17th International Workshop on Algorithms in Bioinformatics

WABI 2017, August 21–23, 2017, Boston, MA, USA

Edited by

Russell Schwartz
Knut Reinert

LIPICS

Editors

Russell Schwartz	Knut Reinert
Carnegie Mellon University	Freie Universität Berlin
Pittsburgh, USA	Berlin, Germany
`russells@andrew.cmu.edu`	`knut.reinert@fu-berlin.de`

ACM Classification 1998
F.2.2 Nonnumerical Algorithms and Problems, F.2.2 Pattern Matching, I.1.2 Algorithms, J.3 Life and Medical Sciences

ISBN 978-3-95977-050-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-050-7.

Publication date
August, 2017

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.WABI.2017.0

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

17th International Workshop on Algorithms in Bioinformatics (WABI 2017).
Editors: Russell Schwartz and Knut Reinert
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This proceedings volume contains papers presented at the 17th Workshop on Algorithms in Bioinformatics (WABI 2017), which was held in Boston, MA, USA in conjunction with the 8th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB) from August 21–23, 2017.

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 explore directions of future research.

WABI 2017 is grateful for the support of ACM-BCB in allowing us to cohost the meetings, as well as to ACM-BCB's sponsors: the Association for Computing Machinery (ACM) and ACM's SIGBIO.

In 2017, a total of 55 manuscripts were submitted to WABI from which 27 were selected for presentation at the conference. This year, WABI is adopting a new proceedings form, publishing the conference proceedings through the LIPIcs (Leibniz International Proceedings in Informatics) proceedings series. 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 27 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 statistical inference, phylogenetic studies, sequence and genome analysis, comparative genomics, and mass spectrometry data analysis.

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 also thank all the conference participants and speakers who contribute to a great scientific program. In particular, we are indebted to the keynote speaker of the conference, Tandy Warnow, for her presentation. We also thank Christopher Pockrandt for setting up the WABI webpage and Umit Acar for his help with coordinating the WABI and ACM-BCB pages. Finally, we thank the ACM-BCB Organizing Committee, especially Nurit Haspel and Lenore Cowen, for their hard work in making all of the local arrangements and working closely with us to ensure a successful and exciting WABI and ACM-BCB.

List of Authors

Anastasiia Abramova
Saint Petersburg State University
St. Petersburg, Russia
abramova.asya93@gmail.com

Bahar Alipanahi
University of Florida
Gainesville, USA
bahar.panahie@gmail.com

Yannis Almirantis
National Center for Scientific Research
Demokritos
Athens, Greece
yalmir@bio.demokritos.gr

Fatemeh Almodaresi
Stony Brook University
New York, USA
falmodaresit@cs.stonybrook.edu

Stephen Altschul
National Institutes of Health
Bethesda, USA
altschul@ncbi.nlm.nih.gov

Amir H. Bayegan
Boston College
Boston, USA
bayegan@bc.edu

Shamsuzzoha Bayzid
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
shams.bayzid@gmail.com

Christina Boucher
University of Florida
Gainesville, USA
christinaboucher@ufl.edu

Hector Bravo
University of Maryland
College Park, USA
hcorrada@umiacs.umd.edu

Brian Brubach
University of Maryland
College Park, USA
bbrubach@cs.umd.edu

Evelyn Bunnik
University of Texas
Houston, USA
bunnik@uthscsa.edu

Pannagiotos Charalampopoulos
King's College London
London, UK
panagiotis.charalampopoulos@kcl.ac.uk

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

Michal Ciach
University of Warsaw
Warsaw, Poland
m_ciach@student.uw.edu.pl

Peter Clote
Boston College
Boston, USA
clote@bc.edu

Matteo Comin
University of Padova
Padova, Italy
comin@dei.unipd.it

Steven Cornwell
University of Pennsylvania
Philadelphia, USA
steffenc@seas.upenn.edu

Maxime Crochemore
King's College London
London, UK
maxime.crochemore@kcl.ac.uk

Peter Ebert
Max Planck Institute for Informatics
Saabrücken, Germany
pebert@mpi-inf.mpg.de

Nadia El-Mabrouk
University of Montreal
Montreal, Canada
mabrouk@iro.umontreal.ca

Alexandre Francisco
Universidade de Lisboa
Lisboa, Portugal
aplf@ist.utl.pt

Yves Frank
ETH Zurich
Zurich, Switzerland
yfrank@student.ethz.ch

Jia Gao
King's College London
London, UK
jia.gao@kcl.ac.uk

Manuela Geiß
University of Leipzig
Leipzig, Germany
manuela@bierdepot.bioinf.uni-leipzig.de

Samuele Girotto
University of Padova
Padova, Italy
samuele.girotto@gmail.com

Jay Ghurye
University of Maryland
College Park, USA
jayg@cs.umd.edu

Pavel Górecki
University of Warsaw
Warsaw, Poland
gorecki@mimuw.edu.pl

Mohamed Gunady
University of Maryland
College Park, USA
mgunady@cs.umd.edu

Md Abid Hasan
University of California, Riverside
Riverside, USA
mhasa006@ucr.edu

Marc Hellmuth
University of Greifswald
Greifswald, Germany
mhellmuth@mailbox.org

Jan Holub
Czech Technical University in Prague
Prague, Czech Republic
Jan.Holub@fit.cvut.cz

Thomas Hruz
ETH Zurich
Zurich, Switzerland
tomas.hruz@inf.ethz.ch

Chenyue Hu
Rice University
Houston, USA
wendyhu001@gmail.com

Costas Iliopoulos
King's College London
London, UK
costas.iliopoulos@kcl.ac.uk

Hosna Jabbari
University of Alberta
Edmonton, Canada
jabbari@ualberta.ca

Sreeran Kannan
University of Washington
Seattle, USA
ksreeram@uw.edu

Jonkyu Kim
Freie Universität Berlin
Berlin, Germany
j.kim@fu-berlin.de

Anton Korobeynikov
Saint Petersburg State University
St. Petersburg, Russia
anton@korobeynikov.info

Karine Le Roch
University of California, Riverside
Riverside, USA
karine.leroch@ucr.edu

Hanyang Li
Rice University
Houston, USA
hl43@rice.edu

Stefano Lonardi
University of California, Riverside
Riverside, USA
stelo@cs.ucr.edu

Mohamed Manal
King's College London
London, UK
manal.mohamed@kcl.ac.uk

Shunfu Mao
University of Washington
Seattle, USA
shunfu@uw.edu

Daniel Merkle
University of Southern Denmark
Odense, Denmark
daniel@imada.sdu.dk

Sohail Mohajer
University of Minnesota
Rochester, USA
soheil@umn.edu

Erin K. Molloy
University of Illinois
Urbana-Champaign, USA
emolloy2@illinois.edu

Carlo Montemagno
University of Alberta
Edmonton, Canada
montemag@ualberta.ca

Steven Mount
University of Maryland
College Park, USA
smount@umd.edu

Martin Muggli
University of Helsinki
Helsinki, Finland
Martin.Muggli@colostate.edu

Anna Muszewsak
Polish Academy of Sciences
Warsaw, Poland
mmusze@ibb.waw.pl

Sarvesh Nikumbh
Max Planck Institute for Informatics
Saabrücken, Germany
snikumbh@mpi-inf.mpg.de

Nikolai Nøjgaard
University of Greifswald
Greifswald, Germany
nnoej10@student.sdu.dk

Aïda Ouangraoua
Sherbrooke university
Quebec, Canada
aida.ouangraoua@usherbrooke.ca

Weihua Pan
University of California, Riverside
Riverside, USA
wpan005@ucr.edu

Prashand Pandey
Stony Brook University
New York, USA
ppandey@cs.stonybrook.edu

Rob Patro
Stony Brook University
New York, USA
Rob.Patro@cs.stonybrook.edu

Nico Pfeifer
University of Tübingen & Max Planck
Institute for Informatics
Saabrücken, Germany
pfeifer@informatik.uni-tuebingen.de

Solon Pissis
King's College London
London, UK
solon.pissis@kcl.ac.uk

Cinzia Pizzi
University of Padova
Padova, Italy
cinzia.pizzi@dei.unipd.it

Anton Polishko
University of California, Riverside
Riverside, USA
polishka@cs.ucr.edu

Dimitri Polychronopoulos
Imperial College London
London, UK
dpolychr@imperial.ac.uk

Mihai Pop
University of Maryland
College Park, USA
mpop@umiacs.umd.edu

Petr Procházka
Czech Technical University in Prague
Prague, Czech Republic
Petr.Prochazka@fit.cvut.cz

Simon Puglisi
University of Helsinki
Helsinki, Finland
puglisi@cs.helsinki.fi

Jens Quedenfeld
Technical University of Munich
Munich, Germany
jens.quedenfeld@in.tum.de

Amina Qutub
Rice University
Houston, USA
aminaq@rice.edu

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

Kannan Ramachandran
University of California, Berkeley
Berkeley, USA
kannanr@eecs.berkeley.edu

Knut Reinert
Freie Universität Berlin
Berlin, Germany
knut.reinert@fu-berlin.de

Abbas Roayaei Ardakany
University of California, Riverside
Riverside, USA
roayaei@gmail.com

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

Pijus Simonaitis
ENS Lyon
Lyon, France
pijus.simonaitis@ens-lyon.fr

Nidhi Shah
University of Maryland
College Park, USA
nidhi@cs.umd.edu

Aravind Srinivasan
University of Maryland
College Park, USA
srin@cs.umd.edu

Peter F. Stadler
University of Leipzig
Leipzig, Germany
studla@bioinf.uni-leipzig.de

Krister Swenson
Université Montpellier
Montpellier, France
swenson@lirmm.fr

Jussi Taipala
Karolinska Institutet
Stockholm, Sweden
jussi.taipale@ki.se

Jarkko Toivonen
University of Helsinki
Helsinki, Finland
jarkko.toivonen@cs.helsinki.fi

Thomas Tschager
ETH Zurich
Zurich, Switzerland
thomas.tschager@inf.ethz.ch

David Tse
Stanford University
Stanford, USA
dntse@stanford.edu

Esko Ukkonen
University of Helsinki
Helsinki, Finland
esko.ukkonen@cs.helsinki.fi

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

Cátia Vaz
Universidade de Lisboa
Lisboa, Portugal
cvaz@cc.isel.ipl.pt

Valentin Venzin
ETH Zurich
Zurich, Switzerland
vvenzin@student.ethz.ch

Ian Wark
University of Alberta
Edmonton, Canada
wark@ualberta.ca

Tandy Warnow
University of Illinois
Urbana-Champaign, USA
warnow@illinois.edu

Nicolas Wieseke
University of Leipzig
Leipzig, Germany
wieseke@informatik.uni-leipzig.de

Sebastian Will
University of Vienna
Vienna, Austria
will@tbi.univie.ac.at

Disentangled Long-Read De Bruijn Graphs via Optical Maps*

Bahar Alipanahi[1], Leena Salmela[2], Simon J. Puglisi[2], Martin Muggli[4], and Christina Boucher[5]

1 **Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA**
baharpan@ufl.edu
2 **Department of Computer Science, HIIT, University of Helsinki, Helsinki, Finland**
leena.salmela@cs.helsinki.fi
3 **Department of Computer Science, HIIT, University of Helsinki, Helsinki, Finland**
puglisi@cs.helsinki.fi
4 **Department of Computer Science, Colorado State University, Fort Collins, CO, USA**
martin.muggli@colostate.edu
5 **Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA**
christinaboucher@ufl.edu

Abstract

While long reads produced by third-generation sequencing technology from, e.g, Pacific Biosciences have been shown to increase the quality of draft genomes in repetitive regions, fundamental computational challenges remain in overcoming their high error rate and assembling them efficiently. In this paper we show that the de Bruijn graph built on the long reads can be efficiently and substantially disentangled using optical mapping data as auxiliary information. Fundamental to our approach is the use of the positional de Bruijn graph and a succinct data structure for constructing and traversing this graph. Our experimental results show that over 97.7% of directed cycles have been removed from the resulting positional de Bruijn graph as compared to its non-positional counterpart. Our results thus indicate that disentangling the de Bruijn graph using positional information is a promising direction for developing a simple and efficient assembly algorithm for long reads.

1998 ACM Subject Classification J.3 [Life and Medical Sciences] Biology and Genetics, G.2.2 Graph Theory

Keywords and phrases Positional de Bruijn graph, Optical maps, PacBio

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.1

1 Introduction

Today, high-throughput DNA sequencing technology is central to every major (re)sequencing and *de novo* assembly project. Complex and long repetitive regions in genomes are a challenge

* This work was supported by Academy of Finland grants 284598 (CoECGR) and 294143, and the National Science Foundation (NSF) grant III: 1618814.

for accurate assembly, especially for short-read sequencing technologies like Illumina, and this has driven a recent shift toward long-read sequencing technologies like Pacific Biosciences (PacBio). Long reads have already been successful in disambiguating the repetitive regions, leading to draft assemblies with fewer mis-assembled regions [25]. To date, however, long reads have a high error rate, which increases the complexity of assembly. For example, PacBio produces reads up to 50,000 bp in length, but with an insertion/deletion error rate of 15–20% [15].

