = \{\ell^*(r(T[x_i])) \mid i \in [n]\}$.*

case	$\ell^*(r(T[x]))$	$\ell^*(r(T[y_{i,j}]))$	$\ell^*(v_1)$	$\ell^*(v_2), \ldots, \ell^*(v_5)$	$\ell^*(v_6)$
(i)	$y_{i,j}$	$y_{i,j}$	$\{y_{i,j}, \neg y_{i,j}\} \cup C$	$y_{i,j}$	$\{y_{i,j}, \neg y_{i,j}\}$
(ii)	$y_{i,j}$	$\bot$	$\{\bot\} \cup C \cup X$	$\{\bot, y_{i,j}\}$	$y_{i,j}$
(iii)	$\neg y_{i,j}$	$\neg y_{i,j}$	$\{y_{i,j}, \neg y_{i,j}\} \cup C$	$\{y_{i,j}, \neg y_{i,j}\}$	$\neg y_{i,j}$
(iv)	$\neg y_{i,j}$	$\bot$	$\{\bot\} \cup C \cup X$	$\neg y_{i,j}$	$\{\bot, \neg y_{i,j}\}$

Proof. The lemma follows by case analysis, with four cases that each correspond to a unique case in the above table. First, it holds that $\ell^*(r(T[x])) \in \{y_{i,j}, \neg y_{i,j}\}$ by Lemma 7. Thus, we distinguish two cases.

1. $\ell^*(r(T[x])) = y_{i,j}$: By construction, $T[y_{i,j}]$ has leaves labeled by $y_{i,j}$ and $\neg y_{i,j}$. Hence, by Proposition 5, Lemma 6 and the fact that $\ell^*(r(T[x])) = y_{i,j}$, we have that $\ell^*(r(T[y_{i,j}])) \in \{y_{i,j}, \bot\}$. Thus, we distinguish two additional subcases.
   a. $\ell^*(r(T[y_{i,j}])) = y_{i,j}$: Vertices v_2, v_3, v_4, v_5 must be labeled by $y_{i,j}$ as their parent $r(T[y_{i,j}])$ is labeled by $y_{i,j}$ and they themselves are the parents of leaves labeled by $y_{i,j}$. The four children of v_6 are leaves labeled by $\neg y_{i,j}$, and the parent $r(T[y_{i,j}])$ of v_6 is labeled $y_{i,j}$. Thus, v_6 must be labeled by either $y_{i,j}$ or $\neg y_{i,j}$. Vertex v_1 is the child of $r(T[y_{i,j}])$ labeled by $y_{i,j}$, and thus the only literals that can label v_1 are $y_{i,j}$ and $\neg y_{i,j}$. In addition, v_1 can be labeled by clauses $c_1, \ldots, c_k$. This subcase corresponds to case (i) of the above table, and is depicted in Fig. 5a.
   b. $\ell^*(r(T[y_{i,j}])) = \bot$: Vertex v_6 must be labeled by $y_{i,j}$ as its parent $r(T[y_{i,j}])$ is labeled by $\bot$ and its four children are leaves labeled by $\neg y_{i,j}$. Vertices v_2, v_3, v_4, v_5 must each be labeled by either $y_{i,j}$ or $\bot$ as their parent $r(T[y_{i,j}])$ is labeled by $\bot$ and their children are leaves labeled by $y_{i,j}$. As vertex v_1 is the child of $r(T[y_{i,j}])$ and its only child a leaf labeled by c_i, we have that v_1 can be labeled by $\bot$, all clauses $C = \{c_1, \ldots, c_k\}$ and all active literals X. This subcase corresponds to case (ii) of the above table, and is depicted in Fig. 5b.
2. $\ell^*(r(T[x])) = \neg y_{i,j}$: By construction, $T[y_{i,j}]$ has leaves labeled by $y_{i,j}$ and $\neg y_{i,j}$. Hence, by Proposition 5, Lemma 6 and the fact that $\ell^*(r(T[x])) = \neg y_{i,j}$, we have that $\ell^*(r(T[y_{i,j}])) \in \{\neg y_{i,j}, \bot\}$. Thus, we distinguish two additional subcases.
   a. $\ell^*(r(T[y_{i,j}])) = \neg y_{i,j}$: Vertex v_6 must be labeled by $\neg y_{i,j}$ as its parent $r(T[y_{i,j}])$ is labeled by $\neg y_{i,j}$ and its four children are leaves labeled by $\neg y_{i,j}$. Vertices v_2, v_3, v_4, v_5 must each be labeled by either $\neg y_{i,j}$ or $y_{i,j}$ as their parent $r(T[y_{i,j}])$ is labeled by $\neg y_{i,j}$ and their children are leaves labeled by $\neg y_{i,j}$. Vertex v_1 is the child of $r(T[y_{i,j}])$ labeled by $y_{i,j}$, and thus the only literals that can label v_1 are $y_{i,j}$ and $\neg y_{i,j}$. In addition, v_1 can be labeled by clauses $c_1, \ldots, c_k$. This subcase corresponds to case (iii) of the above table, and is depicted in Fig. 5c.
   b. $\ell^*(r(T[y_{i,j}])) = \bot$: Vertices v_2, v_3, v_4, v_5 must each be labeled by $\neg y_{i,j}$ as their parent $r(T[y_{i,j}])$ is labeled by $\bot$ and their children are leaves labeled by $y_{i,j}$. Vertex v_6 must be labeled by either $\bot$ or $\neg y_{i,j}$, as its parent $r(T[y_{i,j}])$ is labeled by $\bot$ and its four children are leaves labeled by $\neg y_{i,j}$. As vertex v_1 is the child of $r(T[y_{i,j}])$ and its only child a leaf labeled by c_i, we have that v_1 can be labeled by $\bot$, all clauses $C = \{c_1, \ldots, c_k\}$ and all active literals X. This subcase corresponds to case (iv) of the above table, and is depicted in Fig. 5d. ◀

Next, we prove a lower bound on the minimum migration number of T given $\mathcal{P} = \{S\}$.

▶ **Lemma 9.** *It holds that* $\mu(T, \ell^*) \geq (B+1)n + 25k$.

Proof. Consider clause i composed of literals $y_{i,1}$, $y_{i,2}$ and $y_{i,3}$. By Lemma 8, the vertex labeling ℓ^* of each of the subtrees $T[y_{i,1}]$, $T[y_{i,2}]$ and $T[y_{i,3}]$ must adhere to one of four cases. It is easy to verify that case (i) has migration number $\mu(T[y_{i,j}] \geq 3$ (Fig. 5a), case (ii) has

migration number $\mu(T[y_{i,j}] \geq 10$ (Fig. 5b), case (iii) has migration number $\mu(T[y_{i,j}] \geq 6$ (Fig. 5c) and case (iv) has migration number $\mu(T[y_{i,j}] \geq 10$ (Fig. 5d). At most one of $T[y_{i,1}]$, $T[y_{i,2}]$ and $T[y_{i,3}]$ can be labeled by ℓ^* according to cases (i) or (iii), due to the restriction that G^* must be a tree (i.e. $\mathcal{P} = \{S\}$). Without loss of generality, we assume that only $T[y_{i,1}]$ adheres to case (i) or (iii). We distinguish three cases.

1. $T[y_{i,1}]$ is of case (i), and $T[y_{i,2}]$ and $T[y_{i,3}]$ are of cases (ii) or (iv): In $T[y_{i,1}]$, the parent of vertex v_1 is labeled by $y_{i,j}$. Moreover, in both $T[y_{i,2}]$ and $T[y_{i,3}]$, the parent of vertex v_1 is labeled by $\bot$. As such, these two vertices in $T[y_{i,2}]$ and $T[y_{i,3}]$ must be assigned label $y_{i,j}$. Hence, the minimum number of migrations induced by ℓ^* on the subtree of T induced by vertices $r(T)$, $V(T[y_{i,1}])$, $V(T[y_{i,2}])$ and $V(T[y_{i,3}])$ is $3 + 11 + 11 = 25$.
2. $T[y_{i,1}]$ is of case (iii), and $T[y_{i,2}]$ and $T[y_{i,3}]$ are of cases (ii) or (iv): In $T[y_{i,1}]$, the parent of vertex v_1 is labeled by $\neg y_{i,j}$. Moreover, in both $T[y_{i,2}]$ and $T[y_{i,3}]$, the parent of vertex v_1 is labeled by $\bot$. As such, these two vertices in $T[y_{i,2}]$ and $T[y_{i,3}]$ must be assigned label $\neg y_{i,j}$. Hence, the minimum number of migrations induced by ℓ^* on the subtree of T induced by vertices $r(T)$, $V(T[y_{i,1}])$, $V(T[y_{i,2}])$ and $V(T[y_{i,3}])$ is $6 + 11 + 11 = 28$.
3. $T[y_{i,1}]$, $T[y_{i,2}]$ and $T[y_{i,3}]$ are of cases (ii) or (iv): The minimum number of migrations induced by ℓ^* on the subtree of T induced by vertices $r(T)$, $V(T[y_{i,1}])$, $V(T[y_{i,2}])$ and $V(T[y_{i,3}])$ is $10 + 10 + 10 = 30$.

Thus, for each clause i, the minimum number of migrations induced by ℓ^* on $r(T)$, $V(T[y_{i,1}])$, $V(T[y_{i,2}])$ and $V(T[y_{i,3}])$ is 25. Thus, all k clauses lead to a least $25k$ migrations. By Lemmas 6 and Lemma 7, we have that $\ell^*(r(T)) = \bot$ and $\ell^*(r(T[x])) \in \{x, \neg x\}$ for each variable x. Thus, all variable subtrees $T[x]$ induce $(B + 1)n$ migrations. Hence, $\mu(T, \ell^*) \geq (B + 1)n + 25k$. ◀

Finally, we show that the above lower bound is tight if and only if ϕ is satisfiable.

▶ **Lemma 10.** *Boolean formula ϕ is satisfiable if and only if $\mu(T, \ell^*) = (B + 1)n + 25k$.*

Proof. ($\Rightarrow$) We construct a vertex labeling ℓ with migration number $\mu(T, \ell) = (B+1)n+25k$ such that the resulting migration graph G has an S pattern.

First, we set $\ell(r(T)) = \bot$. For each variable x, we set $\ell(r(T[x])) = x$ if $\theta(x) = 1$ and set $\ell(r(T[x])) = \neg x$ if $\theta(x) = 0$. Next, we consider each clause i. By the premise, there must exist a literal in clause i that satisfies the clause. Without loss of generality, we assume that $\theta(y_{i,1}) = 1$. We assign vertex labels to the vertices of $T[y_{i,1}]$ by setting $\ell(r(T[y_{i,1}])) = y_{i,1}$, $\ell(v_1) = \ldots = \ell(v_5) = y_{i,1}$ and $\ell(v_6) = \neg y_{i,1}$, according to case (i) of Lemma 8 (Fig. 5a). For the other two literals $y_{i,j}$ (where $j \in \{2, 3\}$), we set $\ell(r(T[y_{i,j}])) = \bot$. Let x' be the variable corresponding to $y_{i,j}$. If $\theta(x') = \psi(y_{i,j})$, we set $\ell(v_2) = \ldots = \ell(v_5) = y_{i,j}$ and $\ell(v_6) = y_{i,2}$, according to case (ii) of Lemma 8 (Fig. 5b). Otherwise, we set $\ell(v_2) = \ldots = \ell(v_5) = \neg y_{i,j}$ and $\ell(v_6) = \neg y_{i,2}$, according to case (iv) of Lemma 8 (Fig. 5d).

It is easy to verify that $\mu(T, \ell) = (B + 1)n + 25k$. By construction, each literal label $y_{i,j}$, as well as each clause label c_i, has a unique parent in G. Thus, G adheres to an S pattern. By Lemma 9, it holds that $\mu(T, \ell^*) \geq (B + 1)n + 25k$. Hence, ℓ is an optimal vertex labeling and $\mu(T, \ell^*) = (B + 1)n + 25k$.

($\Leftarrow$) We construct a truth assignment θ from ℓ^*. By Lemma 7, we have that $\ell^*(r(T[x])) \in \{x, \neg x\}$ for each variable x. We set $\theta(x) = \psi(\ell^*(r(T[x])))$. We claim that θ satisfies at least one literal for each clause i of ϕ.

By Lemmas 6, 7 and 9, we have that ℓ^* induces, for each clause i, 25 migrations in the subtree induced by vertices $V(T[y_{i,1}]) \cup V(T[y_{i,2}]) \cup V(T[y_{i,3}]) \cup \{r(T)\}$. Thus, one subtree among $T[y_{i,1}]$, $T[y_{i,2}]$ and $T[y_{i,3}]$ must have 3 migrations, corresponding to case (i) of Lemma 8 (Fig. 5a). Let $T[y_{i,j}]$ be the subtree with 3 migrations and let x be the variable

corresponding to $y_{i,j}$. As $\ell^*(r(T[y_{i,j}])) = y_{i,j}$, we have that $\ell^*(r(T[x])) = y_{i,j}$ and thus $\theta(x) = \psi(\ell^*(r(T[x]))) = \psi(y_{i,j})$. Thus, θ satisfies literal $y_{i,j}$ and thereby clause i. The same argument applies to each clause of ϕ. Hence, θ is a truth assignment that satisfies ϕ. ◀

Phylogenetic tree T has $1+A+(2B+1)n+16kn$ vertices can be constructed in polynomial time from ϕ. For instance, setting $B = 10k + 2$ and $A = 2Bn + 27k + 1$ (respecting the requirement that $B > 10k + 1$ and $A > 2Bn + 27k$), yields a total of $56kn + 27k + 9n + 2$ vertices in T, which is polynomial in the number k of clauses and the number n of variables. Thus, by Lemma 10, we have a polynomial time reduction from 3-SAT to PMH, proving Theorem 3.

4.2 Fixed parameter tractability

A PMH instance $(T, \hat{\ell})$ has $m = |\Sigma|$ locations that label $n = |V(T)|$ vertices. Typically, $m \ll n$ in practical problem instances. Here, we show that PMH is fixed parameter tractable in m and give an algorithm that runs in time $O(nm^m)$.

Our algorithm relies on the notion of a *migration tree* $\hat{G}$, a rooted tree whose vertex set is Σ. Unlike a migration graph, which is a directed multi-graph, $\hat{G}$ is a simple directed graph with at most one edge between every pair of vertices. We say that a vertex labeling ℓ is *consistent* with migration tree $\hat{G}$ provided that for any distinct pair (s, t) of locations there exists an edge $(u, v) \in E(T)$ such that $\ell(u) = s$ and $\ell(v) = t$ if and only if $(s, t) \in E(\hat{G})$.

For a given PMH instance $(T, \hat{\ell})$ and migration tree $\hat{G}$ there may not exist a vertex labeling consistent with $\hat{G}$ (Fig. 6). Let $\text{LCA}_{\hat{G}}(u) = \text{LCA}_{\hat{G}}(\hat{\ell}(L(T_u)))$ for vertex $u \in V(T)$. For $u \preceq_T v$, let $d_T(u, v)$ be the number of edges on the unique path from u to v. We show that the condition

$$d_T(u, v) \geq d_{\hat{G}}(\text{LCA}_{\hat{G}}(u), \hat{\ell}(v)) \qquad \forall u, v \in V(T) \text{ such that } u \preceq_T v, \tag{2}$$

is both necessary and sufficient for the existence of a vertex labeling of T consistent with $\hat{G}$.

▶ **Lemma 11.** *If there exists a vertex labeling ℓ for T consistent with $\hat{G}$ then (2) holds.*

Proof. Let ℓ be a vertex labeling for T consistent with $\hat{G}$. Let u and v be vertices of T such that $u \preceq_T v$. By Proposition 5, we have that $\ell(u) \preceq_{\hat{G}} \text{LCA}_{\hat{G}}(u)$. Thus, the number of migrations on the path from u to v must be at least $d_{\hat{G}}(\text{LCA}_{\hat{G}}(u), \hat{\ell}(v))$. To accommodate these migrations, the path from u to v must have at least $d_{\hat{G}}(\text{LCA}_{\hat{G}}(u), \hat{\ell}(v))$ edges. Hence, $d_T(u, v) \geq d_{\hat{G}}(\text{LCA}_{\hat{G}}(u), \hat{\ell}(v))$. ◀

We prove sufficiency constructively, using the vertex labeling ℓ^* of T defined as

$$\ell^*(v) = \begin{cases} \text{LCA}_{\hat{G}}(r(T)), & \text{if } v = r(T), \\ \sigma(\ell^*(\pi(v)), \text{LCA}_{\hat{G}}(v)), & \text{if } v \neq r(T), \end{cases} \tag{3}$$

where $\sigma(s, t) = s$ if $s = t$ and otherwise $\sigma(s, t)$ is the unique child of s that lies on the path from s to t in $\hat{G}$. In the case where $\sigma(\ell^*(\pi(v)), \text{LCA}_{\hat{G}}(v)) = \text{LCA}_{\hat{G}}(v)$ for all vertices v, vertex labeling ℓ^* is identical to the LCA mapping [5] used in the context of gene-tree species-tree reconciliation to minimize the duplication number.

▶ **Lemma 12.** *If (2) holds then vertex labeling ℓ^* for T is consistent with $\hat{G}$.*

Proof. For ℓ^* to be consistent with $\hat{G}$ it must hold that: (i) for any distinct pair (s, t) of locations there exists an edge $(u, v) \in E(T)$ such that $\ell(u) = s$ and $\ell(v) = t$ if and only if $(s, t) \in E(\hat{G})$, and (ii) $\ell^*(v) = \hat{\ell}(v)$ for each leaf v. Condition (i) holds by definition of ℓ^*.

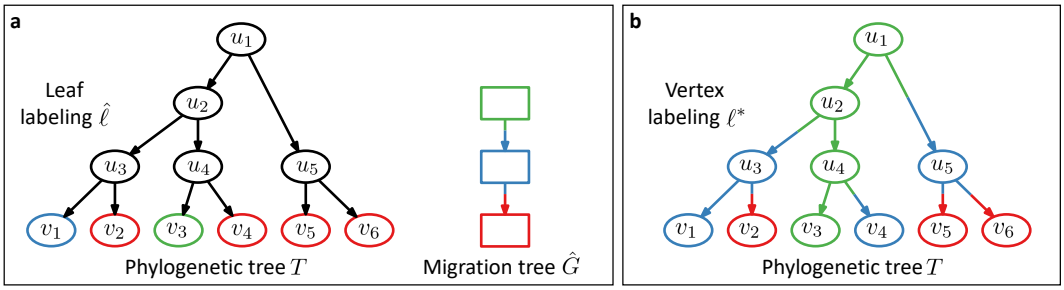


Figure 6 Given a phylogenetic tree T and migration tree $\hat{G}$, there may not exist a vertex labeling of T consistent with $\hat{G}$. (a) Here, any vertex labeling ℓ consistent with $\hat{G}$ must label u_1, u_2, u_4 by green. This, however, introduces a migration from green to red, violating $\hat{G}$. By Lemma 11, there does not a vertex labeling of T consistent with $\hat{G}$. (b) Equivalently, labeling ℓ^* defined in (3) is not a vertex labeling of T, as it changes the labeling of leaf v_4, i.e. $\ell^*(v_4) \neq \hat{\ell}(v_4)$.

That is, each vertex u has either the same label as its parent $\pi(u)$ or is labeled by a child of $\ell^*(\pi(u))$ in $\hat{G}$. As for condition (ii), assume for a contradiction that there exists a leaf v such that $\ell^*(v) \neq \hat{\ell}(v)$. Consider the unique path from $r(T)$ to v. Let u_1 be the vertex closest to v such that either $u_1 = r(T)$ or $\ell^*(u_1) = \ell^*(\pi(u_1))$. Thus, each edge on the path $u_1, \ldots, u_t, v$ is a migration. By definition of ℓ^*, we have that $\ell^*(u_1), \ldots, \ell^*(u_t), \ell^*(v)$ forms a path of $\hat{G}$.

We distinguish two cases. First, $u_1 = r(T)$. We have that $\ell^*(r(T)) = \text{LCA}_{\hat{G}}(r(T))$. As $\ell^*(v) \neq \hat{\ell}(v)$, we have that $d(r(T), v) < d_{\hat{G}}(\text{LCA}_{\hat{G}}(r(T)), \hat{\ell}(v))$. This contradicts the premise. Second, $u_1 \neq r(T)$. Let $u_0 = \pi(u_1)$. As $\ell^*(u_0) = \ell^*(u_1)$, we have that $\ell^*(u_1) = \text{LCA}_{\hat{G}}(u_1)$. Since $\ell^*(v) \neq \hat{\ell}(v)$, we have that $d(u_1, v) < d_{\hat{G}}(\text{LCA}_{\hat{G}}(u_1), \hat{\ell}(v))$. This contradicts the premise. Hence, ℓ^* is a vertex labeling of T that is consistent with $\hat{G}$. ◀

Vertex labeling ℓ^* has minimum migration number $\mu(T, \ell^*)$. To show this, we prove the following lemma that states that ℓ^* assigns each vertex a location nearest to the leaf set $L(\hat{G})$.

▶ **Lemma 13.** *Let ℓ be a vertex labeling ℓ of T that is consistent with $\hat{G}$. Then, $\ell(u) \preceq_{\hat{G}} \ell^*(u)$ for each vertex $u \in V(T)$.*

Proof. Assume for a contradiction that vertex u is the nearest vertex to $r(T)$ such that $\ell(u) \not\preceq_{\hat{G}} \ell^*(u)$ (i.e. $d_T(r(T), u)$ is minimum). By Proposition 5, we have that $\ell^*(u) \preceq_{\hat{G}} \text{LCA}_{\hat{G}}(u)$ and $\ell(u) \preceq_{\hat{G}} \text{LCA}_{\hat{G}}(u)$. Thus, $\ell(u) \not\preceq_{\hat{G}} \ell^*(u)$ implies that $\ell^*(u) \prec_{\hat{G}} \ell(u)$. Moreover, we have that $u \neq r(T)$, as $\ell^*(r(T)) = \ell(r(T)) = \text{LCA}_{\hat{G}}(r(T))$ by the same proposition. Since u is the nearest vertex to $r(T)$ such that $\ell(u) \not\preceq_{\hat{G}} \ell^*(u)$, we have that $\ell^*(\pi(u)) = \ell(\pi(u))$.

We distinguish two cases. First, $\ell(u) = \ell(\pi(u))$. Thus, $\ell(u) = \ell^*(\pi(u))$. Therefore, $\ell^*(u) \prec_{\hat{G}} \ell(u)$ implies that $\ell^*(u) \prec_{\hat{G}} \ell^*(\pi(u))$. This contradicts the definition of ℓ^*. Second, $\ell(u) \neq \ell(\pi(u))$. As $\ell^*(\pi(u)) = \ell(\pi(u))$, we have that $(\pi(u), u)$ is a migration from $\ell^*(\pi(u)) = \ell(\pi(u))$ to $\ell(u)$ in ℓ and to $\ell^*(u)$ in ℓ^*. By Lemma 12, we have that ℓ^* is consistent with $\hat{G}$. As $\ell^*(u) \prec_{\hat{G}} \ell(u)$, the labeling by ℓ of edge $(\pi(u), u)$ results in an edge $(\ell(\pi(u)), \ell(u))$ that is not in $\hat{G}$. Hence, ℓ is not consistent with $\hat{G}$, yielding a contradiction. Both cases yield a contradiction. Hence, the lemma follows. ◀

Finally, we prove that ℓ^* achieves the minimum migration number among all vertex labelings that are consistent with a given migration tree $\hat{G}$.

▶ **Lemma 14.** *Let T be a phylogenetic tree with leaf labeling $\hat{\ell}$ satisfying (2) for a given migration graph $\hat{G}$. Then, ℓ^* is a minimum migration labeling of T that is consistent with $\hat{G}$.*

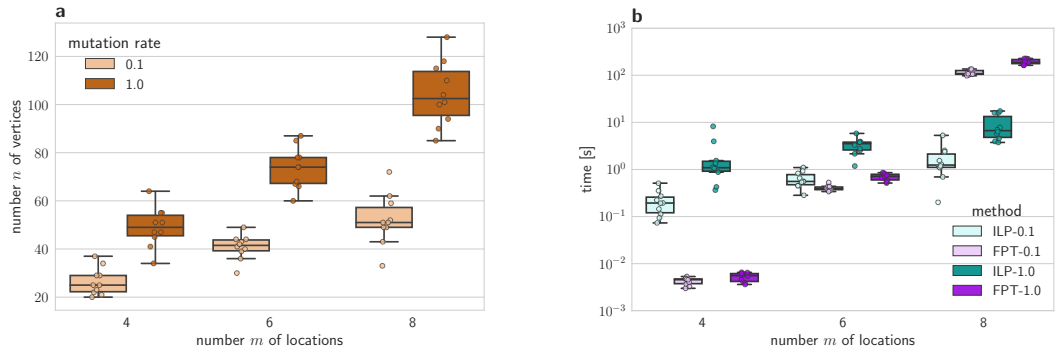


Figure 7 The fixed parameter tractable (FPT) algorithm is faster than the previously published [2] integer liner program (ILP) for small number m of locations. Using a tool for simulating metastatic cancers, we generated 60 PMH instances with varying number m of locations and number n of vertices. (a) The number of vertices increased with increasing mutation rate. (b) Running time in seconds (logarithmic scale) for FPT algorithm and ILP for instances with varying number m of locations and mutation rate.