Most assemblers targeting short read technologies use the Eulerian approach [13, 24]. In this assembly paradigm, all distinct k-mers (substrings of length k) are first extracted from the set of reads. A de Bruijn graph is then constructed with a vertex v for every $(k-1)$-mer present in the set of reads, and an edge (v, v') for every observed k-mer in the reads with $(k-1)$-mer prefix v and $(k-1)$-mer suffix v'. The prefix of string S is any substring of S that includes its first character, and accordingly the suffix of string S is any substring of it that includes its last character. A contig corresponds to a non-branching path through this graph. SPAdes [1], ABySS [29], and Velvet [32] are examples of short read assemblers using the Eulerian approach. This approach is computationally efficient but does not easily adapt to reads with a high error rate. Moreover, applying it to long reads seems to discard the long range information in those reads.

With the above-mentioned caveats in mind, the first assemblers for long reads have adopted the *Overlap-Layout-Consensus (OLC)* approach. OLC first calculates the overlap between all pairs (or a subset of the pairs) of sequence reads and builds an overlap graph (in which there is an edge between pairs of reads having highest overlap). Similarly to the Eulerian approach, contigs then correspond to the non-branching paths through this graph. The computational bottleneck in OLC is the computation of (approximate) suffix-prefix overlaps between reads, which becomes computationally infeasible when the number of reads and the error rate grows.

Optical mapping is another technology that has been proposed for solving the repetitive regions in genomes. A genome-wide optical map contains the approximate genomic location of each restriction site corresponding to one or more restriction enzymes. Put another way, the optical map is the sequence of locations corresponding to all the occurrences of a short nucleotide sequence (typically 5–7bp) in the genome. Optical maps span significantly larger genomic regions than long reads: the typical region covered by a genome wide optical map is 300 kbp [9], as opposed to the 15 kbp average for long reads. This quality combined with recent increased commercial availability, have lead to a rise in both data generation and tool development [31, 18, 16].

In this paper we consider Eulerian assembly applied to long reads in the presence of optical map data. In particular, we propose to use genome-wide optical maps to disentangle a de Bruijn graph constructed from long read data. In our approach we first correct sequencing errors in the long reads and then align the reads to the genome-wide optical map. This alignment information is then incorporated into the de Bruijn graph by constructing a *positional de Bruijn graph*, which is constructed from a set of *positional k-mers* (k-mer sequences with approximate positions associated with them) rather than k-mers alone [26]. Since this variant of the de Bruijn graph effectively creates a separate k-mer for each distinct occurrence of it in the genome, a space-efficient representation is vital for the graph to be constructed and used. We devise a space-efficient representation of the positional de Bruijn graph by augmenting a succinct BWT-based de Bruijn graph data structure [3]. We implement this method in a tool called KOOTA (Finnish for “assemble”).

More specifically, our contributions are as follows: (1) a new Eulerian approach for long read assembly that is based on the positional de Bruijn graph; (2) a space-efficient representation of the positional de Bruijn graph, and (3) the first long read-optical map hybrid assembler. Our experimental results demonstrate that the positional information greatly reduces the complexity of the de Bruijn graph. In this paper, we study this complexity in terms of the number of cycles in the graph – which, using standard genome assembly terminology, are referred to as *bulges* (undirected cycles) and *whirls* (directed cycles). In our experiments on *E. coli* and yeast data all bulges and more than 97% of whirls were removed from the graph when positional information was introduced.

The results of Koota on *E. coli* and yeast are compared to those of ABruijn [17] and Canu [14], two leading long read assemblers. These were selected for comparison since ABruijn [17] is currently the only other de Bruijn graph long-read assembler, and Canu has been shown to be the most memory and time efficient long read assembler [14]. Koota was competitive with respect to both memory and runtime. Further, we show that Koota achieved the lowest mismatch rate for both yeast and *E. coli*, and had a competitive genome fraction. This later statistic demonstrates that the fraction of the genome in the graph is not reduced by the removal of whirls and complexities within the graph. Thus these results show that disentangling the de Bruijn graph using positional information is a promising direction to develop an efficient and simple algorithm for long read assembly. Lastly, we note that Koota is freely available at `https://github.com/baharpan/cosmo/tree/positional`.

2 Background and Related Work

Optical Mapping

Optical mapping is a technology that generates ordered, high-resolution, restriction maps of an entire genome. Optical maps are produced by immobilizing DNA molecules on a plate and applying a restriction enzyme on the molecules. Restriction enzyme will cleave the molecules at a specific DNA pattern E. The molecules are then imaged and the length of the fragments between restriction sites can be measured from the image. An optical map of a sequence is thus a sequence $R = r_1, r_2, \dots, r_n$ where each r_i is the length of the fragment between consecutive restriction sites. Given a DNA sequence X and an enzyme recognizing the restriction site pattern E we can create an *in silico* digested optical map of it by mimicking how the enzyme cleaves the DNA molecule. Let $i_1, i_2, \dots, i_k$ be the occurrences of E in X. Then the *in silico* digested optical map of X is $M(X|E) = i_2 - i_1, i_3 - i_2, \dots, i_k - i_{k-1}$. For example if $X =$`ACGAGACGGTTACGTG` and $E =$`ACG` then the occurrences of E in X are $1, 6, 12$ and $M(X|E) = 5, 6$.

Since 2015 several methods for alignment of optical mapping data have become available, including OPTIMA [31], Maligner [18], and OMBlast [16]. Previously optical maps have been used for genome assembly in SOMA [20]. SOMA is a Eulerian assembler that uses both sequence data and optical mapping data. It builds the de Bruijn graph from short sequence reads and uses the optical map to eliminate or promote paths in the de Bruijn graph.

Long-Read Assemblers

Canu [14], HGPA [7] and MHPA [2] are long read assemblers using the OLC approach. Canu is a fork of Celera assembler [19], which uses tf-idf weighted MinHash and a sparse assembly graph construction on its overlapping strategy. HGPA uses the Celera assembler [19] for the assembly and performs self-correction of continuous long reads sequences (CLR). MHPA [2], uses a probabilistic, locality-sensitive hashing for overlapping long reads that also works along with Celera assembler [19]. Lin *et al.* [17] present an Eulerian approach to assembling long reads. Their tool, called ABruijn, constructs the A-Bruijn graph from a set of sufficiently frequent k-mers and uses path extension to derive genomic paths from short-read paths during traversal of the graph; errors are later corrected using an *OLC* approach. Lin *et al.* [17] demonstrate that in order to correctly assemble long reads, only a small number of the reads are actually needed. They extract all sufficiently frequent distinct k-mers from long sequence reads, the rationale being that those with low frequency are erroneous and those with high frequency originate from repetitive regions. Building the de Bruijn graph with this smaller set of filtered k-mers removes whirls and bulges in the resulting graph and simplifies the assembly process. The hybrid assembly using both short and long reads has also been considered for example by Pendleton et al. [22]. They combine long single-molecule and short high-throughput sequences to generate a hybrid genome assembly, which they then use to determine single nucleotide variants and structural variations.

Succinct Representations

Fundamental to our method is the succinct data structure for the positional de Bruijn graph. Although, there is a significant amount of work in constructing succinct de Bruijn graph representations – one of the first such approaches was introduced by Simpson et al. [28] as part of the development of the ABySS assembler – this is the first such representation for this de Bruijn graph variant.

Minia, by Chikhi and Rizk [6], uses a Bloom filter to store edges. They traverse the graph by generating all possible outgoing edges at each node and testing their membership in the Bloom filter. The representation that most closely reflects our work is BOSS graph representation of Bowe, Onodera, Sadakane and Shibuya [3] which is based on the Burrows-Wheeler transform [4]. More recently, Chikhi et al. [5] implemented the de Bruijn graph using an FM-index and *minimizers*.

3 Methods

In this section we present our method for disentangling the de Bruijn graph using a genome-wide optical map. We start by describing how the sequencing errors in the reads are corrected and how the long reads are aligned to the genome-wide optical map. Then we define the positional de Bruijn graph and describe methods to construct and extract reads from it.

3.1 Error Correction and Alignment of Long Reads

Before we can construct the positional de Bruijn graph, each long read must be localized on the genome-wide optical map. This involves three subprocesses: error correction, *in silico* digestion, and alignment. We first apply two rounds of the LoRMA long-read error correction method [27] to the long reads: once before, and once after aligning them to the optical map. LoRMA, which is purely long-read based, proceeds in three phases. First a de Bruijn graph based approach is used for rough correction. The regions of the reads deemed to be unrecoverable by the de Bruijn graph based method are then cut out. This process trims and splits the reads. In the final phase multiple alignments are formed between similar reads. In this work we used the intermediate reads after de Bruijn graph based correction to avoid splitting the reads.

After this first round of error correction, we create for each read r an *in silico* digested optical map $M(r|E)$ where E is the restriction site pattern(s) recognized by the restriction enzyme(s) used to build the genome-wide optical map. We then align these *in silico* digested

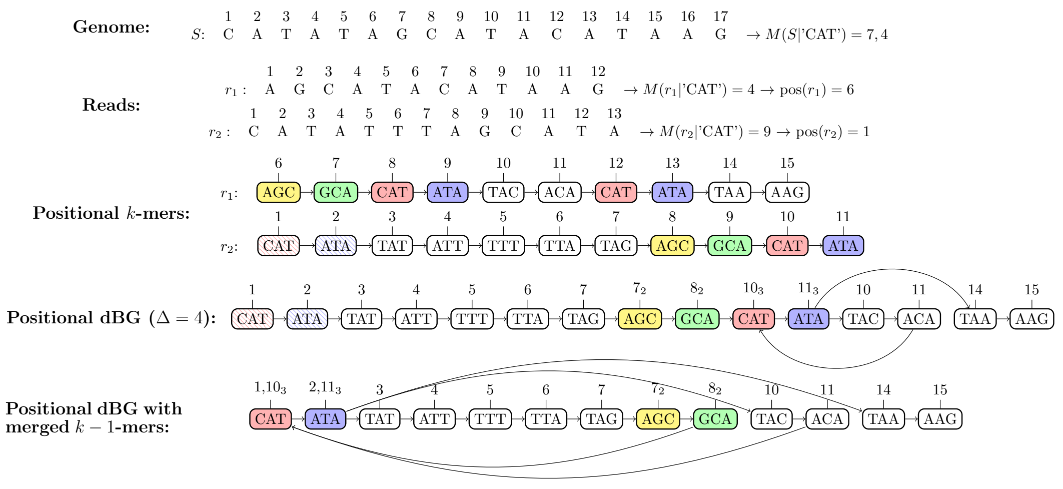

Genome:
S: C A T A T A G C A T A C A T A A G → M(S|'CAT') = 7, 4
Reads:
r1 : A G C A T A C A T A A G → M(r1|'CAT') = 4 → pos(r1) = 6
r2 : C A T A T T T A G C A T A → M(r2|'CAT') = 9 → pos(r2) = 1
Positional k-mers:
r1: AGC GCA CAT ATA TAC ACA CAT ATA TAA AAG
r2: CAT ATA TAT ATT TTT TTA TAG AGC GCA CAT ATA
Positional dBG (Δ = 4):
CAT ATA TAT ATT TTT TTA TAG AGC GCA CAT ATA TAC ACA TAA AAG
Positional dBG with merged k − 1-mers:
CAT ATA TAT ATT TTT TTA TAG AGC GCA TAC ACA TAA AAG

Figure 1 Positional de Bruijn graph $G_{4,4}$ constructed from the reads r_1 and r_2 and the genome wide optical map $M(S|\texttt{'CAT'})$. Top of the figure shows the genome sequence S and the corresponding genome wide optical map. Note that S is not available to the method but it is shown here for clarity. Our method then *in silico* digests the error corrected reads r_1 and r_2 producing optical maps $M(r_1|\texttt{'CAT'})$ and $M(r_2|\texttt{'CAT'})$. They are aligned against the genome wide optical map yielding the positions 6 and 1 for the reads r_1 and r_2, respectively. The positional $(k-1)$-mers obtained from the reads and their positions are shown in the middle. The positional de Bruijn graph is then constructed by gluing together $(k-1)$-mers whose positions are within $\Delta = 4$ of each other. The position of a glued $k-1$-mer is the average of the positions of the original $k-1$-mers. The positions of the $(k-1)$-mers are shown above the nodes and the multiplicity of the positional $(k-1)$-mer is shown as a subscript if it is greater than one. Note that the positional $(k-1)$-mers $(\texttt{AGC}, 6)$ and $(\texttt{AGC}, 8)$ are correctly glued together as they originate from the same genomic positions but are derived from different reads. On the other hand we see that although the positional $(k-1)$-mers $(\texttt{CAT}, 8)$, $(\texttt{CAT}, 12)$, and $(\texttt{CAT}, 10)$ do not all originate from the same genomic position, they are all glued together creating a small whirl in the graph. Bottom of the figure shows the positional de Bruijn graph where all positional $(k-1)$-mers with the same $(k-1)$-mer are merged to a single node with a list of positions and their multiplicities.

reads to the genome-wide optical map using the method by Valouev et al. [30]. Of the alignments returned by that method, we retain only those for which at least 40% of fragments align (this threshold was found experimentally, and reduces the number of clearly erroneous alignments reported by Valouev et al.'s software). A second round of error-correction is then applied to this subset using LoRMA. We saw superior results with the *E. coli* data (see Section 4) when we error corrected a second time, but results were not substantially improved by a third round of error correction. After these steps we have a set of error corrected reads and for each of these reads we have an approximate genomic position based on the alignment to the genome wide optical map. This set of aligned reads and their genomic positions will then be used to build the positional de Bruijn graph.

3.2 Succinct Positional de Bruijn Graph

After the alignment of the reads to the optical map, the *positional k-mers* are extracted from this alignment. The set of positional k-mers from a given read r of length n is the set of all distinct k-length substrings (k-mers) in r with a list of the positions where the substring occurred in the optical map. Thus, we can state this more formally using the notation from

Ronen et al. [26] as follows: $n-k+1$ positional k-mers $([r_1 \dots r_k], i)$, $([r_2 \dots r_{k+1}], i+1)$, $\dots$, $([r_{n-k+1} \dots r_n], i+n-k)$ can be extracted from $r = [r_1 \dots r_n]$ when aligned to i of the optical map. Thus, one k-mer can have many different positions because they can come from different reads which allows us to disambiguate k-mers at different positions. The *multiplicity* of a positional k-mer occurring at position p is defined as the number of the occurrences of that k-mer at p. We further note that the alignment to the optical map also gives the orientation of each read and thus also the orientation of each k-mer. Therefore, unlike in a de Bruijn graph without positional information, there is no need to merge a k-mer and its reverse complement which simplifies the construction and processing of the graph.

Next, we define the positional de Bruijn graph using an analogous constructive definition to the one for the de Bruijn graph [23]. Hence, to construct the positional de Bruijn graph $G_{k,\Delta}$ for a multi-set of positional k-mers and input parameter Δ a set of directed edges is constructed, which contains a directed edge $((\text{prefix}(s_k), p), (\text{suffix}(s_k), p+1))$ for each positional k-mer (s_k, p), where prefix(s_k) and suffix(s_k) are the first and last $k-1$ characters of s_k, respectively. Therefore after all edges are formed, the graph undergoes a gluing operation, where positional $(k-1)$-mers are glued together as follows. We group together positional $(k-1)$-mers having the same $(k-1)$-mer sequence and positions within Δ of each other. Such a group of m positional $(k-1)$-mers is then replaced with a single positional $(k-1)$-mer having a position equal to the average position of the group. The associated multiplicity is also stored. The definition, which is conceptually identical to that of Ronen et al. [26], is included here for completeness. We refer the interested reader to this work for another usage of this data structure.

Disambiguating identical k-mers (with positional information) should lead to a simpler graph, but an overall increase in space usage is likely because, for example, the positional de Bruijn graph will have more nodes than the plain graph, so care must be taken with graph representation.

We have implemented a space-efficient data structure for storing and traversing the positional de Bruijn graph that is based on the BOSS de Bruijn graph representation of Bowe, Onodera, Sadakane, and Shibuya [3]. We begin by briefly defining the BOSS construction of the de Bruijn graph and then demonstrate how this structure can be extended to allow positional information to be stored. The first step of constructing this graph G for a given set of k-mers is to add dummy k-mers (edges) to ensure that there exists an edge k-mer starting with first $k-1$ symbols of another edge's last $k-1$ symbols. These dummy edges ensure that each edge in G has an incoming node. After this small perturbation of the data, a list of all edges sorted into right-to-left lexicographic order of their last $k-1$ symbols (with ties broken by the first character) is constructed. We denote this list as F, and refer to its ordering as *co-lexicographic ordering.* Next, we define L to be the list of edges sorted co-lexicographically by their starting nodes with ties broken co-lexicographically by their ending nodes. Hence, two edges with same label have the same relative order in both lists; otherwise, their relative order in F is the same as their labels' lexicographic order. The sequence of edge labels (k-mers) sorted by their order in list L is called the *edge-BWT*. Now, let BF be a bit vector in which every 1 indicates the last incoming edge of each node in L, and let BL be another bit vector with every 1 showing the position of the last outgoing edge of each node in L. Given a character c and a node v with co-lexicographic rank $\mathsf{rank}(c)$, we can determine the set of v's outgoing edges using BL and then search the edge-BWT(G) for the position of edge e with label c. Using BF we can find the co-lexicographical rank of e's outgoing edge.

We augment the BOSS representation with extra information per edge to obtain a positional de Bruijn graph. In particular, we build and store a vector V of integer vectors (containing positions associated with each node). Integers in each vector are stored bit packed, using the SDSL library [10], which also provides us fast random access to individual positions. V is indexed by k-mer lexicographic rank, so that $V[i]$ is the set (vector) of positions where the ith lexicographically ranked k-mer in the input occurs. Rank operations on edge-BWT allow us to easily map from positions in edge-BWT (edges in the de Bruijn graph) to associated sets of positions in V. Figure 1 illustrates a small example of the positional de Bruijn graph representation built for a set of 4-mers and Δ=4.

3.3 Construction of the Positional de Bruijn Graph

In our implementation, we first count the k-mers and calculate their associated positions (using the positional information of each read that comes from its alignment to the optical map). After clustering the positional k-mers as described in the previous section, we write the lexicographically sorted k-mers and the associated positions in separate files (both ordered by k-mer). Each k-mer is indexed by its lexicographic rank (lexrank). We build and store a vector of position sets V, in which, $V[i]$ is the set of positions at which the k-mer with lexrank i occurs in the genome. Then we construct the BOSS representation for the k-mers such that instead of co-lexicographically sorting k-mers only, we sort (k-mer, lexrank) pairs.

To construct the F table, for each k-mer, (k-mer, lexrank) pairs will be sorted by the first $k-1$ symbols (the source node of the edge). Similarly, to construct the L table, we also sort each k-mer (without its row number) by the last $k-1$ symbols (the next node of the edge). At this step we calculate $\mathsf{F} \setminus \mathsf{L}$ (comparing only the $(k-1)$-length prefixes and suffixes respectively), and $\mathsf{L} \setminus \mathsf{F}$ to find the nodes that require incoming dummy edges and outgoing dummy edges, respectively. We will sort the set of incoming dummy edges by their first $k-1$ symbols. We call this table D. The set of outgoing edges does not require sorting. Eventually we merge D with F and $\mathsf{L} \setminus \mathsf{F}$. During the merge we push the index of each resulting edge to a vector. Afterward, while traversing the jth edge (k-mer) in the graph, the k-mer's index allows us to map to the jth element of the index vector, providing us access to the appropriate part of V containing the set of its associated positions of the k-mer. Sorting the D and F (arrays of k-mers) is the computational bottleneck in construction, and overall construction of the data structure takes $O(k|\mathsf{F}| \log |\mathsf{F}|)$ time.

3.4 Graph Traversal and Contig Recovery

A contig in the positional de Bruijn graph is a non-branching path and thus to recover the contigs it is sufficient to enumerate all non-branching paths in the graph. The following procedure is repeated until all positional k-mers in the graph have been visited. We start by picking an unvisited positional k-mer (s_k, p) and mark it as visited. We then traverse the graph both forward and backward starting from (s_k, p). Let us consider the forward traversal. In our representation we need to retrieve all out-neighbors of the k-mer s_k. We then filter the position lists of the out-neighbors to find all positional k-mers (s'_k, p') such that p' is within Δ of p. We say that these positional k-mers are consecutive to (s_k, p). If a consecutive positional k-mer is marked visited or if there are no or several consecutive positional k-mers, we have reached the end of a non-branching path and stop our traversal. If there is exactly one consecutive positional k-mer, we mark that k-mer visited and continue the traversal from that k-mer. After the forward traversal finishes, we will traverse backward from the initial k-mer which proceeds analogously to the forward traversal.

4 Results

4.1 Datasets

We simulated 92,818 PacBio reads from the reference genome of *E. coli* K-12 substr. MG 1655 with model-based simulation of PBSIM [21] using the following parameters: mean accuracy of 85 %, average read length of 10,000, and minimum read length of 1,000, and average coverage of 200x. According to observed distributions of real PacBio read length, the model-based method simulates PacBio reads with a log-normal length distribution. The average accuracy over the length of each read is taken from a normal distribution. We simulated an optical map using the reference genome for *E. coli* (str. K-12 substr. MG1655) and the enzymes XhoI, NheI and EagI since there is no publicly available one for this genome. The simulation was done by finding the locations of each restriction site in the reference genome and then *in silico* digesting at those locations.

Our second dataset consists of 220,336 sequence reads from *Saccharomyces cerevisiae* str. W303 (yeast) using data generated using PacBio RS II System and P4-C2 chemistry. The reads are available for public download from PacBio DevNet[1]. The average read length is 6,349 bp, with the minimum and maximum read length being 500 bp and 30,164 bp, respectively. Given there is no publicly available optical map for yeast, one was simulated using the reference genome of *Saccharomyces cerevisiae* str. W303 and enzymes XhoI, NheI and EagI.

4.2 The Effect of Filtering and Error Correction

As previously mentioned, in order to use long reads in the construction of the positional de Bruijn graph, they need to be aligned to the genome-wide optical map. Error correction was used to maximize the number of reads that aligned to the optical map and thus, could be used for assembly. Prior to error correction, only 30 % of the simulated *E. coli* reads aligned to the optical map, whereas 57 % of them aligned to the optical map after error correction. Of these 57 % of reads, 45 % of them had an alignment where at least 40 % of the fragments aligned. This increase in the aligned reads reflects the increase in the overall quality of the reads. The distribution of the frequency of k-mers changed dramatically with both the first and second error correction. This is illustrated in Figure 2. Prior to the first error correction a large portion of the reads had either very high frequency or very low frequency. We note that both of these sets of reads would be filtered by ABruijn. After the first error correction, alignment, and second error correction, the distribution of the k-mer frequency was much more uniform, with the majority of the k-mers having frequency between 20 and 90. Thus, as can be seen, the majority of these k-mers can be more effectively used for the assembly process by disambiguating them.

4.3 Comparison Between Assemblies

We analyzed the ability of Canu, ABruijn, and Koota to accurately assemble both datasets. As previously stated, these were selected since Canu has been shown to be more memory and time efficient than Miniasm[12], FALCON [8], SPAdes [1], and ABruijn – which only compares against Canu – is the only other de Bruijn graph long-read assembler. All assemblers were

[1] `https://github.com/P50acificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs`

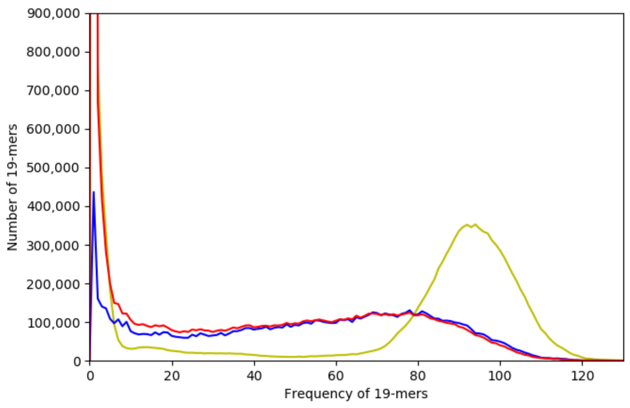


Figure 2 An illustration of the effect of error correction on number of aligned reads and 19-mer frequency on the *E. coli* data. The histogram in yellow illustrates the frequency of all distinct 19-mers in the initial corrected data. The histogram in red illustrates the frequency of all distinct 19-mers that occur in reads that underwent error correction once and aligned to the genome-wide optical map. Lastly, the histogram in blue illustrates the frequency of all 19-mers that underwent error correction twice (once prior to alignment and once afterward) and aligned to the genome-wide optical map.

run with their default parameters, on the filtered, error corrected data and using $k = 19$. Koota used $\Delta = 500$. All statistics were computed by QUAST in default mode [11]. The results demonstrate that Koota achieves the best *E. coli* assembly with respect to both genome fraction and rate of mismatches. Although the Canu and Koota *E. coli* assembly had similar genome fractions – 93.25% and 94.23%, respectively – ABruijn had a much lower genome fraction (62.94%) – and Canu had a significantly higher number of mismatches than both ABruijn and Koota. Koota's contigs had a mismatch rate of 0.37 per 100 kbp. ABruijn and Canu had a mismatch rate of 1.16 and 2.89 per 100 kbp, respectively.