Proof. Assume for a contradiction that ℓ is a vertex labeling of T consistent with $\hat{G}$ such that $\mu(T,\ell) < \mu(T,\ell^*)$. Consider a location t such that the number of migrations with target t is greater in ℓ^* than ℓ. Observe that $t \neq r(\hat{G})$. Let s be the parent of t in $\hat{G}$. Let $Y = \{v \in V(T) \mid \ell(\pi(v)) = s, \ell(v) = t\}$ and $Y^* = \{v \in V(T) \mid \ell^*(\pi(v)) = s, \ell^*(v) = t\}$. By the premise we have that $|Y^*| > |Y|$.

For any vertex labeling ℓ' consistent with $\hat{G}$, distinct edges $(u,v),(u',v') \in E(T)$ labeled by $\ell'(u) = \ell'(u') = s$ and $\ell'(v) = \ell'(v') = t$ are incomparable. Thus, the vertices in Y (Y^*) are pairwise incomparable in T. Let $L(t)$ be the set of leaves u of T labeled by $\hat{\ell}(u) = t'$ where $t \preceq_{\hat{G}} t'$. Let $L(T_Y)$ ($L(T_{Y^*})$) be the combined set of leaves in the subtrees rooted at each vertex $v \in Y$ ($v \in Y^*$). As both ℓ and ℓ^* are consistent with $\hat{G}$, we have that $L(t) = L(T_Y) = L(T_{Y^*})$. Since $|Y^*| > |Y|$ and $L(T_Y) = L(T_{Y^*})$, there must exist two distinct vertices $v, w \in Y^*$ and vertex $u \in Y$ such that $u \prec_T v$ and $u \prec_T w$. Since $u \preceq_T \pi(v)$, $u \preceq_T \pi(w)$ and $\ell^*(\pi(v)) = \ell^*(\pi(w)) = s$, we have that $\ell^*(u) \preceq_{\hat{G}} s$. Given that $\ell(u) = t$ and $s \prec_{\hat{G}} t$, we have that $\ell^*(u) \preceq_{\hat{G}} s \prec_{\hat{G}} t \preceq_{\hat{G}} \ell(u)$, which contradicts Lemma 13. Hence, ℓ^* is a minimum migration labeling of T that is consistent with $\hat{G}$. ◀

We note that ℓ^* can be computed in $O(nm)$ time—i.e., each $\text{LCA}_{\hat{G}}(\cdot)$ query can be resolved in $O(m)$ time using a simple post-order tree traversal of $\hat{G}$. The number of unrooted trees on m labeled vertices is m^{m-2}, which is known as Cayley's formula [1]. Since every unrooted tree can be rooted at each of its m vertices, the number of migration trees is m^{m-1}. Thus, a brute-force algorithm that computes ℓ^* for each migration tree $\hat{G}$ requires $O(nm^m)$ time, proving Theorem 4.

5 Experimental Evaluation

We implemented the FPT algorithm in C++ and used Prüfer sequences [8] to enumerate migration trees. The code is available at `https://github.com/elkebir-group/PMH-S`. Using simulated data of metastatic cancers, we compared the FPT algorithm and the previously published integer linear program (ILP) [2]. More specifically, for each combination of $m \in \{4,6,8\}$ locations and mutation rates $N \in \{0.1, 1.0\}$, we simulated 10 metastatic cancer where each of the $m-1$ metastases was seeded by clones from a single location. The simulation algorithm is described in more detail in [2]. In total, we simulated 60 PMH problem instances where $\mathcal{P} = \{S\}$.

Increasing the mutation rate N resulted in larger phylogenetic trees (Fig. 7a). We ran both the FPT algorithm and the ILP in single-threaded mode on a computer with two Intel Xeon CPUs at 2.6 GHz (32 cores) and 512 GB of RAM. As a sanity check, we found that both the ILP and the FPT algorithm resulted in the same minimum migration number for each instance (data not shown). We found that the running time of the FPT algorithm is only moderately affected by the size of the phylogenetic tree in contrast to the ILP (Fig. 7b). In addition, we found that the FPT algorithm outperformed the ILP for $m \in \{4, 6\}$ locations in terms of running time, but was slower for $m = 8$ locations (Fig. 7b). These findings are in line with the asymptotic running time of $O(nm^m)$ for the FPT algorithm. For modest number m of locations the FPT algorithm is the method of choice due to its simplicity.

6 Conclusion

In this work, we studied the complexity of the Parsimonious Migration History (PMH) problem, a constrained multi-objective optimization problem for reconstructing the migration history of a given phylogenetic tree T whose leaves are labeled by locations. For the case where the resulting migration graph G is restricted to a tree (i.e. $\mathcal{P} = \{S\}$), we showed that PMH is NP-hard and fixed parameter tractable in the number m of locations. We demonstrated that our FPT algorithm runs in time $O(nm^m)$ and outperforms the previously integer linear program for small m on simulated instances of practical size. While we did not consider the polytomy resolution variant of PMH, as introduced in [2], our hardness proof easily extends to this case by appropriately binarizing the gadgets used in the reduction. The hardness of PMH for the cases where G is restricted to a DAG (i.e. $\mathcal{P} = \{S, M\}$) or left unrestricted (i.e. $\mathcal{P} = \{S, M, R\}$) remains open.

References

1 A Cayley. A theorem on trees. *The Quart. J. of Pure and Appl. Math.*, 23:376–8, 1889.
2 Mohammed El-Kebir, Gryte Satas, and Benjamin J Raphael. Inferring parsimonious migration histories for metastatic cancers. *Nature Genetics*, 50(5):718–726, 2018.
3 Joseph Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, nov 1981.
4 Walter M Fitch. Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Systematic Zoology*, 20(4):406, 1971.
5 M Goodman et al. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Biology*, 28(2):132–163, 1979.
6 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.
7 A W McPherson et al. Divergent modes of clonal spread and intraperitoneal mixing in high-grade serous ovarian cancer. *Nature Genetics*, may 2016.
8 H Prüfer. Neuer Beweis eines Satzes über Permutationen. *Arch Math Phys*, 27:742–4, 1918.
9 David Sankoff. Minimal Mutation Trees of Sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, jan 1975.
10 M Slatkin and W P Maddison. A cladistic measure of gene flow inferred from the phylogenies of alleles. *Genetics*, 123(3):603–613, 1989.

The Wasserstein Distance as a Dissimilarity Measure for Mass Spectra with Application to Spectral Deconvolution

Szymon Majewski[1]
Institute of Mathematics, Polish Academy of Sciences, Warsaw, Poland
smajewski@impan.pl

Michał Aleksander Ciach[1]
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
m_ciach@student.uw.edu.pl

Michał Startek
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland

Wanda Niemyska
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland

Błażej Miasojedow
Institute of Mathematics, Polish Academy of Sciences, Warsaw, Poland

Anna Gambin
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland

Abstract

We propose a new approach for the comparison of mass spectra using a metric known in the computer science under the name of Earth Mover's Distance and in mathematics as the Wasserstein distance. We argue that this approach allows for natural and robust solutions to various problems in the analysis of mass spectra. In particular, we show an application to the problem of deconvolution, in which we infer proportions of several overlapping isotopic envelopes of similar compounds. Combined with the previously proposed generator of isotopic envelopes, IsoSpec, our approach works for a wide range of masses and charges in the presence of several types of measurement inaccuracies. To reduce the computational complexity of the solution, we derive an effective implementation of the Interior Point Method as the optimization procedure. The software for mass spectral comparison and deconvolution based on Wasserstein distance is available at `https://github.com/mciach/wassersteinms`.

2012 ACM Subject Classification Applied computing → Chemistry

Keywords and phrases Mass spectrometry, Tandem mass spectrometry, Wasserstein distance, Earth mover's distance, Similarity measure, Jaccard score, Tanimoto similarity

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.25

Funding This work was partially supported by Polish National Science Center grants 2014/12/W/ST5/00592 and UMO-2017/26/D/ST6/00304.

[1] Both authors contributed equally to this work.

1 Introduction

Mass Spectrometry (MS) is one of the main analytical techniques of modern proteomics and metabolomics, which allows for identification and quantification of molecular compounds. In the first step, the particles are ionized; next, they are separated in an electromagnetic field according to their mass to charge ratio (m/z), and finally, transferred to a detector. The detected signal, usually proportional to the number of ions, is plotted against the corresponding m/z value on a *mass spectrum*. A pair of detected m/z value and the corresponding signal intensity is called a *peak*. The signal intensity is often referred to as *ion current* [11, 3].

The m/z value can be used to infer the chemical composition of molecules (see e.g. [1]), but it does not give information about its chemical structure. To gain insight into the latter, several measurement steps are performed in a technique called Tandem Mass Spectrometry (Tandem MS). After each step, a range of m/z value is selected, and ions from that range are subjected to fragmentation before the next measurement. The mass spectrum obtained from the n-th measurement is referred to as an MS^n spectrum.

Even though the MS^1 spectrum is recorded prior to any fragmentation, a single compound can give rise to several peaks. This is due to the natural occurrence of *isotopes* – atoms with the same number of electrons and protons, but different numbers of neutrons. Molecules which differ only in their isotopic compositions are termed *isotopologues*. A group of peaks corresponding to isotopologues of a single molecule is referred to as an *isotopic envelope* (c.f. Fig. 1).

Tandem MS can be used to identify the molecule under study. There are two main approaches to this task: *de novo* sequencing and database search. The first one strives to identify the elemental composition and/or structure of the molecule purely based on the mass spectrum of fragments. The second one searches a database of mass spectra obtained from known molecules to find the most similar one [16, 24, 25].

To be able to search for a similar spectrum, either a similarity or a distance measure needs to be employed. There are two main groups of such measures. The first one relies on the number of *matching peaks*. Two peaks are said to match if their m/z values differ by less than a given threshold. An example of such measure is the Jaccard score, equal to the number of matching peaks divided by the number of distinct peaks in both spectra. The second group of measures takes into account both the location and the intensities of peaks. An example of such measure is the Euclidean distance or the correlation coefficient [16, 24].

Both groups are similar in the sense that they compare peaks with the same m/z value. As a consequence, they are highly sensitive to even the slightest differences in chemical formulas. For example, *apigenin* ($C_{15}H_{10}O_5$) and *quercetin* ($C_{15}H_{10}O_7$) are two molecules which differ by two oxygen atoms (see Fig. 1). Even though this difference is relatively small compared to the overall atom count, the MS^1 spectra contain no matching peaks. Consequently, the discussed measures do not detect any similarity between these molecules. Some approaches make a preprocessing of spectra to infer an optimal pairwise matching of peaks before computing the similarity [10].

Our contribution. In the present work, we propose a new measure which quantifies both the differences in intensities and m/z values of peaks in a continuous way. As such, the measure is more robust to changes in chemical formulas than the most common measures based on peak matching. The measure is based on the concept of transporting the ion current between the spectra. The distance between spectra is equal to the total distance in the m/z domain covered by the current. This allows to express the distance between spectra in Daltons. This

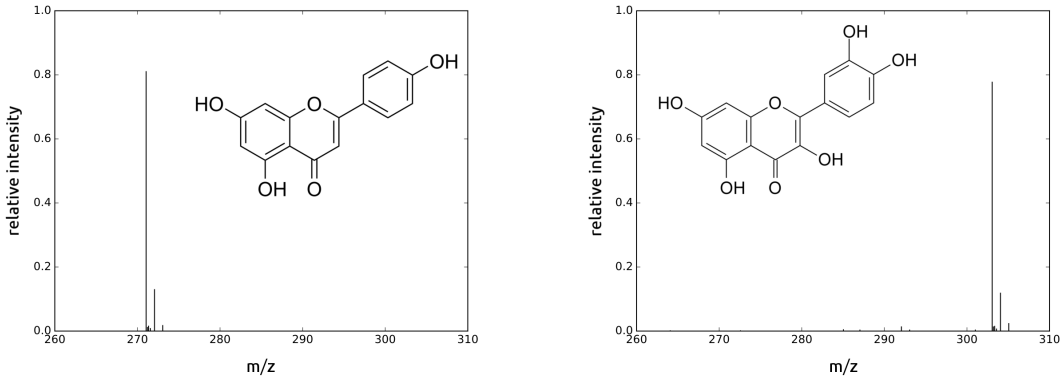


Figure 1 Molecular structures and MS^1 spectra of apigenin (left) and quercetin (right) showing their isotopic envelopes. Peak intensities have been normed to sum to 1. The mass spectra have been downloaded from the MassBank database (MassBank IDs: TY000164, TY000119).

distance is known in the field of probability theory as the (first) Wasserstein metric, and in the field of image processing as the Earth Mover's distance. Under certain assumptions, it can be computed in time linear in the number of distinct peaks in both spectra.