Of the 220,336 reads from yeast, 95,289 (approximately 43 %) of them aligned to optical map, were error corrected a second time, and subsequently used for assembly. All the assemblies produced by Koota and ABruijn had similar genome fractions – 92 % and 93.5 %, respectively; however, ABruijn had a substantially higher mismatch rate (90.47 mismatches per 100 kbp) than Koota (2.02 mismatches per 100 kbp). The Canu assembly had a moderately higher genome fraction (95 %) in comparison to ABruijn and Koota but also a higher mismatch rate (22.9 mismatches per 100 kbp) in comparison to Koota.

4.4 Time and Memory Usage

We compared the resource usage of Koota with ABruijn and Canu on the two datasets, in particular peak memory usage, which was measured as the maximum resident set size, and run time, measured as the user process time. All experiments were performed on a 2 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz server with 512GB of RAM, and both resident set size and user process time were reported by the operating system. Again, Canu, ABruijn, and Koota were applied to long reads that had undergone error correction and filtering. Table 1 shows the memory and time usage of the three different assemblers on both the *E. coli* and yeast datasets. The assembly time of Canu was moderately less than Koota. Canu required 7 minutes and 35 seconds to assemble *E. coli* and 3 hours and 38 minutes to assemble yeast;

Table 1 Comparison between the peak memory and time usage required to assemble the (error corrected and aligned) *E. coli* reads using Koota, ABruijn, and Canu, and the rate of mismatch in their assembly. k = 19 was used for all assemblers. The peak memory is given in megabytes (MB) or gigabytes (GB). The running time is reported in seconds (s), minutes (m), and hours (h). The mismatch rate (MM) is reported per 100 kbp.

	Koota			Canu			ABruijn		
	Time	Memory	MM	Time	Memory	MM	Time	Memory	MM
E.coli	32m 20s	1.18GB	0.37	7m 35s	3.7GB	2.89	3h 46m	2.7GB	1.16
Yeast	12h 4m	4.4GB	2.02	3h 38m	3.8GB	22.9	48h 44m	15.4GB	90.47

Table 2 Comparison between the number of whirls (directed cycles) and bulges (undirected cycles) in the positional de Bruijn graph (denoted as PDBG) and the de Bruijn graph (denoted as DBG) for *E.coli* and yeast.

	Number of Whirls		Number of Bulges	
	DBG	PDBG	DBG	PDBG
E.coli	1,940	3	17,200	0
Yeast	13,223	302	59,283	0

whereas, Koota required 32 minutes and 20 seconds to assemble *E. coli* and 12 hours and 4 minutes to assemble yeast. Both Canu and Koota also used less than 5 GB of memory to assemble both yeast and *E.coli*. Lastly, as can be seen in Table 1, ABruijn required more time and memory to produce assemblies for both *E.coli* and yeast.

4.5 The Simplicity of the Positional De Bruijn Graph

Lastly, we compared the properties of the positional de Bruijn graph with the de Bruijn graph on the yeast and *E.coli* datasets. In particular, we built the positional de Bruijn graph and the de Bruijn graph using the code base for Koota by considering the graph construction with positional k-mers and (non-positional) k-mers – the former gives rise to the positional de Bruijn graph and the latter gives rise to the de Bruijn graph. In each graph we considered the in-degree, out-degree of each node, the number of bulges, and the number of whirls. Tables 2 and 3 summarize our findings. As can be seen in the following tables, taking the positional information into account in the construction of the de Bruijn graph significantly decreases the complexity of the graph and this reduction in the number of bulges becomes even more prevalent as the genome size increases. For example, the number of bulges in the de Bruijn graph for *E. coli* and yeast was 17,200 and 59,283, respectively; whereas, there exists zero bulges in the positional de Bruijn graph for these genomes. A significant reduction in the number of whirls can also be seen in Table 2. There was more than 300x and 40x more whirls in the de Bruijn graph than the positional counterpart for *E.coli* and yeast, respectively. These results mirror the results seen in Table 3.

In each case the number of nodes that have in-degree and out-degree greater than one decreased by at least two orders of magnitude. We should note that the number of nodes in the positional de Bruijn graph is guaranteed to be at least the number of nodes in the de Bruijn graph since we considered only the Koota implementation, which does not contain any graph simplification step.

Table 3 Comparison between the percentage of nodes with out-degree and in-degree greater than one in de Bruijn graph (denoted as DBG) and the positional de Bruijn graph (denoted as PDBG) in *E.coli* and yeast.

	E.coli			Yeast		
Out-degree	2	3	4	2	3	4
DBG	0.27	0.0043	0.001	0.71	0.039	0.01
PDBG	0.0079	0.00011	0.000011	0.0051	0.00038	0.00023
In-degree	2	3	4	2	3	4
DBG	0.27	0.004	0.00083	0.68	0.033	0.005
PDBG	0.01	0.00022	0.000022	0.0039	0.00031	0.00015

5 Discussion and Conclusions

Development of a production quality assembler requires sophisticated traversal algorithms, the implementation of which is well beyond the scope of this paper. Our aim in developing Koota is to demonstrate that incorporating the approximate positions of the k-mers into the de Bruijn graph construction can greatly reduce the complexity of the resulting graph. Furthermore, using space-efficient encodings, the positional information can be added without a dramatic increase in memory requirements.

Koota required the least space to assemble the simulated *E. coli* reads; 1.18 GB in comparison to the 3.7 GB required by Canu and the 2.7 GB required by ABruijn. Koota also had the highest genome fraction of the methods tested, and a low mismatch rate. Taken together these statistics show that we have not discarded a significant portion of the genome, making accurate assembly possible. For completeness we report that Koota's N50 scores are currently low (2,301 vs. 126,754 for Canu on the *E. coli* dataset), however this belies the absence of a sophisticated traversal algorithm to effectively deal with branches in the graph, and to resolve the remaining whirls. We reemphasize that our goal was not to compete with state-of-the-art assemblers, but instead to demonstrate how positional information can simplify the de Bruijn graph, in the context of long reads.

Indeed, the real influence of the optical map is its ability to disentangle the de Bruijn graph by assigning approximate positions to each of the long reads (and so the k-mers), and the addition of positions to the graph greatly reduces the number of whirls and bulges. We showed in Section 4.5 that in our experiments all bulges and over 97% of whirls disappear from the de Bruijn graphs when positional information is added to the graph.

Our main contribution has been to demonstrate the effect of adding positional information to long read assembly and how optical mapping data can assist in the assembly of long reads. Given the rarity of whirls in the positional de Bruijn graph, we expect that even slightly more sophisticated traversal algorithms would be capable of constructing 94 % or more of the *E. coli* genome with only a few contigs that have a small mismatch rate (Koota has 0.37 mismatches per 100 kbp) without using more than 1.2 GB of space. This would bridge the gap between long-read and short-read assembly since it would enable longer (more complicated genomes) to be assembled the same accuracy as short reads. A further advantage of integrating the positional information into the de Bruijn graph is that it allows for a meaningful partitioning of the graph. Each partition of the graph would contain the k-mers belonging to an interval of positions. Each of these partitions could be independently processed yielding a natural way to develop parallel or distributed algorithms for the positional de Bruijn graph.

References

1 A. Bankevich et al. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *J. Comp. Bio.*, 19(5):455–477, 2012.

2 K. Berlin et al. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotech.*, 33:623–630, 2015.

3 A. Bowe et al. Succinct de Bruijn graphs. In *Proc. WABI*, pages 225–235, 2012.

4 M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

5 R. Chikhi et al. On the representation of de Bruijn graphs. In *Proc. RECOMB*, pages 35–55, 2014.

6 R. Chikhi and G. Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms Mol. Biol.*, 8(22), 2012.

7 C.-S. Chin et al. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature Methods*, 10(6):563–569, 2013.

8 C.-S. Chin, P. Peluso, F. J. Sedlazeck, M. Nattestad, G. T. Concepcion, A. Clum, C. Dunn, and R. et al. O'Malley. Phased diploid genome assembly with single-molecule real-time sequencing. *Nat Methods*, 13:1050–1054, 2016.

9 Y. Dong et al. Sequencing and automated whole-genome optical mapping of the genome of a domestic goat (*Capra hircus*). *Nature Biotech.*, 31(2):135–141, 2013.

10 S. Gog et al. From theory to practice: Plug and play with succinct data structures. In *Proc. SEA*, pages 326–337, 2014.

11 A. Gurevich et al. QUAST: Quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

12 Li. H. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, pages 2103–2110, 2016.

13 R. M. Idury and M. S. Waterman. A new algorithm for DNA sequence assembly. *J. Comp. Bio.*, 2:291–306, 1995.

14 S. Koren et al. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, 2017. `doi:10.1101/gr.215087.116`.

15 S. Koren and A. M. Phillippy. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *Cur. Opin. Microbiol.*, 23:110–120, 2015.

16 A. K.-Y. Leung et al. OMBlast: Alignment tool for optical mapping using a seed-and-extend approach. *Bioinformatics*, 2016. To appear.

17 Y. Lin et al. Assembly of long error-prone reads using de bruijn graphs. *Proceedings of the National Academy of Sciences*, 2016. `doi:10.1073/pnas.1604560113`.

18 L. M. Mendelowitz, D. C. Schwartz, and M. Pop. MAligner: a fast ordered restriction map aligner. *Bioinformatics*, 32(7):1016–1022, 2016.

19 E. W. Myers et al. A whole-genome assembly of drosophila. *Science*, 287:2196–2204, 2000.

20 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.

21 Y. Ono, K. Asai, and M. Hamada. PBSIM: PacBio reads simulator – toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

22 M. Pendleton et al. Assembly and diploid architecture of an individual human genome via single-molecule technologies. *Nature Methods*, 12:780–786, 2015.

23 P. A. Pevzner, H. Tang, and G. Tesler. De novo repeat classification and fragment assembly. *Genome Res.*, 14(9):1786–1796, 2004.

24 P. A. Pevzner, H. Tang, and M. S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Nat. Acad. Sci.*, 98(17):9748–9753, 2001.

25 A. Rhoads and K. F. Au. PacBio sequencing and its applications. *Genomics, Proteomics & Bioinformatics*, 13(5):278–289, 2015.

26 R. Ronen, C. Boucher, H. Chitsaz, and P. Pevzner. SEQuel: Improving the accuracy of genome assemblies. *Bioinformatics*, 28(12):i188–i196, 2012.
27 L. Salmela et al. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics*, 2016. To appear.
28 J. T. Simpson and R. Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373, 2010.
29 J. T. Simpson et al. ABySS: A parallel assembler for short read sequence data. *Genome Res.*, 19(6):1117–1123, 2009.
30 A. Valouev et al. Alignment of optical maps. *J. Comp. Bio.*, 13(2):442–462, 2006.
31 D. Verzotto et al. OPTIMA: Sensitive and accurate whole-genome alignment of error-prone genomic maps by combinatorial indexing and technology-agnostic statistical analysis. *GigaScience*, 5:2, 2016.
32 D. R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

A Appendix

Figure 3 illustrates when a whirl can persist in a positional de Bruijn graph. In this example, (a) and (b) illustrate the de Bruijn graph and the positional de Bruijn graph constructed for $k = 4$ and $\Delta = 4$ and read `CTAACTAACG` that aligns to position 30 in the genome. Both the de Bruijn graph and its positional counterpart contain a whirl. The whirl in the positional de Bruijn graph is created since the occurrences of `CTAA` and `TAAC` are clustered together at positions 32 and 33, respectively, creating positional k-mers that have multiplicity greater than one. The third graph in Figure 3 shows, more typically, how positional information resolves the whirls within the graph.