The Wasserstein distance allows to more accurately reflect the differences in chemical compositions of the molecules; for example, the distance between the MS^1 spectra from Fig. 1 is equal to 31.48 Da, while the difference in their masses is equal to 32.19 Da. Apart from quantifying the dissimilarity, the computed transport of ion current allows to match corresponding peaks in the compared spectra, which aids in the detection of differences in elemental composition and chemical structure (see Fig 2).

As a more advanced application of the Wasserstein distance, we show how it can be used to efficiently solve the problem of mass spectral deconvolution, i.e. the problem of separating overlapping isotopic envelopes.

Structure of the article. This article is structured as follows. In the next section, we introduce the formal definition of the Wasserstein metric. We also give some examples of the distance and the ion current transport between both experimental and *in silico* generated spectra. In Section 3, we give a formal statement of the deconvolution problem and the proposed solution based on the Wasserstein metric. Next, we show the results of computational experiments on *in silico* generated spectra to assess the performance of the solution. The results show that the Wasserstein distance allows for precise deconvolution in the presence of measurement errors even when the isotopic envelopes show considerable overlap.

2 The Wasserstein distance

In this section we introduce the first Wasserstein distance for discrete probability measures with finite support on $\mathbb{R}^d$. It is known in the computer science community as the Earth Mover's Distance, and was successfully used in a variety of applications including image processing [20]. In this article, we assume that the mass spectra are normalized so that the peak intensities sum to 1. This allows to interpret them as discrete probability measures.

Let $\mu = \sum_{i=1}^{n} w_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{m} v_j \delta_{y_j}$ be two discrete probability distributions, where w_i and v_i are sets of weights, x_i and y_i are points in $\mathbb{R}^d$, and δ_{x_i} is the Dirac delta centered at x_i. Suppose that those measures describe the distribution of some mass, i.e. there is a

mass of w_i at point x_i. Then, the Wasserstein distance can be defined as the minimal cost of transporting all the mass given by μ onto the mass given by ν. An example of such transport is given by Fig. 2.

Let $\gamma(i,j)$ denote the amount of mass transported from the point x_i to y_j. This function is referred to as a *transport plan*. Since all the mass from x_i needs to be transported somewhere, we have $\sum_{j=1}^{m} \gamma(i,j) = w_i$. On the other hand, the mass at y_j needs to be filled completely, implying that $\sum_{i=1}^{n} \gamma(i,j) = v_j$. This, combined with the non-negativity of the transport plan, means that $\gamma(i,j)$ can be interpreted as the joint probability distribution of the two measures.

Let $\Gamma(\mu,\nu)$ denote the set of all possible transport plans between the measures μ and ν. Then, the Wasserstein distance between the two measures, $W_1(\mu,\nu)$, is equal to the total distance traversed by the mass under the optimal transport plan:

$$W_1(\mu,\nu) = \min_{\gamma\in\Gamma(\mu,\nu)} \sum_{i=1}^{n}\sum_{j=1}^{m} \gamma(i,j)||x_i - y_j||.$$

It is easy to see that the function W_1 defined above is a metric on the space of discrete probability measures with finite support. Furthermore, the above construction can be extended to more general measures, underlying spaces and cost functions. For details we refer our reader to [21].

In the applications to mass spectrometry data, we will be mainly interested in the W_1 distance on the real line. This distance has the following important representation:

▶ **Lemma 1** (Proposition 2.17 in [21]). *Let μ and ν be two discrete probability measures with finite support on real line $\mathbb{R}$, and let F_μ and F_ν be their cumulative distribution functions. Then, we have:*

$$W_1(\mu,\nu) = \int_{\mathbb{R}} |F_\mu(x) - F_\nu(x)|dx$$

Based on the above Lemma, we can easily compute the distance $W_1(\mu,\nu)$ for finite discrete probability measures corresponding to normalized mass spectra. A common way of representing such a spectrum is a peak list, i.e. a list of pairs (x_i, p_i) such that x_i are in increasing order and represent m/z values of peaks with intensities p_i.

Algorithm 1, adapted from [12], shows how to compute W_1 for two such lists of peaks. It is based on the observation that the absolute difference between the cumulative distribution functions of mass spectra, $|F_\mu - F_\nu|$, is a step function, and therefore it is easily integrable. Furthermore, the value $F_\mu(x) - F_\nu(x)$ is equal to the surplus of the intensity in μ relative to ν, which needs to be transported through the point x.

Note that the runtime of Algorithm 1 is $\mathcal{O}(n+m)$: in each iteration of the main loop either i or j is incremented, no index variable will ever exceed the length of the corresponding list, and the algorithm terminates when both indices have reached the end of their respective lists.

2.1 Some basic properties

To help the reader gain some initial intuitions behind the Wasserstein distance, in this short subsection we discuss some of its qualitative properties when applied to mass spectra. In the next subsection, we illustrate some of the points discussed here by computational experiments.

First, for MS^1 spectra of two molecules, the Wasserstein distance is approximately equal to the absolute mass difference of the molecules. The other main factor that influences the distance in this case is the presence of measurement inaccuracies. Note, however, that

Algorithm 1: Computation of Wasserstein distance between two spectra.

Data: Two lists, N, M, of pairs (x, p), containing the lists of peaks of respective spectra
Result: W_1 distance between given spectra

$i \leftarrow 0$; $j \leftarrow 0$; $ret \leftarrow 0.0$; $\gamma \leftarrow$ an empty transport scheme
$n \leftarrow \text{length}(N); m \leftarrow \text{length}(M)$
while $i < n \vee j < m$ **do**
  $d \leftarrow \min(N[i].p, M[j].p)$
  $ret \leftarrow ret + d \cdot |N[i].x - M[j].x|$
  $N[i].p \leftarrow N[i].p - d$
  $M[j].p \leftarrow M[j].p - d$
  $\gamma(i, j) \leftarrow d$
  if $0 = N[i].p$ **then**
    $i \leftarrow i + 1$
  else
    $j \leftarrow j + 1$
  end
end
The variable ret contains the Wasserstein distance and γ the transport plan.

this influence remains small as long as the inacurracies in intensity measurements are small compared to the corresponding peak intensities.

Usually, some inaccuracy in both the intensity and the mass measurement is present. Naturally, the latter poses a major problem for measures based on peak matching. On the other hand, the Wasserstein distance is not significantly influenced by small mass measurement errors – instead, the imprecise measurement simply gets shifted to match its theoretical counterpart.

The implicit assumption of this metric, which may not be desirable in some applications, is that the mass difference reflects chemical difference. Therefore, two molecules differing by an OH group are assumed to be more similar to each other than two molecules differing by a C_2H_5 group. It is possible to relax this assumption by applying a different metric in the mass domain, say $c(x, y)$, in the definition of the Wasserstein distance:

$$W_c(\mu, \nu) = \min_{\gamma \in \Gamma(\mu,\nu)} \sum_{i=1}^{n} \sum_{j=1}^{m} \gamma(i, j) c(x_i - y_j).$$

An important caveat in this case is that treating all modifications as equivalent may lead to unexpected results – notably, a protein being treated as a single carbon atom with an extremely large modification. Furthermore, using other distances in the mass domain may lead to difficult optimization problems. The use of absolute difference, $|x_i - y_j|$, allows to avoid the optimization in the space of all possible transport plans.

If the two molecules differ by a modification which does not change their fragmentation pattern, then the Wasserstein distance between their MS^2 spectra will not exceed the weight of the modification. This follows from the observation that the modification is present only in some of the fragments, which account to a fraction of the total intensity in the spectrum. Note that this is a highly idealized example, since modifications may significantly change the fragmentation patterns and inflict a greater influence on the Wasserstein distance. In general, however, this distance cannot exceed the mass of the heavier molecule.

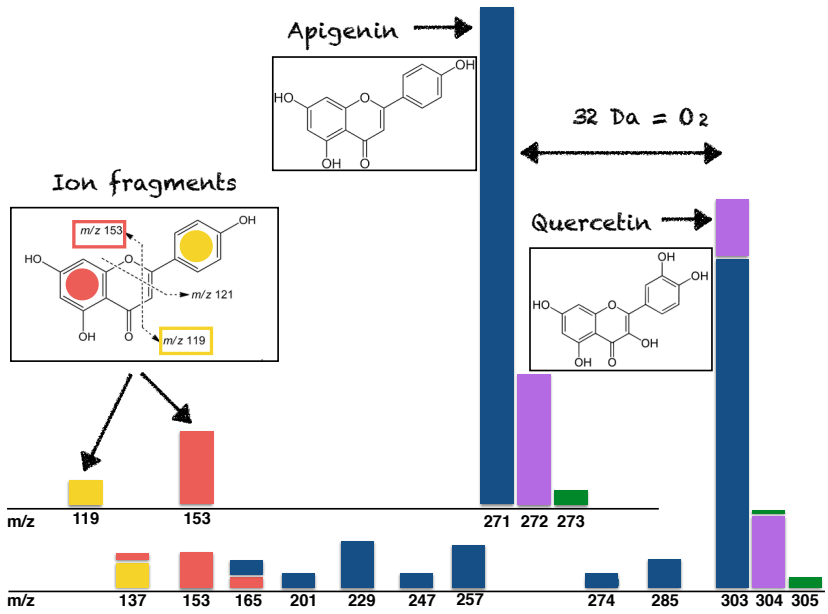


Figure 2 The optimal ion current transport plan for MS^2 spectra of apigenin (top) and quercetin (bottom), fragmented using 30 eV collision energy. The colors on the quercetin mass spectrum correspond to the origin of the transported ion current. The isotopic envelope of quercetin is shifted by 32 Da, i.e. the mass of two oxygen atoms.

Lastly, the structure of the optimal transport plan is highly sensitive to chemical noise, i.e. the presence of unexpected molecules. Recall that all the intensity from one spectrum needs to be used to explain all the intensity of the second spectrum. Therefore, if one of the analyzed spectra contains an additional peak, some of the intensity from the first spectrum needs to be used to explain it. This may lead to global changes in the structure of the optimal transport plan. It follows that in the presence of chemical noise, the Wasserstein distance may not reflect the similarity between the analyzed compounds.

2.2 Case study

To quantitatively analyze the properties of the Wasserstein metric when applied to mass spectral data, we have analyzed two sets of spectra obtained from the MassBank database [5]. In both cases, we have compared the performance of the Wasserstein distance with two other popular approaches: the Euclidean distance and the Jaccard score (i.e. the ratio of matching peaks to the total number of different peaks in both spectra). When analyzing those two measures, the spectra were binned to 0.01 Da resolution to increase the number of matching peaks and decrease their sensitivity to small measurement errors. No binning was performed during the analysis of the Wasserstein metric.

The first test was based on 615 MS^1 ESI-QTOF spectra with positive ionization mode. The goal of comparing MS^1 spectra was to verify the correlation between Wasserstein distance and the difference in mass of the molecules. The spectra have been compared pairwise, resulting in 188805 pairs. These pairs were then used to compute the Spearman's rank correlation between the distance and the absolute difference between masses of the corresponding molecules. The results are summarized in Table 1.

Table 1 Spearman's rank correlations between the Wasserstein distance, Jaccard score, Euclidean distance, and either the absolute mass difference or Tanimoto similarity of chemical structures. M, absolute mass difference; W, Wasserstein distance; J, Jaccard score; E, Euclidean distance; T, Tanimoto similarity; RW, relative Wasserstein distance; RE, relative Euclidean distance (see text).

MS^1 spectra

	M	**W**	**J**	**E**
M	1.00	0.89	-0.07	-0.37
W	–	1.00	-0.08	-0.22
J	–	–	1.00	-0.17
E	–	–	–	1.00

MS^2 spectra

	T	**RW**	**J**	**RE**
T	1.00	-0.41	0.22	-0.24
RW	–	1.00	-0.21	0.43
J	–	–	1.00	-0.11
RE	–	–	–	1.00

As expected, the metics based on peak matching are sensitive to mass differences, and therefore less correlated than the Wasserstein distance. Note that for the Wasserstein and Euclidean distance the correlation is expected to be positive, while for the Jaccard similarity metric it is expected to be negative. Surprisingly, we have found a negative correlation between the mass difference and the Euclidean distance.

The second test was based on MS^2 ESI-QTOF spectra with positive ionization mode. Here, the goal was to investigate the relationship between Wasserstein distance and structural similarity. Note that this distance is sensitive to the fragmentation intensity – two MS^2 spectra obtained for a given molecule can have a large distance if there is a significant difference in the intensity of fragments. To account for that, we have selected a subset of 473 MS^2 spectra for different molecules in which the precursor peak had around 10% relative intensity. This resulted in 111628 pairs of spectra. For each pair of spectra, we have computed the Wasserstein, Jaccard and Euclidean metrics. Next, we have computed the Tanimoto similarity between the structures of the corresponding molecules, based on the Morgan circular fingerprints [17, 19]. The fingerprints have been computed using the RDKit package (`http://www.rdkit.org`), with the radius of 2 and the default set of the feature-based invariants. The results are summarized in Table 1.

Note that the selected set of spectra comes from a diverse set of molecules. In particular, the mean mass is 310 Da, while the standard deviation is 160 Da. This poses a problem for the Wasserstein metric, as a pairs of small molecules will yield small distances regardless of the structural similarity. To account for this, we have divided the distance by the product of masses of the analyzed molecules. Without this correction, the correlation between the Wasserstein metric and the Tanimoto similarity drops to -0.22. This procedure also improved the correlation between the Tanimoto similarity and the Euclidean distance, but not the Jaccard score. We refer to the distances with this correction as *relative* distances.

The detailed relationship between the relative Wasserstein distance and the Tanimoto similarity is depicted in Fig. 3. For comparison, the Figure also shows the relationship between the Tanimoto similarity and the Jaccard score. Note that all compounds with high Tanimoto similarity have small relative Wasserstein distances. However, this relative distance is much more variable for compounds with low similarity. This frequent occurence of molecules with highly divergent structures but similar MS^2 spectra decreases the extent to which the Wasserstein distance correlates with the Tanimoto structural similarity.

The experiments show that the approaches to spectral comparison based on Wasserstein distance outperform the Jaccard score and the Euclidean distance in terms of correlation to mass in MS^1 spectra and to the chemical structure in MS^2 spectra. However, at this moment the Wasserstein distance should be applied only to MS^2 spectra with similar proportions of

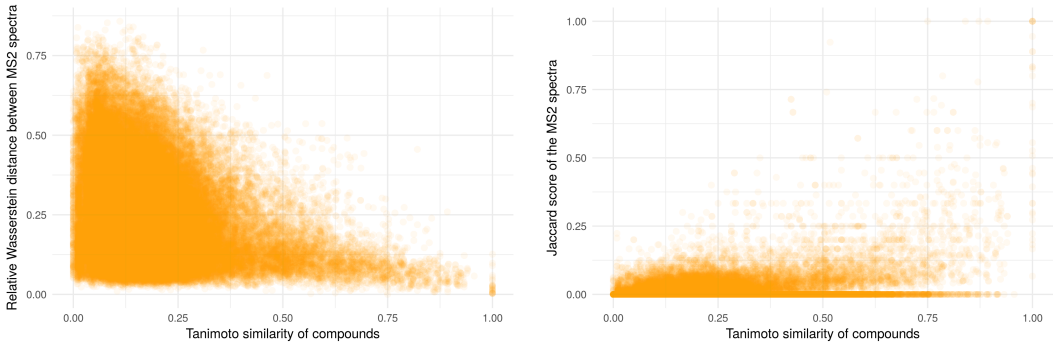


Figure 3 The relationship between MS^2 spectra and the structural similarity according to the relative Wasserstein distance (left) and the Jaccard score (right).

precursor molecules, and preferably obtained from compounds of similar mass. The results so far are optimistic, but more work needs to be done in order to generalize the Wasserstein distance so that it can be applied to a broader class of mass spectra.

3 Mass Spectral Deconvolution

In the mass spectral literature, the term *deconvolution* is used to refer to several problems which usually deal with separating overlapping peaks and/or isotopic envelopes. The authors of [4] define deconvolution as inferring the relative quantities of molecules with overlapping isotopic envelopes. Similar problems have been described in [7, 6, 15]. The term deconvolution is sometimes used as a synonym for *deisotoping*, that is, conversion of isotopic envelopes into single peaks with average m/z value and joint intensity [8, 13]. Other authors have used this term for conversion of m/z values to masses of multiply charged molecules [14, 18, 2].

We propose the following formalization of the mass spectral deconvolution, which encompasses several problems described above.

▶ **Problem 2** (Mass Spectral Deconvolution, MSD). *Let ν be a normalized mass spectrum, and let $\{\mu_i \colon 1 \leq i \leq k\}$ be a collection of normalized mass spectra. Let $d(\nu, \mu)$ be a distance measure between spectra. Let Δ_{k-1} be a $k-1$-dimentional probability simplex. Find a set of weights $p^* \in \Delta_{k-1}$ which minimizes the distance between ν and the convex combination of $\{\mu_i \colon i = 1, 2, \ldots, k\}$:*

$$p^* = \underset{p \in \Delta_{k-1}}{\operatorname{argmin}} \; d\left(\sum_{i=1}^{k} p_i \mu_i, \nu\right) \tag{1}$$

In the above definition, ν is referred to as an *experimental* spectrum, and μ_i are referred to as *theoretical* spectra. Note that neither the distance measure nor the origin of the theoretical spectra is not specified. Depending on the latter, this definition can be reduced to several of the problems mentioned at the beginning of this section. For example, if the theoretical spectra correspond to isotopic envelopes, the solution to MSD can be used for deisotoping, in which case p_i corresponds to the joint intensity of i-th envelope. On the other hand, if μ_i correspond to mass spectra of a single molecule with different charges, the problem reduces to conversion of m/z to mass. Finally, if μ_i are mass spectra from a database, the problem can be reduced to annotation of mass spectrum.

In this section, we propose a solution to the deconvolution problem using the Wasserstein metric as the distance measure, i.e. in (1) we use $d = W_1$. For the sake of clarity, we assume that the goal of MSD is to infer the proportions of compounds with overlapping isotopic

envelopes. Therefore, we assume that the user knows the molecular formulas of compounds which may be found in the analyzed spectrum. The theoretical spectra μ_i will correspond to mass spectra of those compounds predicted by the IsoSpec [26] algorithm. Note however that the proposed method is general, and can be easily extended to other use cases like deisotoping by simply choosing a different set of the theoretical spectra.

3.1 Efficient algorithm for the MSD problem

In what follows we will present an efficient method of solving MSD problem. First, we demonstrate the reduction of the original problem to a linear programming problem. Next, we will show how to use the structure of resulting linear programming problem to efficiently compute one iteration of a standard primal-dual Interior Point Method (IPM) (see for example [9]). In this subsection we present the main ideas behind the solution and state the most important results. The technical details and proofs are relegated to the Appendix.

Let $\nu = \sum_{j=1}^{m_0} w_{0,j}\delta_{x_{0,j}}$ be the experimental spectrum and for $i = 1, 2, \ldots, k$ let $\mu_i = \sum_{j=1}^{m_i} w_{i,j}\delta_{x_{i,j}}$ be the i-th theoretical spectrum. Denote the set of all support points from the empirical and theoretical spectra by $\mathcal{S} = \{x_{i,j} : 1 \leq j \leq m_i, 0 \leq i \leq k\}$ and let $n = |\mathcal{S}|$.

Let $\boldsymbol{s} = s_1 < s_2 < \cdots < s_n$ be the vector of ordered elements of $\mathcal{S}$. Notice that the cumulative distribution functions of μ_i's and ν are constant on intervals $[s_j, s_{j+1})$. For $1 \leq j \leq n-1$ let $f_{i,j}$ and g_j be the values of the cdfs of μ_i and ν respectively on the interval $[s_j, s_{j+1})$, and set $f_{i,n} = 1 = g_n$. Let $d_j = s_{j+1} - s_j$ be the length of the j-th interval.

Denote by F the $k \times n$ matrix with entries $F[i, j] = f_{i,j}$ for any $1 \leq i \leq k$ and $1 \leq j \leq n$, I_k an identity matrix of size k, and J_n an $(n-1) \times n$ matrix equal to the identity matrix of size n without the last row. We define

$$A = \begin{bmatrix} -J_n & -J_n & 0 \\ F & -F & -I_k \end{bmatrix}.$$

Finally, let c be a vector of length $2n + k$, such that $c = (g, -g, 0_k)$, where 0_k is the vector of zeros of length k, and $b = (-d, 0_k)$ be a vector of length $n - 1 + k$. The following Lemma, proved in the Appendix, states that MSD can be reduced to linear programming.

▶ **Lemma 3.** *The following dual linear programming problems:*

$$\begin{array}{lll} \min\limits_{x} & x^T c & \\ \text{s.t.} & Ax = b & \\ & x \geq 0 & \end{array} \qquad \begin{array}{lll} \max\limits_{y} & y^T b & \\ \text{s.t.} & A^T y + z = c & \\ & z \geq 0 & \end{array} \tag{2}$$

are feasible. Furthermore, for any solution (x_*, y_*, z_*) of the above problem, the vector of the last k elements of y_* belongs to the set of solutions of MSD.

We propose to solve the above linear program using IPM, while using the structure of our linear programming problem to significantly decrease the time and memory cost of each iteration. In general, IPM for linear programming solves both primal and dual problems simultaneously, by solving a cleverly chosen nonlinear approximation of those problems using the Newton's Method. For an overview of IPM we refer our reader to [9] and references therein. In Algorithm 2 we present the pseudocode for the general scheme of IPM for the dual problem (2). In what follows, x_t, y_t, z_t are t-th iterates of variables x, y, z from the problem, while $X_t = \mathrm{diag}(x_t)$, $Z_t = \mathrm{diag}(z_t)$ are diagonal matrices.

A triple (x_t, y_t, z_t) is called an ϵ-feasible ϵ-solution if it is both primal-dual feasible and optimal up to ϵ tolerance. Since the computational cost of each iteration is dominated by

Algorithm 2: General scheme of a primal-dual Interior Point Method.

Data: Matrix A and vectors b, c defining dual linear problems. A starting point (x_0, y_0, z_0) and error tolerance $\epsilon > 0$.

Result: an ϵ-feasible ϵ-solution of the linear problem

Set $t = 0$ **repeat**

compute centrality $\mu_t = \langle x_t, z_t \rangle / (2n + k)$ compute primal residual $r_p^t = b - Ax_t$ and dual residual $r_d^t = c - A^T y_t - z_t$ choose scaling factor σ_t (see Appendix) find direction (d_x, d_y, d_z) by solving the system of linear equations:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z_t & 0 & X_t \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} r_p^t \\ r_d^t \\ \sigma_t \mu_t \mathbf{1} - X^t z_t \end{bmatrix} \tag{3}$$

find $\alpha_p \in (0, 1]$ such that $x_{t+1} = x_t + \alpha_p d_x > 0$ find $\alpha_d \in (0, 1]$ such that $z_{t+1} = z_t + \alpha_d d_z > 0$ and take $y_{t+1} = y_t + \alpha_d d_y$ $t = t + 1$

until *triple* (x_t, y_t, z_t) *is an* ϵ*-feasible* ϵ*-solution*;

solving the equation (3), we focus on this part and relegate to the Appendix the discussion about choosing the stopping condition, the starting point (x_0, y_0, z_0) and the scaling factors σ_t.

The solution of the equation (3) can be obtained by solving the normal equation

$$AZ_t^{-1} X_t A^T d_y = b + AZ_t^{-1}(X_t r_d^t - \sigma_t \mu_t \mathbf{1}).$$

After solving the normal equation d_x, d_z can be obtained using formulas:

$$d_z = z = r_d^t - A^T d_y, \quad d_x = -x_t + Z_t^{-1}(\sigma_t \mu_t \mathbf{1} - X_t d_z).$$

The computational cost of single step of IPM is dominated by solving the normal equation, which is of order $\mathcal{O}((k(n-1)^3)$. However, thanks to the specific structure of matrix A, for any v, w we can compute Av, $A^T v$ and solve an equation $AZ_t^{-1} X_t A^T v = w$ efficiently. The detailed derivation of efficient algorithm, tailored to deconvolution problem is given in the Appendix. This allows us to perform one step of IPM efficiently, i.e the cost of single step is of order $\mathcal{O}(k^3 + k \sum_{i=1}^{k} m_i + n)$.

▶ **Lemma 4.** *One step of IPM for problem defined in Lemma 3 can be computed in time* $\mathcal{O}(k^3 + k \sum_{i=1}^{k} m_i + n)$ *and memory* $\mathcal{O}(k^2 + n)$.

Note that the for the given error tolerance ϵ, the IPM needs $\mathcal{O}(\sqrt{2n + k} \log(\epsilon^{-1}))$ iterations to find an ϵ-accurate solution.

3.2 Computational experiments

We have performed several computational experiments to illustrate the performance of the proposed solution to MSD, and to analyze its robustness to various kinds of distorsions occuring in MS measurements. In contrast to the previous case studies, in this section we use *in silico* generated spectra. This allows us to precisely control the signal-to-noise ratio, and to rigorously estimate the error of the method.