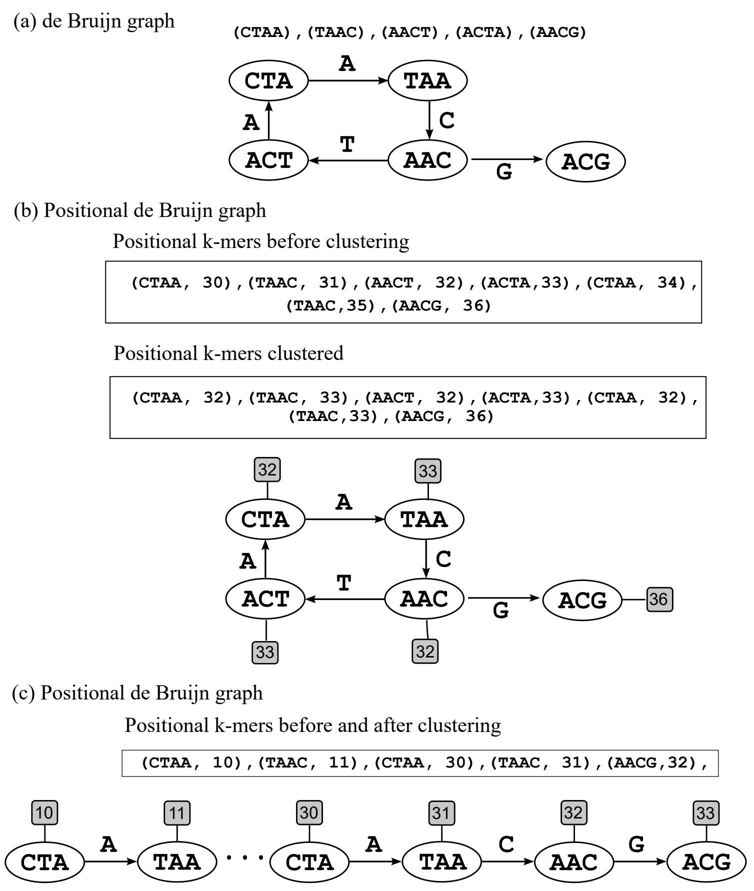

(a) de Bruijn graph
(CTAA),(TAAC),(AACT),(ACTA),(AACG)
CTA
A
TAA
A
C
ACT
T
AAC
G
ACG
(b) Positional de Bruijn graph
Positional k-mers before clustering
(CTAA, 30),(TAAC, 31),(AACT, 32),(ACTA,33),(CTAA, 34),(TAAC,35),(AACG, 36)
Positional k-mers clustered
(CTAA, 32),(TAAC, 33),(AACT, 32),(ACTA,33),(CTAA, 32),(TAAC,33),(AACG, 36)
32
33
CTA
A
TAA
A
C
ACT
T
AAC
G
ACG
36
33
32
(c) Positional de Bruijn graph
Positional k-mers before and after clustering
(CTAA, 10),(TAAC, 11),(CTAA, 30),(TAAC, 31),(AACG,32),
10
11
30
31
32
33
CTA
A
TAA
. . .
CTA
A
TAA
C
AAC
G
ACG

Figure 3 An illustration showing when a whirl in the positional de Bruijn graph can prevail. (a) shows a de Bruijn graph constructed for a read `CTAACTAACG` and $k = 4$. (b) shows the positional de Bruijn graph with constructed for a read `CTAACTAACG` whose alignment starts at position 30 of the genome, $k = 4$ and $\Delta = 4$. The set of positional k-mers before and after clustering with Δ are illustrated. (c) shows the positional de Bruijn graph with constructed for a read `CTAA..CTAACG` whose alignment starts and resumes at position 10 and 30 of the genome respectively. In this last example, $k = 4$ and $\Delta = 4$. As can be seen by these illustrations, whirls will persist in the positional de Bruijn graph for short genomic repeats when the difference between k and Δ is reasonably small since they will create positional k-mers whose multiplicity is greater than one. In (b) `CTAA` at position 32 and `TAAC` at position 33 both have multiplicity 2 after clustering.

Gene Tree Parsimony for Incomplete Gene Trees[*]

Md. Shamsuzzoha Bayzid[1] and Tandy Warnow[2]

1 Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
shams_bayzid@cse.buet.ac.bd

2 Department of Computer Science, University of Illinois Urbana-Champaign, Illinois, IL, USA
warnow@illinois.edu

Abstract

Species tree estimation from gene trees can be complicated by gene duplication and loss, and "gene tree parsimony" (GTP) is one approach for estimating species trees from multiple gene trees. In its standard formulation, the objective is to find a species tree that minimizes the total number of gene duplications and losses with respect to the input set of gene trees. Although much is known about GTP, little is known about how to treat inputs containing some *incomplete gene trees* (i.e., gene trees lacking one or more of the species). We present new theory for GTP considering whether the incompleteness is due to gene birth and death (i.e., true biological loss) or taxon sampling, and present dynamic programming algorithms that can be used for an exact but exponential time solution for small numbers of taxa, or as a heuristic for larger numbers of taxa. We also prove that the "standard" calculations for duplications and losses exactly solve GTP when incompleteness results from taxon sampling, although they can be incorrect when incompleteness results from true biological loss. The software for the DP algorithm is freely available as open source code at `https://github.com/shamsbayzid/DynaDup`.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory, J.3 Life and Medical Sciences

Keywords and phrases Gene duplication and loss, gene tree parsimony, deep coalescence

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.2

1 Introduction

The estimation of species trees is often performed by estimating multiple sequence alignments for some collection of genes, concatenating these alignments into one super-matrix, and then estimating a tree (often using maximum likelihood or a Bayesian technique) on the resultant super-matrix. However, this approach cannot be used when the species' genomes contain multiple copies of some gene, which can result from gene duplication. Since gene duplication and loss is a common phenomenon, the estimation of species trees requires a different type of approach in this case.

Gene Tree Parsimony (GTP) is an optimization problem for estimating species trees from a set of gene trees (estimated from individual gene sequence alignments). In its most typical formulations, only gene duplication and loss are considered, so that GTP depends upon two parameters: c_d (the cost for a duplication) and c_l (the cost for a loss). The two most popular versions of GTP are MGD (minimize gene duplication), for which $c_d = 1$ and $c_l = 0$, and

[*] This work was supported by US NSF (1062335) to TW and Fulbright Fellowship to SB. The authors thank Siavash Mirarab for his helpful suggestions and help in implementing the software.

MGDL (minimize gene duplication and loss), for which $c_d = c_l = 1$. The version of GTP that seeks the tree minimizing the total number of losses has also been studied; for this, $c_d = 0$ and $c_l = 1$. These variants of GTP are NP-hard optimization problems [14], but software such as DupTree [24] and iGTP [4] for GTP are in wide use.

Basic to all these problems is the ability to compute the number of duplications and losses implied by a species tree and gene tree. This problem is called the "reconciliation problem", surveyed in [7], and intensively studied in the literature (see, for example, [9, 12, 18, 16, 19, 17, 20, 27, 13, 14, 10, 28]). The mathematical formulation of the reconciliation problem was derived for the case where the gene tree and the species tree have the same set of taxa, and then extended to be able to be used on *incomplete* gene trees, i.e., trees that can miss some taxa.

Incomplete gene trees are quite common, and can arise for two different reasons: (1) *taxon sampling*: the gene may be available in the species' genome, but was not included for some reason in the dataset for that gene, or (2) *gene birth/death*: as a result of gene birth and death (true biological gene loss), the species does not have the gene in its genome.

Given a gene tree gt and a species tree ST, two formulations for the number of losses have been defined. The most commonly used one computes the number of losses by first computing the "homeomorphic subtree" $ST(gt)$ of ST induced by gt, and then computing the number of losses required to reconcile gt with $ST(gt)$ (see, for example, [12, 14, 28]). Although this second formulation is in wide use (and is the basis of both iGTP [4] and Duptree [24], two popular methods for "solving" GTP), we will show that this can be incorrect when incompleteness is due to true biological loss. We refer to this formulation as the "standard" approach because of this widespread use in both software and the theoretical literature on GTP. The other, described in [5, 23], correctly computes the number of losses when incompleteness is a result of true gene loss, as we will prove.

This paper addresses the GTP problem for the case where some of the input gene trees may be incomplete due to either sampling or true biological loss. The main results are as follows:

- We formalize the duploss reconciliation problem when gene trees are incomplete due to taxon sampling as the "optimal completion of a gene tree" (Section 2.2), and we prove (Theorem 1) that the standard calculation correctly computes losses for this case.
- We show by example that the standard calculation for losses in GTP can be incorrect when incompleteness is due to true biological loss (Section 2.3).
- We show how to compute the number of losses implied by a gene tree and species tree, when incompleteness is due to true biological loss (Section 3).
- We formulate variants of the GTP problem (when gene tree incompleteness is due to true biological loss) as minimum weight maximum clique problems (see Theorem 10 for one duploss variant), and show how to solve these problems efficiently using dynamic programming (Section 4). We show that these optimal cliques can be found in polynomial time in the number of vertices of the graph, because of the special structure of the graphs. We also show that a constrained version of these problems, where the subtree-bipartitions of the species tree are drawn from the subtree-bipartitions of the input gene trees, can be solved in time that is polynomial in the number of gene trees and taxa.

2 Basics

2.1 Notation and terminology

Throughout this paper we will assume that gene trees and species trees are rooted binary trees, with leaves drawn from the set $\mathcal{X}$ of n taxa, and we allow the gene trees to have multiple copies of the taxa, and even to miss some taxa. We let gt denote a gene tree and ST denote a species tree. We let $L(t)$ denote the set of taxa at the leaves of the tree t, and require that $L(gt) \subseteq L(ST)$. If $L(gt) = L(ST)$ we say that gt is complete, and otherwise we say that gt is incomplete.

We now define some general terminology we will use throughout this paper; other terminology will be introduced as needed. Let T be a rooted binary tree. We denote the set of vertices of T by $V(T)$, the set of edges of T by $E(T)$, the root by $r(T)$, the internal nodes by $V_{int}(T)$, and the set of taxa that appear at the leaves by $L(T)$. Note that if T is a gene tree, it can be incomplete, and so it is possible for $|L(T)|$ to be smaller than the number of leaves in T. A *clade* in T is a subtree of T rooted at a node in T, and the set of leaves of the clade is called a *cluster*. Given a node v in T, the cluster of leaves below v is denoted by $c_T(v)$, and the subtree of T rooted at v is denoted by T_v. The most recent common ancestor (MRCA) of a set A of leaves in T is denoted by $MRCA_T(A)$. Given a gene tree gt and a species tree ST, we define $\mathcal{M} : V(gt) \to V(ST)$ by $\mathcal{M}(v) = MRCA_{ST}(c_{gt}(v))$. Finally, given a node u in a rooted binary tree, we let r denote the right child of u and l denote the left child of u.

For a rooted gene tree gt and a rooted species tree ST, where $L(gt) \subseteq L(ST)$, an internal node v in gt is called a duplication node if $\mathcal{M}(v) = \mathcal{M}(v')$ for some child v' of v, and otherwise v is a speciation node [28, 12, 14, 1].

$ST(gt)$ is the homeomorphic subtree of ST induced by the taxon set of gt, and is produced as follows: ST is restricted to the taxon set of gt, and then nodes with in-degree and out-degree 1 are suppressed. $ST^*(gt)$ is the tree obtained by restricting ST to the taxon set of gt, but not suppressing nodes of in-degree and out-degree 1.

We say that clade cl in ST is a *missing clade* with respect to gt if $L(gt) \cap L(cl) = \emptyset$, and a *maximal missing clade* if it is not contained in any other missing clade. Maximal missing clades that are descendants of $\mathcal{M}(r(gt))$ are called the “lower” maximal missing clades, and those that are not descendants of $\mathcal{M}((r(gt))$ are called the “upper” maximal missing clades. We denote by $LMMC(gt, ST)$ (or $LMMC$), the set of lower maximal missing clades, and $UMMC(gt, ST)$ (or $UMMC$), the set of upper maximal missing clades. Note $UMMC(gt, ST) = \emptyset$ iff $\mathcal{M}((r(gt)) = r(ST)$.

2.2 The standard formula for computing losses

The *standard* formula (see, for example, [12, 14, 10, 28, 1]) for computing the minimum number of losses of a (potentially incomplete) gene tree gt with respect to a species tree ST is denoted $L_{std}(gt, ST)$, and is defined to be $L_{std}(gt, ST) = \sum_{u \in V_{int}(gt)} F(u, ST(gt))$, where $F(u, T)$ is defined for internal nodes u with children l and r (which can be interchanged in the formula below) by:

$$F(u,T) = \begin{cases} d(\mathcal{M}(r),\mathcal{M}(u)) + 1 & \text{if } \mathcal{M}(r) \neq \mathcal{M}(u) \ \& \ \ \mathcal{M}(l) = \mathcal{M}(u), \\ d(\mathcal{M}(l),\mathcal{M}(u)) + 1 & \text{if } \mathcal{M}(l) \neq \mathcal{M}(u) \ \& \ \ \mathcal{M}(r) = \mathcal{M}(u), \\ d(\mathcal{M}(r),\mathcal{M}(u)) & \\ +d(\mathcal{M}(l),\mathcal{M}(u)) & \text{if } \mathcal{M}(r) \neq \mathcal{M}(u) \ \& \ \ \mathcal{M}(l) \neq \mathcal{M}(u), \\ 0 & \text{if } \mathcal{M}(r) = \mathcal{M}(l) = \mathcal{M}(u). \end{cases} \tag{1}$$

where $d(s, s')$ is the number of internal nodes in T on the path from s to s'. When gt is complete, then $ST(gt) = ST$, and this formula follows from [5].

Optimal completion of a gene tree

- Input: rooted binary gene tree gt and rooted binary species tree with $L(gt) \subseteq L(ST)$.
- Output: complete gene tree $T_{samp}(gt, ST)$ that is an extension of gt such that $T_{samp}(gt, ST)$ implies a minimum number of losses with respect to ST.