Our main goal is to demonstrate the applicability of the Wasserstein distance to MSD in case of noisy experimental spectra. There are several sources of noise in mass spectrometric measurements, among others: *(i)* precision of the intensity measurement, *(ii)* precision of the m/z measurement, *(iii)* resolving power, i.e. the ability to detect peaks with similar masses,

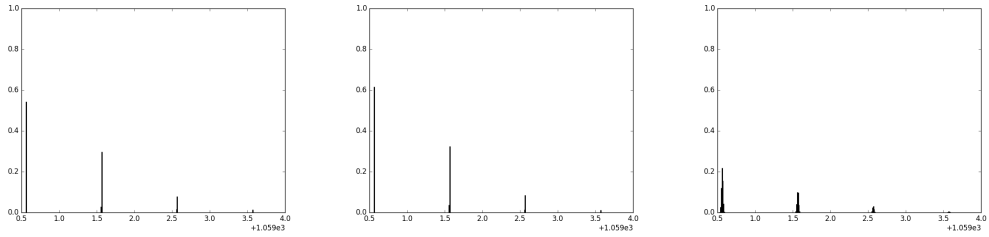


Figure 4 An illustration of the simulated measurement inaccuracies based on a theoretical spectrum of bradykinin ($C_{50}H_{73}N_{15}O_{11}$). Left: clean spectrum. Middle: noise in the intensity domain. Right: noise in the mass domain. The apparent change in intensity in the right spectrum is caused only by blurring the peaks and binning afterwards.

(iv) chemical noise, i.e. presence of unexpected molecules in a spectrum [3]. In this section, we focus mostly on the first three types of noises, i.e. low resolving power and/or precision. The first step of all our experiments was to generate the isotopic envelopes of selected molecules by the IsoSpec algorithm [26]. These envelopes form the set of the theoretical spectra. The experimental spectrum was obtained by taking a convex combination of the latter. Finally, the experimental spectrum was distorted in the following manner:

- Gaussian noise has been added to the logarithm of the peak intensity, and the result has been exponentiated,
- Each peak has been replaced by the density function of the normal distribution,
- The resulting intensity distribution has been binned.

Both Gaussian noises had a standard deviation of 0.01. For binning of the mass spectrum, we have assumed two resolving powers of the spectrometer: 0.001 Da and 0.01 Da. An example of the result of this procedure is depicted in Fig. 4. A spectrum without these distortions is referred to as *clean*, while the distorted one as *noisy*.

The performance of our approach to MSD has been quantified by the Root Mean Square Error (RMSE) between the original and inferred proportions of different isotopic envelopes. Note that the absolute prediction error for any variable does not exceed RMSE; moreover, if RMSE for deconvolution of n variables is equal ϵ, then it cannot happen that the absolute error exceeds $\epsilon/\sqrt{n}$ on every variable at the same time.

We have performed three tests, inspecting the method's sensitivity to the number of overlapping envelopes, molecular mass of deconvolved molecules, and their charge. The first test is based on random molecules. The next two are based on simulated proteins composed of *averagine* – a model aminoacid with molecular formula $C_4 \cdot 9384\, H_7 \cdot 7583\, N_1 \cdot 3577\, O_1 \cdot 4773\, S_0 \cdot 0417$ and average molecular mass of 111.1254 Da [22]. In the first test, we have inspected both clean and noisy spectra. In test two and three, only noisy spectra were analyzed. To check the standard deviation of the prediction error, the tests were replicated, with noise added independently in each replicate. Below we present each test in detail.

Test no. 1 – increasing number of molecules. This test is based on 17 randomly chosen isobars consisting of carbon, oxygen, hydrogen, nitrogen and sulfur, each one with the nominal mass of 30 000 Da. The experimental spectra with a range of interfering isobars were constructed by gradually extending a subset of those molecules. This procedure was replicated 20 times, resulting in 340 experimental spectra. The results are shown on Fig. 5. The prediction error is very low and stable for less than 5 isobars, suggesting that the ratios of particular elements in the deconvolved molecules have no considerable influence on the method's performance.

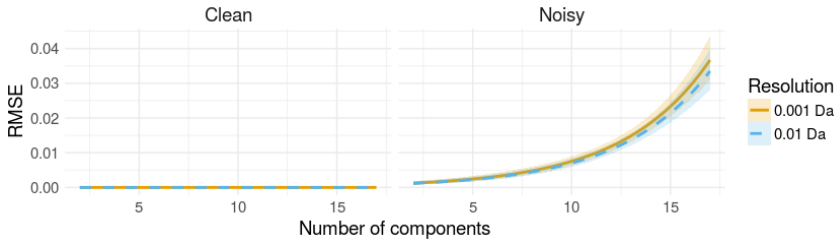


Figure 5 The performance of MSD method for increasing number of deconvolved molecules. The solid line represents the average RMSE over 20 repetitions. The ribbon represents the standard deviation of the error.

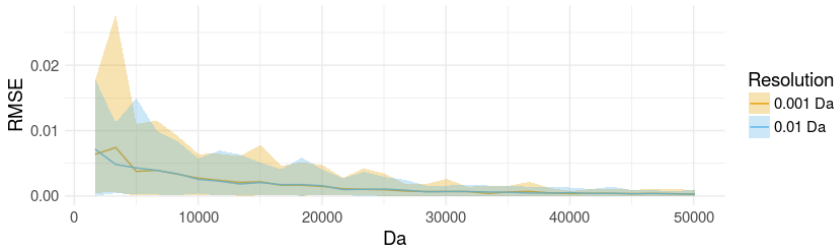


Figure 6 The performance of MSD method for increasing mass of deconvolved molecules. The solid line represents the average RMSE over 50 repetitions. The ribbon represents the standard deviation of the error.

Test no. 2 – increasing mass of molecules. In this test, we consider overlapping isotopic evelopes of two types of proteins: singly charged protein consisting of n units of averagine, and doubly charged protein consisting of $2n$ units, where the values of n were selected so that the average m/z ratio of proteins spans the range from 1,500 Da to 45,000 Da. For any given n, the isotopic envelopes of the two corresponding proteins were mixed in proportions 0.8 and 0.2 respectively. The procedure was replicated 50 times. The outcome of this experiment is presented in Fig. 6.

Test no. 3 – increasing charge of molecules. In this test, we consider the following four proteins based on averagine: $C_{1482}H_{2328}N_{408}O_{444}S_{12}$, $C_{1482}H_{2329}N_{408}O_{444}S_{12}$, $C_{1482}H_{2330}N_{408}O_{444}S_{12}$ and $C_{1481}H_{2341}N_{408}O_{444}S_{12}$, mixed in proportions 0.3, 0.5, 0.1 and 0.1, respectively. The first three molecules differ by one hydrogen atom, resulting in partially overlapping isotopic envelopes. The fourth molecule is an isobar of the second one, as one carbon has been replaced by 12 hydrogens. All molecules have been equally charged, with the charge varying from 1 to 10. This yields a sequence of deconvolution problems with increasing difficulty, because the peaks become more densely packed while the resolution stays constant. For each charge, 50 replicates were performed. The results are presented in Fig. 7.

Computational experiments show that our approach is able to deconvolve complex spectra even in the presence of measurement inaccuracies. Even for 17 isobars the RMSE does not exceed 0.05, which implies that the prediction error does not exceed 0.0125 on all 17 variables simultaneously. However, it must be noted that our approach to MSD is expected to be sensitive to chemical noise, because the Wasserstein metric requires that all the intensity of the experimental spectrum is explained. Therefore, our solution to MSD should be applied to mass spectra of highly purified compounds.

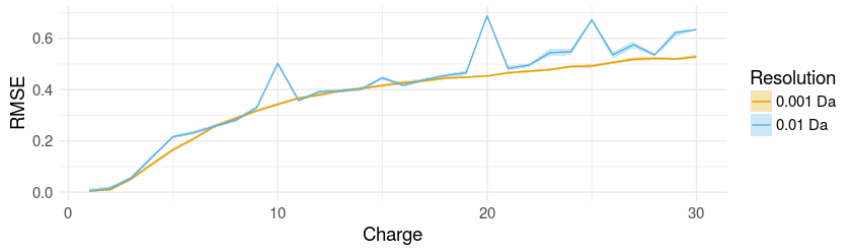


Figure 7 The performance of MSD method for increasing charge of deconvolved molecules. The solid line represents the average RMSE over 50 repetitions. The ribbon represents the standard deviation of the error.

4 Discussion and conclusions

In this work, we have presented a new method of comparing mass spectra. The method is based on the first Wasserstein metric, which is a well-established and well-studied metric in both probability theory and image processing field. Compared to the current approaches for spectral comparison, it is more robust to differences in chemical composition and measurement errors. In the MS^2 spectra of similar compounds, the Wasserstein distance reflects differences in both chemical structure and fragmentation intensities.

We have proposed a new formalization of the Mass Spectral Deconvolution (MSD) problem, which encompasses separating of overlapping isotopic envelopes, deisotoping, and decharging. We have shown that the Wasserstein distance can be used to effectively solve this problem in the presence of measurement inaccuracies.

The proposed solution for MSD works for a wide range of m/z values and multiply charged ions. Furthermore, it is not limited to a single class of compounds like peptides or metabolites. The theoretical isotopic envelopes can be either predicted *in silico* or measured experimentally.

The implementation of the Wasserstein metric for mass spectral comparison and deconvolution is available at `https://github.com/mciach/wassersteinms`.

References

1 S. Böcker and K. Dührkop. Fragmentation trees reloaded. *Journal of Cheminformatics*, 8(1):5, 2016.

2 S. Cappadona, P. R. Baker, P. R. Cutillas, A. JR. Heck, and B. van Breukelen. Current challenges in software solutions for mass spectrometry-based quantitative proteomics. *Amino acids*, 43(3):1087–1108, 2012.

3 I. Eidhammer, K. Flikka, L. Martens, and S-O. Mikalsen. *Computational methods for mass spectrometry proteomics*. John Wiley & Sons, 2008.

4 F. Lermyte et al. Conformational Space and Stability of ETD Charge Reduction Products of Ubiquitin. *J. Am. Soc. Mass Spectrom.*, Aug 2016.

5 H. Hisayuki et al. Massbank: a public repository for sharing mass spectral data for life sciences. *Journal of Mass Spectrometry*, 45(7):703–714, 2010.

6 K. Xiao et al. Accurate and Efficient Resolution of Overlapping Isotopic Envelopes in Protein Tandem Mass Spectra. *Sci Rep*, 5:14755, Oct 2015.

7 M.M Koek et al. Quantitative metabolomics based on gas chromatography mass spectrometry: status and perspectives. *Metabolomics*, 7(3):307–328, Sep 2011.

8 N. Jaitly et al. Decon2ls: An open-source software package for automated processing and visualization of high resolution mass spectrometry data. *BMC Bioinformatics*, 10(1):87, 2009.

9 J. Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

10 M. E. Hansen and J. Smedsgaard. A new matching algorithm for high resolution mass spectra. *J. of Am. Soc. Mass Spectrom.*, 15(8):1173–1180, 2004.

11 C. E. Housecroft and E. C. Constable. *Chemistry: An introduction to organic, inorganic and physical chemistry*. Pearson education, 2010.

12 Jedrzej Jablonski and Anna Marciniak-Czochra. Efficient algorithms computing distances between radon measures on r. *arXiv preprint arXiv:1304.3501*, 2013.

13 Q. Kou, L. Xun, and X. Liu. Toppic: a software tool for top-down mass spectrometry-based proteoform identification and characterization. *Bioinformatics*, 32(22):3495–3497, 2016.

14 M. Mann, C. K. Meng, and J. B. Fenn. Interpreting mass spectra of multiply charged ions. *Analytical Chemistry*, 61(15):1702–1708, 1989.

15 J. Meija and J.A. Caruso. Deconvolution of isobaric interferences in mass spectra. *J. of Am. Soc. Mass Spectrom.*, 15(5):654–658, 2004.

16 S. Neumann and S. Böcker. Computational mass spectrometry for metabolomics: identification of metabolites and small molecules. *Analytical and Bioanalytical Chemistry*, 398(7-8):2779–2788, 2010.

17 Nina Nikolova and Joanna Jaworska. Approaches to measure chemical similarity–a review. *Molecular Informatics*, 22(9-10):1006–1026, 2003.

18 B. B. Reinhold and V. N. Reinhold. Electrospray ionization mass spectrometry: Deconvolution by an entropy-based algorithm. *J. of Am. Soc. Mass Spectrom.*, 3(3):207–215, 1992.

19 David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010.

20 Y. Rubner, C. Tomasi, and L.J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, Nov 2000.

21 F. Santambrogio. *Optimal transport for applied mathematicians*. Springer, 2015.

22 Michael W. Senko, Steven C. Beu, and Fred W. McLafferty. Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions. *Journal of the American Society for Mass Spectrometry*, 6(4):229–233, 1995.

23 R. J. Vanderbei et al. *Linear programming*. Springer, 2015.

24 K. X. Wan, I. Vidavsky, and M. L. Gross. Comparing similar spectra: from similarity index to spectral contrast angle. *J. of Am. Soc. Mass Spectrom.*, 13(1):85–88, 2002.

25 Ş. Yilmaz, E. Vandermarliere, and L. Martens. *Methods to Calculate Spectrum Similarity*, pages 75–100. Springer New York, New York, NY, 2017.

26 M. K. Łącki et al. IsoSpec: Hyperfast Fine Structure Calculator. *Anal. Chem.*, 89(6):3272–3277, Mar 2017.

A Appendix

A.1 Derivation of linear program

In this subsection we show how the MSD problem defined in (1) can be reduced to linear programming in the case when the distance measure is the Wasserstein metric W_1. We also prove Lemma 3.

Recall that our problem is to find:

$$p_* = \operatorname*{argmin}_{p \in \Delta_{k-1}} W_1 \left(\sum_{i=1}^{k} p_i \mu_i, \nu \right) \tag{4}$$

where μ_i, ν are discrete probability measures with finite support.

First, we show that the MSD problem can be restated as weighted L_1 regression on the probability simplex Δ_{k-1}. Using Lemma 1 we can write:

$$\operatorname*{argmin}_{p \in \Delta_{k-1}} W_1 \left(\sum_{i=1}^{k} p_i \mu_i, \nu \right) = \operatorname*{argmin}_{p \in \Delta_{k-1}} \int_{\mathbb{R}} \left| \sum_{i=1}^{k} p_i F_{\mu_i}(x) - F_\nu(x) \right| dx. \tag{5}$$

Recall that $\mathcal{S}$ denotes the set of points from theoretical and empirical spectra, and that $(s_i)_{i=1}^n$ are elements of $\mathcal{S}$ ordered increasingly. Note that for $x < s_1$ and $x \geq s_n$, the function under the integral on the RHS of (5) is zero. At the same time, the function is constant on intervals $[s_i, s_{i+1})$. Let $1 \leq j \leq n-1$ let $f_{i,j}$ and g_j be the values of the cdf on the interval $[s_j, s_{j+1})$ of μ_i and ν respectively, and set $f_{i,n} = 1 = g_n$. For $1 \leq j \leq n-1$ let $d_j = s_{j+1} - s_j$ be the length of the interval. We write:

$$\int_{\mathbb{R}} \left| \sum_{i=1}^{k} p_i F_{\mu_i}(x) - F_\nu(x) \right| dx = \sum_{j=1}^{n-1} d_j \left| \sum_{i=1}^{k} p_i f_{i,j} - g_j \right|,$$

and we reduce the optimization problem (4) to weighted L_1 regression on probability simplex:

$$p_* = \operatorname*{argmin}_{p \in \Delta_{k-1}} \sum_{j=1}^{n-1} d_j \left| \sum_{i=1}^{k} p_i f_{i,j} - g_j \right| \tag{6}$$

Now, we apply a well known technique of representing weighted L_1 regression as linear programming problem (see e.g. [23]). Let us introduce dummy variables t_j, such that $t_j \geq |\sum_{i=1}^m p_i f_{i,j} - g_j|$. With this notation the problem (6) is equivalent to minimizing linear function $\sum_{i=1}^{n-1} d_j t_j$. For any j the inequality $t_j \geq |\sum_{i=1}^m p_i f_{i,j} - g_j|$ can be represented by two linear inequalities, $t_j \geq \sum_{i=1}^m p_i f_{i,j} - g_j$ and $t_j \geq -\sum_{i=1}^m p_i f_{i,j} + g_j$. To also take into account the fact that vector $(p_i)_{i=1}^k$ needs to belong to the probability simplex, we just need to add inequality constraints $p_i \geq 0$ and equality constraint $\sum_{i=1}^k p_i = 1$. By rewriting the

latter as two inequality constraints, we end up with the following linear program:

$$\begin{aligned} \min_{p,t} \quad & d^T t \\ \text{s.t.} \quad & -t_j + \sum_{i=1}^{m} p_i f_{i,j} \leq g_j \\ & -t_j - \sum_{i=1}^{m} p_i f_{i,j} \leq -g_j \\ & 1 \leq \sum_{i=1}^{m} p_i \leq 1 \\ & p_i \geq 0. \end{aligned} \tag{7}$$

The minimized function in (7) does not depend itself on p, however variable p appears in constrains. From the construction of (7) it follows that for any feasible pair (p, t) we have:

$$d^T t \geq \sum_{i=1}^{n-1} d_j \left| \sum_{i=1}^{k} p_i f_{i,j} - g_j \right|$$

with equality holds if and only if $t_j = |\sum_{i=1}^{k} p_i f_{i,j} - g_j|$. Therefore for any solution p_* of problem (6) there exists t_* such that (p_*, t_*) is a solution of (7). It also follows that for any solution (p_*, t_*) of (7), vector p_* is a solution of (6). Furthermore, since (6) is a problem of optimizing continuous convex function on a compact set, at least one solution exists.

We have thus reduced MSD problem (4) to a Linear Programming problem. Denote by F the $k \times n$ matrix with entries $F[i, j] = f_{i,j}$ for any $1 \leq i \leq k$ and $1 \leq j \leq n$, I_k an identity matrix of size k, and J_n an $(n-1) \times n$ matrix equal to the identity matrix of size n without the last row. Ussing definitions of vectors g and d, we can rewrite the problem (7) in a simplified form:

$$\begin{aligned} \max_{t} \quad & -d^T t \\ \text{s.t.} \quad & \begin{bmatrix} -J_n^T & F^T \\ -J_n^T & -F^T \\ 0 & -I_k \end{bmatrix} \begin{pmatrix} t \\ p \end{pmatrix} \leq \begin{pmatrix} g \\ -g \\ 0_k, \end{pmatrix} \end{aligned}$$

where 0_k is a zero vector of length k. We bring to our reader's attention that the constrain $\sum_{i=1}^{k} p_i = 1$, splits into two inequality constraints $\sum_{i=1}^{k} p_i \leq 1$ and $-\sum_{i=1}^{k} p_i \leq -1$, which was included in the above program, since the last row of matrix F^T contains ones.

Let us recall definitions of $c = (g, -g, 0_k)$, $b = (-d, 0_k)$ and

$$A = \begin{bmatrix} -J_n & -J_n & 0 \\ F & -F & -I_k \end{bmatrix}.$$

Using the above notation, and adding the slack variables z to replace inequality constrains by equality constrains, we rewrite the problem (7) as

$$\begin{aligned} \max_{y} \quad & y^T b \\ \text{s.t.} \quad & A^T y + z = c \\ & z \geq 0. \end{aligned} \tag{8}$$

This summarizes the reduction of the original minimization problem to linear programming. We end this section with the proof of Lemma 3.

Proof of Lemma 3. From the above discussion it follows that the feasible region of the problem (8) is nonempty. Furthermore, note that the problem given by (6) is bounded from below. This, combined with the inequality:

$$y^T b \leq -\sum_{j=1}^{n-1} d_j \left| \sum_{i=1}^{k} p_i f_{i,j} - g_j \right|,$$

means that the maximization problem from Lemma 3 has a solution. From the duality theory for linear programs it follows that the minimization problem is feasible as well and the duality gap is zero.

Finally, from the above discussion it follows as well that if (x_*, y_*, z_*) is a solution of dual problems in Lemma 3, then the last k elements of y_* form a vector in Δ_{k-1} that is a solution of the initial MSD problem. ◀

A.2 Interior Point Method

We remind our reader that our goal is to solve dual linear problems from Lemma 3. In the Section 3.1 we presented a general scheme for a primal-dual Interior Point Method for the case when primal problem has only equality constraints, as is the case for the problem of interest to us. In this section we discuss the details of the Algorithm 2, and prove our main claim presented in Lemma 4 by presenting an efficient way of computing one iteration of an IPM.

We first address the issues of initial conditions (x_0, y_0, z_0), stopping criterion and choosing the scaling factor σ_t. The IPM does not require the starting point or the iterates to be in feasible region. The only requirement is that all elements of vectors x_t, z_t are positive for $t \geq 0$. Hence we can choose almost arbitrary starting point, although in practice it is beneficial to choose x, z such that they are not to close to zero. We propose the following choice of x_0: for $1 \leq i \leq n-1$ we take $(x_0)_i = (x_0)_{n+i} = d_i/2$, and $(x_0)_n = 2/3$, $(x_0)_{2n} = 1/3$. We also let $(x_0)_{2n+i} = 1/3$ for $1 \leq i \leq k$. It is straightforward to check, that such x_0 satisfies $Ax_0 = b$, and $x_0 > 0$. Finding a point (y_0, z_0) that is dual feasible and $z_0 \geq 1$ is straightforward using the connection of the dual problem to L_1 regression described in (6). We choose a uniform vector $p \in \Delta_{k-1}$ and t_j equal to $|\sum_{i=1}^{k} p_i f_{i,j} - g_j| + 1$ for all $j \leq n-1$. Taking $y_0 = (t, p)$ and $z_0 = c - A^T y_0$ we get a dual feasible point with $z_0 \geq 1$. For now we just note that this construction can be easily computed using subroutines for computing $F^T v$ and $A^T v$ for given vectors v of appropriate size, and vector operations on vectors of size $\mathcal{O}(n+k)$. The efficient multiplication by F^T and A^T is described later.

The stopping criterion of the main loop of the IPM is triple (x_t, y_t, z_t) begin and ϵ-feasible ϵ-solution. The residuals r_p^t, r_d^t measure how far the triple (x_t, y_t, z_t) is from feasibility, while the duality gap $\langle x_t, z_t \rangle$ measures how far the point is from optimality. We say a point is an ϵ-solution it the duality gap is smaller than a given ϵ. For a point to be ϵ-feasible, the norms of the residuals have to be smaller than ϵ, that is $\|r_p^t\|_2 < \epsilon$ and $\|r_d^t\|_2 < \epsilon$. In practice usually different $\epsilon_o, \epsilon_p, \epsilon_d$ are chosen based on the problem, and the stopping criterion is $\langle x_t, z_t \rangle \leq \epsilon_o$, $\|r_p^t\|_2 < \epsilon_p$ and $\|r_d^t\|_2 < \epsilon_d$. We also note, that to prevent infinite loops practical algorithms stop after reaching a set maximal amount of iterations. From the perspective of our analysis, the most important fact is that checking the stopping criterion is of order $\mathcal{O}(n+k)$.

Choosing the scaling factor can be done in a variety of ways. The simplest method is to choose $\sigma_k = \gamma \in (0, 1)$. More sophisticated approaches like predictor-corrector method first find the Newton direction $(\hat{d}_x, \hat{d}_y, \hat{d}_z)$ for the most optimistic $\hat{\sigma}_t = 0$. Then the actual scaling factor σ_t is chosen based on how much reduction in duality gap could be achieved

while going in the direction $(\hat{d_x}, \hat{d_y}, \hat{d_z})$. Again, for more details on methods of choosing the scaling factor we refer to [9] and references therein. We just note that there exist methods for choosing the scaling factor that are both practical and lead to theoretical guarantees on the number of iterations necessary for an IPM to converge. Furthermore for the state of the art methods the cost of computing the scaling factor is the same as the cost of solving the system of linear equations for (d_x, d_y, d_z), and is actually done by solving a system of equations with the same RHS. As will be clear from what follows, the cost of solving this system of linear equations does not depend on the RHS.

The above discussion justifies our claim, that the cost of computing one step of an IPM is dominated by the cost of finding the solution of the system:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z_t & 0 & X_t \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} r_p^t \\ r_d^t \\ \sigma_t \mu_t \mathbf{1} - X^t z_t, \end{bmatrix} \tag{9}$$

where $r_p^t = b - Ax_t$ and $r_d^t = c - A^T y_t - z_t$.

As we previously noted, a common technique for solving the system of equations (9) is to reduce it to solving the normal equation $\Sigma d_y = r$, where:

$$\Sigma = A Z_k^{-1} X_k A^T$$

and

$$r = b + A(Z_k)^{-1}(X_k r_d^k - \sigma_k \mu_k \mathbf{1}).$$

This equation is arrived at, after applying a blockwise Gaussian elimination on the linear system (9). Given a solution d_y of the normal equation, we can compute d_x, d_z from equations $d_z = r_d^k - A^T d_y$ and $d_x = -x_k + (Z_k)^{-1}(\sigma_k \mu_k \mathbf{1} - X_k d_z)$.

Therefore as we have noted in Section 3.1 one iteration of IPM can be computed efficiently, if we can efficiently compute Av, $A^T v$ and solve $\Sigma v = w$ for v, w vectors of appropriate length, and $\Sigma = AHA^T$ where H is a diagonal matrix with positive elements on the diagonal. We devote the rest of this section to presenting and analyzing methods for those three computational problems, that take advantage of the specific structure of matrix A.