In other words, we add all the missing taxa into gt (each taxon added at least once, but perhaps several times) so as to produce a complete binary gene tree that has a minimum number of losses with respect to ST. Let $L_{samp}(gt, ST) = L_{std}(T_{samp}(gt, ST), ST)$. Thus, $L_{samp}(gt, ST)$ denotes the total number of losses needed for an optimal completion of gt. Similarly, we can define $DL_{samp}(gt, ST)$ to be the total number of duplications and losses needed for a completion of gt that minimizes the duploss score.

▶ **Theorem 1.** *Given a binary rooted gene tree gt and a binary rooted species tree ST such that $L(gt) \subseteq L(ST)$, the MRCA mapping defines a reconciliation that minimizes the number of duplications, the number of losses, and hence also the total number of duplications and losses, where we treat losses as due to sampling. Furthermore, $L_{std}(gt, ST) = L_{samp}(gt, ST)$, which means the standard formula correctly computes the number of losses when we treat incompleteness as due to sampling.*

Proof omitted due to space constraints.

2.3 Incompleteness due to gene birth and death

As we will see, while the MRCA mapping is still an optimal reconciliation when gene trees are incomplete due to gene birth and death (implied from [5, 11]), the standard formula does not correctly compute the number of losses.

Consider the simple example $gt = ((a, b), c)$ and $ST = ((a, (b, d)), c)$. Under the standard formula, $L_{std}(gt, ST) = 0$, since $ST(gt) = gt$. Under the assumption that incompleteness is due to true biological loss, the genome for d does not have the gene. Because d is sister to b and all the other taxa have the gene, the gene must have been present in the parent of d, and lost on the branch leading to d. *Therefore, the standard formula for the number of losses can be incorrect when gene trees are incomplete due to gene birth and death (i.e., true biological loss).*

3 How to calculate losses

We now show how to calculate the number of losses for an incomplete gene tree gt and species tree ST, treating incomplete gene trees as due to gene birth and death. How this is defined will depend upon whether one assumes, *a priori*, that the gene is present in the genome of the common ancestor of the species in ST (i.e., at the root of ST). Thus, this section shows how to calculate the following values:

- $L^*_{bd}(gt, ST)$, the minimum number of losses, under the assumption the gene is present in the common ancestor of the species in ST ($DL^*_{bd}(gt, ST)$ is defined similarly for the total number of duplications and losses), and
- $L_{bd}(gt, ST)$ the minimum number of losses *without* assuming the gene is present in the common ancestor of the species in ST ($DL_{bd}(gt, ST)$ is defined similarly for duplications and losses).

We now show how to compute the *number* of losses (i.e., $L_{bd}(gt, ST)$ and $L^*_{bd}(gt, ST)$), using the fact that the MRCA mapping defines an optimal reconciliation.

▶ **Theorem 2.** *Let gt be a gene tree and ST a species tree such that $L(gt) \subseteq L(ST)$. Then, $L_{bd}(gt, ST) = \sum_{u \in V_{int}(gt)} F(u, ST)$, and $L^*_{bd}(gt, ST) = L_{bd}(gt, ST) + |UMMC(gt, ST)|$. Furthermore, these values can be calculated in $O(n + n')$ time, where ST has n leaves and gt has n' leaves.*

Proof. Note that we use a modification of the standard formula, $F(u, ST)$, so that we do not replace ST by $ST(gt)$ as was done in [5, 23]. The equality for L_{bd} is implied from [5] and we omit the proof concerning L_{bd} due to space constraints.

Derivation of $L^*_{bd}(gt, ST)$. By definition of $L^*_{bd}(gt, ST)$, the gene is assumed to be present at the root of the species tree ST. If $\mathcal{M}((r(gt)) = r(ST)$, then $UMMC(gt, ST) = \emptyset$, and the result follows. However, if $\mathcal{M}((r(gt)) \neq r(ST)$, the gene must be present on the path between $r(ST)$ and $\mathcal{M}((r(gt))$. Since the gene is not present in any leaf that is not below $\mathcal{M}((r(gt))$, to minimize losses, the gene must be lost on every edge off that path, since such edges lead to subtrees that do not have the gene present in any leaf. Note that if $\mathcal{M}((r(gt)) \neq r(ST)$, then the number of edges that lead off that path is $|UMMC(gt, ST)| = d(\mathcal{M}((r(gt)), r(ST)) + 1$. Since the gene must be lost on each of those edges, and the total number of losses is the sum of this value and the number of losses that occur within the subtree rooted at $\mathcal{M}((r(gt))$, it follows that $L^*_{bd}(gt, ST) = L_{bd}(gt, ST) + |UMMC(gt, ST)|$.

The running time follows easily from the fact that the MRCA mapping can be computed in linear time [8]. ◀

4 Algorithms to find species trees

Here we address the problem of finding a species tree that has a minimum total number of duplications and losses, treating incompleteness as due to true biological loss. Prior results on GTP include a branch-and-bound algorithm in [6], based on techniques from [5], a randomized hill climbing based heuristic presented in [24], a probabilistic and computationally expensive method for coestimating gene and species trees [2], and dynamic programming based solutions by Hallett and Lagergren [13], Bayzid *et al.* [1] and Chang *et al.* [3]. However, none of these works takes the reasons of incompleteness into account, and we have already shown in Sec 2.3 that the standard calculation for losses can be incorrect when incompleteness is due to true biological loss.

In this section, we derive a different approach for the GTP problems, treating incomplete gene trees as due to true biological loss (i.e., minimizing $L_{bd}(gt, ST)$ or $L^*_{bd}(gt, ST)$). The techniques we propose can be used to solve GTP exactly for small datasets, or approximately (though without any guaranteed error bounds) on larger datasets. The approach we take here is based on [1] (see also [21, 13, 26, 25], which use very similar techniques). Bayzid *et al.* [1] provided a graph-theoretic formulation for $MGDL_{std}$, whereby an optimal solution to $MGDL_{std}$ corresponded to finding a minimum weight maximum clique inside a graph called the "Compatibility Graph". The nodes of the compatibility graph correspond to "subtree-bipartitions", a concept Bayzid *et al.* [1] introduced and we will also use. [1] showed how to find a minimum weight max clique using a dynamic programming approach. We will use the same graph-theoretic formulation as in [1], but modify the weights appropriately, to show that the optimal solution to $MGDL^*_{bd}$ still corresponds to a minimum weight max clique. The DP algorithm in [1] can then be used directly to find the optimal solution to $MGDL^*_{bd}$. To achieve this, we first derive an efficient formula for $L_{bd}(gt, ST)$ (and $L^*_{bd}(gt, ST)$, similar to the one derived in [28] for $L_{std}(gt, ST)$, but somewhat more involved.

We will let $\mathcal{D}_{gt,ST}$ denote the set of duplication nodes in gt with respect to ST and $\mathcal{S}_{gt,ST}$ denote the set of speciation nodes in gt with respect to ST. When gt and ST are known, we may write these as $\mathcal{D}$ and $\mathcal{S}$. The calculation for the number of losses depends on how we interpret incompleteness in gene trees. Therefore, rather than having a single optimization problem like $MGDL$, we have variants of this problem depending on how we treat incompleteness. As shown in Theorem 1, the term $MGDL$ in the literature refers to $MGDL_{std}$, which (by Theorem 1) is identical to $MGDL_{samp}$. Here, we consider the optimization problems $MGDL^*_{bd}$, where we treat incompleteness as due to gene birth and death. And therefore, we also consider $MGDL_{bd}$, MGL^*_{bd}, and MGL_{bd}.

4.1 Basic material

4.1.1 Subtree-bipartitions

Let T be a rooted binary tree and u an internal node in T. The *subtree-bipartition* of u, denoted by $\mathcal{SBP}_T(u)$, is the unordered pair $(c_T(l)|c_T(r))$, where l and r are the two children of u. Note that subtree-bipartitions are not defined for leaf nodes. The set of subtree-bipartitions of a tree T is denoted by $\mathcal{SBP}_T = \{\mathcal{SBP}_T(u) : u \in V_{int}(T)\}$. Furthermore, any pair A and B of disjoint subsets of $\mathcal{X}$ also define a subtree-bipartition (though we may refer to these as *candidate* subtree-bipartitions to emphasize this).

Subtree-bipartition domination: Let $BP_i = (P_{i_1}|P_{i_2})$ and $BP_j = (P_{j_1}|P_{j_2})$ be two subtree-bipartitions. We say that BP_i is *dominated by* BP_j (and conversely that BP_j *dominates* BP_i) if either of the following two conditions holds: (1) $P_{i_1} \subseteq P_{j_1}$ and $P_{i_2} \subseteq P_{j_2}$, or (2) $P_{i_1} \subseteq P_{j_2}$ and $P_{i_2} \subseteq P_{j_1}$. We say that subtree-bipartition $(A|B)$ is dominated by a species tree T if one of T's subtree-bipartitions dominates $(A|B)$. Bayzid *et al.* showed that an internal node u in a gene tree gt is a duplication node with respect to a species tree ST if $\mathcal{SBP}_{gt}(u)$ is dominated by ST [1]. Finally, for a set $\mathcal{G}$ of gene trees on taxon set $\mathcal{X}$ and for any candidate subtree-bipartition $(A|B)$, we let $W_{dom}(A|B)$ be the total number of subtree-bipartitions in $\mathcal{G}$ that are dominated by $(A|B)$.

Due to space constraints, we refer to Bayzid *et al.* [1] for discussions on subtree-bipartition "domination", "containment" and "compatibility", and the compatibility graph.

4.1.2 Deep coalescence and the MDC problem

Deep coalescence (also called *incomplete lineage sorting*, or ILS) refers to the failure of alleles to coalesce (looking backwards in time) into a common ancestral allele until deeper than the most recent speciation events [15]. One of the measures for incongruence between a gene tree and a species tree under ILS is $XL(gt, ST)$, the number of extra lineages defined for the pair ST and gt [15]. For a gene tree gt and a species tree ST such that $L(gt) \subseteq L(ST)$, the number of extra lineages (summing over all edges) is defined to be $XL(gt, ST) = \sum_{e' \in E(ST^*(gt))} XL(gt, e')$, where $XL(gt, e')$ is the number of extra lineages on e'.

MDC ("minimize deep coalescence") is an optimization problem for estimating species trees in the presence of ILS. The input to MDC is a set $\mathcal{G}$ of gene trees and the output is a species tree ST such that $\sum_{gt \in \mathcal{G}} XL(gt, ST)$ is minimized. This problem is also NP-hard [28], and software for the problem exists in Phylonet [22] and iGTP [4], among others. We now describe theoretical material leading to the algorithmic approach in Phylonet [26].

▶ **Definition 3** (From [26])**.** For $B \subseteq \mathcal{X}$ and gene tree gt, we set $k_B(gt)$ to be the number of B-maximal clusters in gt, where a **B-maximal cluster** is a cluster $Y \subseteq L(gt)$ such that $Y \subseteq B$ but no other cluster of gt containing Y is a subset of B.

▶ **Definition 4.** We define $W_{xl}(x, gt)$ for x either a subtree-bipartition or a subset of $\mathcal{X}$, as follows. If $x \subseteq \mathcal{X}$, then we set $W_{xl}(x, gt) = 0$ if $x \cap L(gt) = \emptyset$ and otherwise

$W_{xl}(x, gt) = k_x(gt) - 1$. If x is a subtree-bipartition, then we let $B = p \cup q$ for $x = (p|q)$, and we set $W_{xl}(x, gt) = 0$ if $B \cap L(gt) = \emptyset$, and otherwise $W_{xl}(x, gt) = k_B(gt) - 1$. For a set $\mathcal{G}$ of gene trees and ST a species tree, we set $W_0 = \sum_{gt \in \mathcal{G}} \sum_{x \in \mathcal{X}} W_{xl}(\{x\}, gt)$.

Yu *et al.* [26] showed that for any edge e in ST, where B is the cluster below e, then $k_B(gt)$ is the number of lineages going through edge e, and so $k_B(gt) - 1$ is the number of extra lineages going through e. They defined weights on potential species tree clusters B by $W_{mdc}(B, gt) = 0$ if $B \cap L(gt) = \emptyset$ and otherwise $W_{mdc}(B, gt) = k_B(gt) - 1$ (i.e., W_{mdc} is defined for clusters while W_{xl} is defined for subtree-bipartitions), and extended this to a set $\mathcal{G}$ of gene trees by $W'_{mdc}(B) = \sum_{gt \in \mathcal{G}} W_{mdc}(B, gt)$, and then to a set C of clusters by $W''_{mdc}(C) = \sum_{B \in C} W'_{mdc}(B)$. From this, it follows easily that a set C of $n-1$ compatible clusters minimizing $W''_{mdc}(C)$ defines a rooted binary species tree with a minimum MDC score.