We first observe that matrix F is related to a sparse matrix. For simplicity we denote $m = \sum_{i=1}^{k} m_i$, where m_i are the sizes of theoretical spectra.

▶ **Lemma 5.** *Let U denote an upper-triangular $n \times n$ matrix, such that $U[i,j] = 1$ for $i \leq j$ and $U[i,j] = 0$ for $i > j$. Then there exists a sparse $m \times n$ matrix W, with m nonzero entries such that $F = WU$. Furthermore sparse representations of matrices W and W^T can be constructed in $\mathcal{O}((n+m)\log(n+m))$ time.*

Proof. Let $s_1 \leq \ldots \leq s_n$ be the ordered point from set $\mathcal{S}$. For any theoretical measure μ_i, we can represent μ_i as $\sum_{j=1}^{\mathcal{S}} w_{i,j}\delta_{s_j}$, whith only m_i nonzero elemets $w_{i,j}$ for each i. Take matrix W, such that $W[i,j] = w_{i,j}$. Then the matrix W is sparse and has a total of $\sum_{i=1}^{k} m_i$ nonzero elements. It is straightforward to check that $F = WU$, since for any i, j we have $WU[i,j] = \sum_{l \leq j} w_{i,l} = f_{i,j}$. In other words, the cdf of distribution μ_i on interval $[s_j, s_{j+1})$ is equal to the sum of probability masses in μ_i with x coordinate less or equal to s_{j+1}.

To finish the proof, we need to show how to construct a sparse representation of W and W^T. In a sparse representation of a matrix, we represent each row of W as a list of nonzero elements, i.e. $(j, w_{i,j})$ for j such that $w_{i,j} > 0$. The matrix W is represented as list of rows. This is a classic representation of a sparse matrix that allows to compute

Wv in time $\sum_{i=1}^{k} m_i$ for any vector $v \in \mathbb{R}^n$. Constructing the sparse representations of W and W^T is very easy, and needs to be done only once before the first iteration of the IPM. As previously let $\nu = \sum_{j=1}^{m_0} w_{0,j}\delta_{x_{0,j}}$ be the experimental spectrum and for $i = 1, 2, \ldots, k$ let $\mu_i = \sum_{j=1}^{m_i} w_{i,j}\delta_{x_{i,j}}$ be the i-th theoretical spectrum. We construct a list of triples $(x_{i,j}, i, w_{i,j})$ and sort it with respect to the first element. Then it is enough to pass once through this sorted list. Each triple with $i > 0$ corresponds to one element of the matrix W. Since we pass through the elements in increasing order of x, we can remember how many different x we have seen so far. For the triple (x, i, w), if we have seen l different x so far, then we know that we have $W[i, l] = w$, and we simply add (l, w) to i-th list in the representation of W and (i, w) to l-th list in the representation of W^T. The total cost of this construction is $\mathcal{O}((n + m) \log(n + m))$, due to the sorting of elements, and the memory needed for storing both representations is $\mathcal{O}(m)$. ◀

▶ **Lemma 6.** *For any vector $v \in \mathbb{R}^N$ and $w \in \mathbb{R}^{n+m-1}$ the products Av and $A^T w$ can be computed in $\mathcal{O}(n + k + m)$ time and using additional $\mathcal{O}(n + k)$ memory.*

Proof. The only nontrivial part of both of those operations is computing Fv or $F^T v$ for some vector v of appropriate length, which can be done efficiently thanks to the representation $F = WU$ given by Lemma 5. To compute $w = Fv$, we first compute $u = Uv$ which can be done in time $\mathcal{O}(n)$ by computing suffix sums of vector v, without the need to explicitly store matrix U. We need $\mathcal{O}(n)$ memory to store the result of this operation. Next, we have $w = Wu$. Thanks to the sparse representation, this multiplication can be done in $\mathcal{O}(m)$ time and we need $\mathcal{O}(k)$ memory for storing the result. Similarly, to compute $w = F^T v$, we first multiply v by W^T and then multiply the result by U^T which corresponds to computing prefix sums, and does not require explicit construction of U. As for multiplication by F we need $\mathcal{O}(n + k + m)$ time and $\mathcal{O}(n + k)$ memory for those operations. ◀

We are left with the task of solving a system of linear equations $\Sigma v = w$, where $\Sigma = AHA^T$ for a diagonal matrix H with positive elements on the diagonal. To prove that this can be done efficiently, we first prove.

▶ **Lemma 7.** *For any diagonal matrix G of size $n \times n$ the matrix FGF^T can be computed in $\mathcal{O}(m(k + m) + n)$ time and using $\mathcal{O}(m^2)$ memory.*

Proof. It is straightforward to check, that for $i, j \leq n$ we have $UDU^T[i, j] = \sum_{l=max_{i,j}}^{n} G[l, l]$. We have, also, $FGF^T = W(UDU^T)W^T$. Therefore if we compute the suffix sums of the diagonal of G in time and space $\mathcal{O}(n)$, we can compute $UDU^T[i, j]$ in constant time for any i, j when we need it. We denote $\alpha_{i,j} = UDU^T[i, j]$. Now let us choose $i, j \leq n$. We have:

$$FGF^T[i, j] = \sum_{p \leq n} \sum_{q \leq n} w_{i,p}\alpha_{p,q}w_{j,q}$$

Given the sparse representations of rows W_i, W_j the above sum could be computed in time $\mathcal{O}(m_i m_j)$, but there is a faster way. Notice that for $q \geq p$ we have $\alpha_{p,q} = \alpha_{q,q}$, and write:

$$\sum_{1 \leq p \leq q \leq n} w_{i,p}\alpha_{p,q}w_{j,q} = \sum_{1 \leq p \leq q \leq n} w_{i,p}\alpha_{q,q}w_{j,q} \tag{10}$$

The above sum can be computed in $\mathcal{O}(m_i + m_j)$. Let L_i, L_j be the lists containing the sparse representations of W_i, W_j. We first compute the suffix sums of $\alpha q, q w_{q,j}$ for $(q, \cdot) \in L_j$, which can be done in $\mathcal{O}(m_j)$ since list L_j is ordered by column number q. Then for any $(p, w_{i,p}) \in L_i$ we add to the result the suffix sum of $\alpha q, q w_{q,j}$ for the smallest $q \geq p$ such

that $(q, \cdot) \in L_j$. Since lists L_i, L_j are ordered by column numbers p, q, this can be done in $\mathcal{O}(m_i + m_j)$ time in standard way. Therefore (10) can be computed in $\mathcal{O}(m_i + m_j)$.

Similarly, the sum:

$$\sum_{1 \le q < p \le n} w_{i,p} \alpha_{p,q} w_{j,q} = \sum_{1 \le q < p \le n} w_{i,p} \alpha_{p,p} w_{j,q}$$

can be computed in $\mathcal{O}(m_i + m_j)$. Therefore after we compute suffix sums of the diagonal of G in time and memory $\mathcal{O}(n)$, the cell (i, j) of matrix FGF^T can be computed in $\mathcal{O}(m_i + m_j)$ time. We have

$$\sum_{1 \le i,j \le n} m_i + m_j = (2k-1)m$$

and therefore the whole matrix FGF^T can be computed in $\mathcal{O}(km + n)$ time and $\mathcal{O}(k^2 + n)$ memory. ◀

We can now prove that the normal equation can be solved efficiently, no matter the right hand side.

▶ **Lemma 8.** *Let $r \in \mathbb{R}^{n+k-1}$ be a vector and let H be a $(2n+k) \times (2n+k)$ diagonal matrix, with positive elements on the diagonal. Then the matrix AHA^T has full rank, and equation $AHA^T v = r$ can be found in $\mathcal{O}(k^3 + km + n)$ time and $\mathcal{O}(k^2 + n)$ space.*

Proof. First we present a usefull decomposition of matrix AHA^T. Suppose that $H = \operatorname{diag}(H_1, H_2, H_3)$ where H_1, H_2, H_3 are diagonal matrices of sizes n, n, k respectively. Then using the definition of A we can write:

$$AHA^T = \begin{bmatrix} J_n(H_1 + H_2)J_n^T & J_n(H_2 - H_1)F^T \\ F(H_2 - H_1)J_n^T & F(H_1 + H_2)F^T + H_3 \end{bmatrix} \tag{11}$$

We will use block LDU decomposition for the right hand side of the above equation. For a given matrix $B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$ with $B_{1,1}$ invertible the block LDU decomposition of B is:

$$B = \begin{bmatrix} I & 0 \\ B_{2,1}B_{1,1}^{-1} & I \end{bmatrix} \begin{bmatrix} B_{1,1} & 0 \\ 0 & B_{2,2} - B_{2,1}B_{1,1}^{-1}B_{1,2} \end{bmatrix} \begin{bmatrix} I & B_{1,1}^{-1}B_{1,2} \\ 0 & I \end{bmatrix}$$

We note, that $J_n(H_1 + H_2)J_n^T$ is a diagonal $(n-1) \times (n-1)$ matrix with positive elements on the diagonal, and is therefore invertible and easy to invert numerically. For brevity, we denote $K = J_n(H_1 + H_2)J_n^T$ and $L = J_n(H_2 - H_1)$ and $G = H_1 + H_2 - L^T K^{-1} L$. Using the block LDU decomposition for (11) we get:

$$AHA^T = \begin{bmatrix} I_{n-1} & 0 \\ FL^T K^{-1} & I_k \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & FGF^T + H_3 \end{bmatrix} \begin{bmatrix} I_{n-1} & K^{-1}LF^T \\ 0 & I_k \end{bmatrix} =: LMR$$

From above representation it follows, that AHA^T has full rank. Indeed, it is a product of square matrices L, M, R, and it is obvious that L and R have full rank. Furthermore M also has full rank. To see why, observe that G is a diagonal matrix with positive elements. This follows from the fact that elementwise we have $G \ge H_1 + H_2 - (H_2 - H_1)^2/(H_1 + H_2)$ (with equality on all elements except the right bottom row). On the other hand $H_1 + H_2 - (H_2 - H_1)^2/(H_1 + H_2) = 4H_1H_2/(H_1 + H_2)$ which is a diagonal matrix with positive elements. Therefore FGF^T is nonnegative definite, and since H_3 is positive definite, we conclude that

$FGF^T + H_3$ is positive definite. Since K is positive definite as well, M has full rank and so does AHA^T.

For given vector r we would like to solve the equation $LMRv = r$. An easy way to do this, is first solve equation $Lv_1 = r$, then solve equation $Mv_2 = v_1$ and lastly $Rv_3 = v_2$. Then we have $LMRv_3 = LMv_2 = Lv_1 = r$. We can therefore work with each matrix L, M, R separately.

Solving $Lv_1 = r$ is trivial. Let us assume that $r = (r_1, r_2)$, where r_1 is of length $n - 1$ and r_2 has length k. Then:

$$v_1 = \begin{pmatrix} r_1 \\ r_2 - FL^T K^{-1} r_1 \end{pmatrix}$$

which is easy to compute, since the cost of multiplying by $L^T K^{-1}$ is $\mathcal{O}(n)$, and we can efficiently multiply vector by F thanks to Lemma 6. Solving the linear equation $Rv_3 = v_2$ is equally easy, since we can efficiently multiply by F^T as well.

The only thing left is solving $Mv_2 = v_1$. Assume $v_1 = (u_1, u_2)$ where u_1 is of size $n - 1$ and u_2 is of size k. Let $P = FGF^T + H_3$. Then P is a positive definite matrix of size $k \times k$ that we can compute in $\mathcal{O}(km + n)$ thanks to Lemma 7. Let v' be a solution of $Pv' = u_2$, that we find using standard methods in time $\mathcal{O}(k^3)$ and space $\mathcal{O}(k^2)$. Then $v_2 = (u_1^T, v'^T)^T$ is the solution of equation $Mv_2 = v_1$.

Summing up, the cost of finding a solution to the equation $AHA^T v = r$ is $\mathcal{O}(k^3 + km + n)$, where $\mathcal{O}(k^3)$ is the cost of inverting a $k \times k$ matrix, $\mathcal{O}(km + n)$ is the cost of creating this matrix, and all other operations have cost linear in n, k, m. For computing the $k \times k$ matrix we need $\mathcal{O}(k^2)$ memory, but all other operations can be done using additional memory linear in n, k. ◀

Proof of Lemma 4. For each iteration we need to solve a finite amount of normal equations (maybe more than one, depending on our mechanism of choosing scaling factor) and additionally perform a finite amount of multiplications Av, $A^T v$, Fv, $F^T v$. Since the other operations can be done in time and memory linear in n, k, m, we conclude that one iteration of a primal-dual Interior Point Method can be done in $\mathcal{O}(k^3 + km + n)$ time and $\mathcal{O}(k^2 + n)$ memory. ◀