4.2 Deriving $L_{bd}(gt, ST)$ and $L^*_{bd}(gt, ST)$

We begin with the following theorem:

▶ **Theorem 5** (From [28]). *Let gt be a rooted binary gene tree, ST a rooted binary species tree and $\mathcal{D}$ the set of duplication nodes in gt with respect to ST. Then*

$$L_{std}(gt, ST) = XL(gt, ST(gt)) + 2|\mathcal{D}| + |V(gt)| - |V(ST(gt))|.$$

We now derive formulas for $L_{bd}(gt, ST)$ and $L^*_{bd}(gt, ST)$; to obtain formulas for $DL_{bd}(gt, ST)$ and $DL^*_{bd}(gt, ST)$, simply add $|\mathcal{D}_{gt,ST})|$.

Recall that in the definition of $F(u, T)$ given in Eqn. 1, losses are associated with internal nodes, and the total number of losses is defined as the sum of losses associated to each internal node. However, the definition of the number of losses corresponding to a node can be rewritten in terms of edges, as we now show. Let $D(s, s')$ be the number of edges in the path in ST between s and s'. Therefore, $D(s, s')$ can be defined as follows.

$$D(s, s') = \begin{cases} d(s, s') + 1 & \text{if } d(s, s') \geq 1, \\ d(s, s') & \text{if } d(s, s') = 0. \end{cases}$$

Then, for a vertex u in gt with children r and l, we can rewrite Eqn. 1 as follows:

$$F(u, ST) = \begin{cases} D(\mathcal{M}(r), \mathcal{M}(u)) + D(\mathcal{M}(l), \mathcal{M}(u)) & \text{if } \mathcal{M}(r) \neq \mathcal{M}(u) = \mathcal{M}(l), \\ (D(\mathcal{M}(r), \mathcal{M}(u)) - 1) + (D(\mathcal{M}(l), \mathcal{M}(u)) - 1) & \text{if } \mathcal{M}(u) \notin \{\mathcal{M}(l), \mathcal{M}(r)\}, \\ D(\mathcal{M}(r), \mathcal{M}(u)) + D(\mathcal{M}(l), \mathcal{M}(u)) & \text{if } \mathcal{M}(r) = \mathcal{M}(u) = \mathcal{M}(l). \end{cases}$$

It is easy to see that in all three branches of the equation above, the two terms of the sum correspond to the edges connecting u to its two children l and r. (The second term in the first branch and both terms in the third branch are 0, but we wrote them in terms of the function $D(.,.)$ for convenience.) Let $p(x)$ be the parent of x in a tree T. Therefore, we can associate gene losses to edges $e = (x, p(x))$ instead of nodes, as follows:
$\mathcal{MD}(e) = D(\mathcal{M}(x), \mathcal{M}(p(x)), and$

$$edgeloss_{ST}(e) = \begin{cases} \mathcal{MD}(e) & \text{if } p(x) \in \mathcal{D}_{gt,ST}, \\ \mathcal{MD}(e) - 1 & \text{otherwise.} \end{cases}$$

We use the subscript ST in $edgeloss_{ST}(e)$ to emphasize the fact that the distance is taken within the tree ST and not within $ST(gt)$. Note therefore $\sum_{u \in V_{int}(gt)} F(u, ST) = \sum_{e \in E(gt)} edgeloss_{ST}(e)$.

▶ **Lemma 6.** *For all gene trees gt and species trees ST with $L(gt) \subseteq L(ST)$,*

$$L_{bd}(gt, ST) = \sum_{e \in E(gt)} \mathcal{M}D(e) - |E(gt)| + 2|\mathcal{D}|, \tag{2}$$

and for a set $\mathcal{G}$ of gene trees,

$$\begin{aligned} L_{bd}(\mathcal{G}, ST) &= \sum_{gt \in \mathcal{G}} L_{bd}(gt, ST) \\ &= \sum_{gt \in \mathcal{G}} \sum_{e \in E(gt)} \mathcal{M}D(e) - \sum_{gt \in \mathcal{G}} |E(gt)| + 2 \sum_{gt \in \mathcal{G}} |\mathcal{D}_{gt,ST}|. \end{aligned} \tag{3}$$

Finally, equalities concerning $DL_{bd}(gt, ST)$ and $DL_{bd}(\mathcal{G}, ST)$ can be obtained from these equalities by adding $|\mathcal{D}_{gt,ST}|$ and $|\mathcal{D}_{\mathcal{G},ST}|$, where $|\mathcal{D}_{\mathcal{G},ST}| = \sum_{gt \in \mathcal{G}} |\mathcal{D}_{gt,ST}|$.

Proof. We partition all the non-root nodes in gt into two sets: CD (children of duplications), consisting of those nodes whose parents are duplication nodes, and CS (children of speciations), consisting of those nodes whose parents are speciation nodes. Note that every edge $(x, p(x)) \in E(gt)$ can be associated with the set containing x. Therefore,

$$\begin{aligned} L_{bd}(gt, ST) &= \sum_{e \in E(gt)} edgeloss_{ST}(e) \\ &= \sum_{x \,\in\, CD} \mathcal{M}D(x, p(x)) + \sum_{x \,\in\, CS} (\mathcal{M}D(x, p(x)) - 1) \\ &= \sum_{e \in E(gt)} \mathcal{M}D(e) - |CS|. \end{aligned} \tag{4}$$

Since each internal node has two children, clearly the number of vertices x for which $p(x)$ is a speciation node is twice the number $|\mathcal{S}|$ of speciation nodes; therefore $L_{bd}(gt, ST) = \sum_{e \in E(gt)} \mathcal{M}D(e) - 2|\mathcal{S}|$. Since each internal node is a speciation node or a duplication node, it follows that $2(|\mathcal{D}| + |\mathcal{S}|) = |E(gt)|$, and the result follows. ◀

Let $L(gt, e)$ be the number of lineages that go through edge $e \in E(ST)$; thus, $XL(gt, e) = L(gt, e) - 1$, and so

$$XL(gt, ST) = \sum_{e' \in E(ST^*(gt))} L(gt, e') - |E(ST^*(gt))|. \tag{5}$$

▶ **Lemma 7.** *For any gene tree gt and species tree ST,*
$\sum_{e \in E(gt)} \mathcal{M}D(e) = \sum_{e' \in E(ST^*(gt))} L(gt, e')$, and (by Equation 5)

$$XL(gt, ST) = \sum_{e \in E(gt)} \mathcal{M}D(e) - |E(ST^*(gt))|. \tag{6}$$

Thus, for a set $\mathcal{G}$ of gene trees and species tree ST,

$$XL(\mathcal{G}, ST) = \sum_{gt \in \mathcal{G}} XL(gt, ST) = \sum_{gt \in \mathcal{G}} \sum_{e \in E(gt)} \mathcal{M}D(e) - \sum_{gt \in \mathcal{G}} |E(ST^*(gt))|.$$

Proof. We establish the first equality, since the remaining ones follow directly from it. Consider the lists of edges in paths in ST from $\mathcal{M}(x)$ to $\mathcal{M}(p(x))$, as x ranges over the internal vertices in gt. It is easy to see that the number of occurrences of an edge $e' \in E(ST^*(gt))$ in these lists is $L(gt, e')$ (the number of lineages through e'). Also, the edges $e \in E(ST) - E(ST^*(gt))$ will not be present in these lists, since these are the edges incident on the missing clades in ST with respect to gt. Therefore, the sum of the lengths of these lists is equal to $\sum_{e \in E(gt)} \mathcal{M}D(e)$ and also equal to $\sum_{e \in ST^*(gt)} L(gt, e)$. ◀

▶ **Theorem 8.** *For all gene trees gt, sets $\mathcal{G}$ of gene trees, and species trees ST, $L_{bd}(gt, ST) = XL(gt, ST) + 2|\mathcal{D}| + |E(ST^*(gt))| - |E(gt)|$, and*

$$L_{bd}(\mathcal{G}, ST) = XL(\mathcal{G}, ST) + 2\sum_{gt\in\mathcal{G}} |\mathcal{D}_{gt,ST}| + \sum_{gt\in\mathcal{G}} (|E(ST^*(gt))| - |E(gt)|). \tag{7}$$

Proof. Follows from Lemma 6 and Lemma 7. ◀

▶ **Corollary 9.** *For all gene trees gt and species trees ST,*

$$\begin{aligned} L^*_{bd}(gt, ST) &= L_{bd}(gt, ST) + |UMMC(gt, ST)| \\ &= XL(gt, ST) + 2|\mathcal{D}_{gt,ST}| + |E(ST^*(gt))| - |E(gt)| + |UMMC(gt, ST)|. \end{aligned}$$

$$\begin{aligned} DL^*_{bd}(gt, ST) &= L_{bd}(gt, ST) + |UMMC(gt, ST)| + |\mathcal{D}_{gt,ST}| \\ &= XL(gt, ST) + 3|\mathcal{D}_{gt,ST}| + |E(ST^*(gt))| - |E(gt)| + |UMMC(gt, ST)| \end{aligned}$$

Proof. The equalities concerning L^*_{bd} follow from Thm. 2 and Thm. 8. The equalities concerning DL^*_{bd} follow by adding $|\mathcal{D}_{gt,ST}|$. ◀

4.3 Assigning weights to subtree-bipartitions

To use the graph-theoretic formulation of $MGDL^*_{bd}$, we have to assign weights to each node in the compatibility graph, $CG(\mathcal{G})$, where $\mathcal{G}$ is the input set of gene trees, so that a minimum weight clique of $n-1$ vertices defines an optimal solution to $MGDL^*_{bd}(\mathcal{G})$. We will define weights $W_{xl}(v), W_{dom}(v), W_{EC}(v)$, and $W_{MMC}(v)$ to each subtree-bipartition (i.e., node in the compatibility graph), and set

$$W_{MGDL^*_{bd}}(v) = W_{xl}(v) - 3W_{dom}(v) + W_{EC}(v) + W_{MMC}(v).$$

We then prove (see Theorem 10) that a set of $n-1$ compatible subtree-bipartitions that has minimum total weight defines a species tree that optimizes $MGDL^*_{bd}$. Note that weights $W_{xl}(v)$ and $W_{dom}(v)$ have already been defined (in Section 4.1.1 and Section 4.1.2, respectively). Hence, all that remains is to define $W_{EC}(v)$ and $W_{MMC}(v)$, and then to prove Theorem 10.

Calculating $W_{EC}(v)$ and $|E(ST^*(gt))|$

We now show how to define weight $W_{EC}(v, gt)$ for every vertex v in the compatibility graph $CG(\mathcal{G})$ so that for all species trees ST, $|E(ST^*(gt))|$ is the sum of the vertex weights for the $n-1$ clique $\mathcal{C}$ in $CG(\mathcal{G})$ corresponding to ST. To count the number of edges in $E(ST^*(gt))$, we need to exclude those edges from $E(ST)$ that are incident on a clade that is missing in gt. For a vertex v associated with the subtree-bipartition $(p|q)$, we define $W_{EC}(v, gt)$ as follows (swapping p and q as needed):

$$W_{EC}(v, gt) = \begin{cases} 0 & \text{if } p\cap L(gt) = \emptyset \text{ and } q\cap L(gt) \in \{L(gt), \emptyset\} \\ 1 & \text{if } p\cap L(gt) = \emptyset \text{ and } \emptyset \neq q\cap L(gt) \subsetneq L(gt) \\ 2 & \text{otherwise.} \end{cases} \tag{8}$$

Then, $|E(ST^*(gt))| = \sum_{u\in\mathcal{SBP}_{ST}} W_{EC}(u, gt)$. We set $W_{EC}(v) = \sum_{gt\in\mathcal{G}} W_{EC}(v, gt)$. Then, for any species tree ST and set $\mathcal{G}$ of gene trees,

$$\sum_{gt\in\mathcal{G}} |E(ST^*(gt))| = \sum_{v\in\mathcal{C}} W_{EC}(v), \tag{9}$$

where $\mathcal{C}$ is the clique in $CG(\mathcal{G})$ that corresponds to ST.

Calculating $W_{MMC}(v)$ and $|UMMC(gt, ST)|$

We now show how to assign the weight $W_{MMC}(v, gt)$ to each vertex v of the compatibility graph so that for all species trees ST, $|UMMC(gt, ST)|$ is the sum of the weights over all the vertices of the clique $\mathcal{C}$ in $CG(\mathcal{G})$ corresponding to ST. Recall that $UMMC(gt, ST)$ is the set of upper maximal missing clades in ST. For a vertex v associated with the subtree-bipartition $(p|q)$, we define $W_{MMC}(v, gt)$ as follows (swapping p and q as needed):

$$W_{MMC}(v, gt) = \begin{cases} 1 & \text{if } p \cap L(gt) = \emptyset \text{ and } q \cap L(gt) = L(gt) \text{ (or vice-versa)} \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Then $|UMMC(gt, ST)| = \sum_{u \in \mathcal{SBP}_{ST}} W_{MMC}(u, gt)$. Finally, we set $W_{MMC}(v) = \sum_{gt \in \mathcal{G}} W_{MMC}(v, gt)$. Then, for any species tree ST and set $\mathcal{G}$ of gene trees,

$$\sum_{gt \in \mathcal{G}} |UMMC(gt, ST)| = \sum_{v \in \mathcal{C}} W_{MMC}(v), \tag{11}$$

where $\mathcal{C}$ is the clique in $CG(\mathcal{G})$ that corresponds to ST.

We can extend the $MGDL^*_{bd}$ techniques to allow for losses and duplications to have different costs, as follows. Let c_d be the cost of a duplication and assume the cost of a loss (c_l) is 1. (Note that, our techniques work for any arbitrary c_d and c_l.) Let $|\mathcal{D}_{\mathcal{G},ST}| = \sum_i^k |\mathcal{D}_{gt_i,ST}|$, and set $DL^*_{bd}(\mathcal{G}, ST, c_d) = c_d * |\mathcal{D}_{\mathcal{G},ST}| + L^*_{bd}(\mathcal{G}, ST)$. Let $MGDL^*_{bd}(\mathcal{G}, c_d)$ be the problem that takes a set $\mathcal{G}$ of gene trees and duplication cost c_d as input, and finds the species tree that minimizes the weighted duploss score $DL^*_{bd}(\mathcal{G}, ST, c_d)$. Let $W^{c_d}_{MGDL^*_{bd}}(v) = W_{xl}(v) - (c_d + 2)W_{dom}(v) + W_{EC}(v) + W_{MMC}(v)$. (If $c_d = 1$, we omit the superscript c_d and write $W_{MGDL^*_{bd}}(v)$.)

▶ **Theorem 10.** *Let $\mathcal{G} = \{gt_1, gt_2, \ldots, gt_k\}$ be a set of binary rooted gene trees on set $\mathcal{X}$ of n species, and set the weights on the vertices in the compatibility graph using $W^{c_d}_{MGDL^*_{bd}}(v)$.* (a) *A set of subtree-bipartitions in an $(n-1)$-clique of minimum weight in $CG(\mathcal{G})$ defines a binary species tree ST that minimizes $DL^*_{bd}(\mathcal{G}, ST, c_d)$. Furthermore, the weighted duploss score of ST is given by $W_0 + W^{c_d}_{MGDL^*_{bd}}(\mathcal{C}) + c_d(N-k)$, where $N = \sum_{i=1}^k n_i$.* (b) *If we reset the weights to be $W_{MGL^*_{bd}}(v) = W_{MGDL^*_{bd}}(v) + W_{dom}(v)$, then a set of subtree-bipartitions in an $(n-1)$-clique of minimum weight in $CG(\mathcal{G})$ defines a binary species tree ST that minimizes $L^*_{bd}(\mathcal{G}, ST)$.*

Proof. We prove (a), since (b) follows directly from (a). Let $\mathcal{C}$ be a clique of size $n-1$ in $CG(\mathcal{G})$ and ST the associated species tree. Let $\mathcal{SBP}_{dom}(gt, ST)$ be the set of subtree-bipartitions in gt that are dominated by a subtree-bipartition in ST. Note that $|\mathcal{SBP}_{dom}(gt, ST)|$ is the number of speciation nodes in gt with respect to ST [1]. Therefore, the total number of speciation nodes in $\mathcal{G}$ is $\sum_{i=1}^k |\mathcal{SBP}_{dom}(gt_i, ST)| = \sum_{v \in V_{int}(ST)} W_{dom}(v)$. Also, $\sum_{v \in \mathcal{C}} W_{xl}(v) = \sum_{i=1}^k XL(gt_i, ST)$, and $\sum_{i=1}^k |\mathcal{D}_{gt_i,ST}| = \sum_{i=1}^k (n_i - 1) - \sum_{v \in \mathcal{C}} W_{dom}(v)$, where n_i is the number of leaves in gt_i. Finally, since all gene trees are rooted binary trees, $|E(gt_i)| = 2n_i - 2$ and $|V_{int}(gt_i)| = n_i - 1$. Recall that W_0 is the number of extra lineages contributed by the leaf set of the species tree (Definition 4). Therefore,

$$\begin{aligned}
DL^*_{bd}(\mathcal{G}, ST, c_d) &= \sum_{i=1}^{k}(c_d * |\mathcal{D}_{gt_i,ST}| + L^*_{bd}(gt_i, ST)) \\
&= \sum_{i=1}^{k}[XL(gt_i, ST) + (c_d+2)|\mathcal{D}_{gt_i,ST}| + \ |UMMC(gt_i, ST)| \\
&\quad + |E(ST^*(gt_i))| - |E(gt_i)|] \text{ (by Cor. 9)} \\
&= W_0 + \sum_{v\in\mathcal{C}} W_{xl}(v) + \sum_{i=1}^{k}(c_d+2)(n_i-1) - (c_d+2)\sum_{v\in\mathcal{C}} W_{dom}(v) \\
&\quad + \sum_{v\in\mathcal{C}} W_{MMC}(v) \ + \sum_{v\in\mathcal{C}} W_{EC}(v) - \sum_{i=1}^{k}(2n_i - 2) \text{ (by Eqns. 9 and 11.)} \\
&= W_0 + W^{c_d}_{MGDL^*_{bd}}(\mathcal{C}) + c_d(N-k).
\end{aligned}$$

Note that W_0 does not depend on the topology of the species tree. Hence, the $(n-1)$-clique $\mathcal{C}$ with minimum weight defines a tree ST that minimizes $DL^*_{bd}(\mathcal{G}, ST, c_d)$. The proof for (b) follows trivially. ◀

4.4 Dynamic programming algorithm

Let $\mathcal{SBP}$ be a set of subtree-bipartitions, with $\mathcal{SBP}$ equal to all possible subtree-bipartitions if an exact solution is desired, and otherwise a proper subset if a faster algorithm is desired or necessary. We present the DP algorithm for the $MGDL^*_{bd}(\mathcal{G}, c_d)$ problem. We compute $score(A)$ in order, from the smallest cluster to the largest cluster $\mathcal{X}$.

Algorithm $MGDL^*_{bd}(\mathcal{G}, c_d)$

```
if $|A| = 1$ then $score(A) = W_{XL}(A)$
else
$score(A) = max\{score(A_1) + score(A - A_1) + W^{c_d}_{MGDL^*_{bd}}(A_1|A - A_1) : (A_1|A - A_1) \in \mathcal{SBP}\}$
```

If there is no $(A_1|A - A_1) \in \mathcal{SBP}$, we set its $score(A)$ to $-\infty$, signifying that A cannot be further resolved. At the end of the algorithm, if $\mathcal{SBP}$ includes at least one clique of size $n-1$, we have computed $score(\mathcal{X})$ as well as sufficient information to construct the optimal set of compatible clusters and hence the optimal species tree (subject to the constraint that all the subtree bipartitions in the output tree are in $\mathcal{SBP}$). If subtree bipartitions in $\mathcal{SBP}$ are not sufficient for building a fully resolved tree on $\mathcal{X}$, then $score(\mathcal{X})$ will be $-\infty$, and our algorithm returns FAIL.

The running time is $O(n|SBP|^2)$. The optimal number of duplications and losses is given by $score(\mathcal{X}) + c_d(N-k)$, by Theorem 10. If $\mathcal{SBP}$ contains all possible subtree-bipartitions, we have an exact but exponential time algorithm. However, if $\mathcal{SBP}$ contains only those subtree-bipartitions from the input gene trees, then the algorithm finds the optimal constrained species tree in time that is polynomial in the number of gene trees and taxa.

4.5 Extensions

It is trivial to extend the theory for $MGDL^*_{bd}$ and MGL^*_{bd} to $MGDL_{bd}$ and MGL_{bd}, as we now show. Recall that $L_{bd}(gt, ST) = L^*_{bd}(gt, ST) - |UMMC(gt, ST)|$ and that

$DL_{bd}(gt, ST) = DL^*_{bd}(gt, ST) - |UMMC(gt, ST)|$. Therefore, to extend the algorithmic approach to solve MGL_{bd} and $MGDL_{bd}$, we define $W_{MGL_{bd}}(v, gt) = W_{MGL^*_{bd}}(v, gt) - W_{MMC}(v, gt)$ and $W_{MGDL_{bd}}(v, gt) = W_{MGDL^*_{bd}}(v, gt) - W_{MMC}(v, gt)$, and then seek a minimum weight maximum clique in the compatibility graph with these modified weights.

5 Conclusion

In this paper we investigated how different reasons for gene tree incompleteness affects the mathematical formulation of gene loss. We present the first mathematical formulation to model gene loss due to true biological loss, and distinguish this from incompleteness due to taxon sampling. We proposed exact and heuristic algorithms to infer species trees from a set of incomplete gene trees by minimizing gene duplications and losses when the incompleteness is due to true biological loss.

References

1 M. S. Bayzid, S. Mirarab, and T. Warnow. Inferring optimal species trees under gene duplication and loss. In *Proc. of Pacific Symposium on Biocomputing (PSB)*, volume 18, pages 250–261, 2013.

2 B. Boussau, G. J. Szöllősi, L. Duret, M. Gouy, E. Tannier, and V. Daubin. Genome-scale coestimation of species and gene trees. *Genome research*, 23(2):323–330, 2013.

3 W. C. Chang, A. Wehe, P. Górecki, and O. Eulenstein. Exact solutions for classic gene tree parsimony problems. In *Proc. of the 5th Int. Conf. on Bioinformatics and Computational Biology*, pages 225–230, 2013.

4 R. Chaudhary, M. S. Bansal, A. Wehe, D. Fernández-Baca, and O Eulenstein. iGTP: a software package for large-scale gene tree parsimony analysis. *BMC Bioinf.*, pages 574–574, 2010.

5 C. Chauve, J. P. Doyon, and N. El-Mabrouk. Gene family evolution by duplication, speciation, and loss. *J. Comp. Biol.*, 15(8):1043–1062, 2008.

6 J. P. Doyon and C. Chauve. Branch-and-bound approach for parsimonious inference of a species tree from a set of gene family trees. *Adv. Exp. Med. Biol.*, 696:287–295, 2011.

7 J. P. Doyon, V. Ranwez, V. Daubin, and V. Berry. Models, algorithms and programs for phylogeny reconciliation. *Brieif. Bioinf.*, 12(5):392–400, 2011.

8 H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proc. 15th ACM Symp. Theory of Comp. (STOC)*, pages 246–251, 1983.

9 M. Goodman, J. Czelusniak, G. Moore, E. Romero-Herrera, and G. Matsuda. Fitting the gene lineage into its species lineage: a parsimony strategy illustrated by cladograms constructed from globin sequences. *Syst. Zool.*, 28:132–163, 1979.

10 P. Górecki. Reconciliation problems for duplication, loss and horizontal gene transfer. In *Proc. 8th Ann. Int. Conf. on Computational Molecular Biology*, pages 316 – 325, 2004.

11 P. Górecki and J. Tiuryn. DLS-trees: A model of evolutionary scenarios. *Theor. Comput. Sci.*, 359(8):378–399, 2006.

12 R. Guigo, I. Muchnik, and T. Smith. Reconstruction of ancient molecular phylogeny. *Mol. Phylog. and Evol.*, 6(2):189–213, 1996.

13 M. T. Hallett and J. Lagergren. New algorithms for the duplication-loss model. In *Proc RECOMB*, pages 138–146, 2000.

14 B. Ma, M. Li, and L. Zhang. From gene trees to species trees. *SIAM J. on Comput.*, 30(3):729–752, 2000.

15 W. P. Maddison. Gene trees in species trees. *Syst Biol*, 46:523–536, 1997.

16 B. Mirkin, I. Muchnik, and T. Smith. A biologically consistent model for comparing molecular phylogenies. *J. Comput. Biol.*, 2(4):493–507, 1995.

17 R. Page and M. Charleston. Reconciled trees and incongruent gene and species trees. In B. Mirkin, F. R. McMorris, F. S. Roberts, and A. Rzehtsky, editors, *Mathematical hierarchies in biology*, volume 37. American Math. Soc., 1997.

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

19 R. D. M. Page. GeneTree: comparing gene and species phylogenies using reconciled trees. *Bioinformatics*, 14(9):819–820, 1998. `doi:10.1093/bioinformatics/14.9.819`.

20 U. Stege. Gene trees and species trees: The gene-duplication problem is fixed-parameter tractable. In *Proc. of the 6th Int. Workshop on Algorithms and Data Structures (WADS'99)*, pages 166–173, 1999.

21 C. V. Than and L. Nakhleh. Species tree inference by minimizing deep coalescences. *PLoS Comp. Biol.*, 5(9), 2009.

22 C. V. Than, D. Ruths, and L. Nakhleh. PhyloNet: A software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC Bioinf.*, 9:322, 2008.

23 B. Vernot, M. Stolzer, A. Goldman, and D. Durand. Reconciliation with non-binary species trees. *J. Comp. Biol.*, 15(8):981–1006, 2008.

24 A. Wehe, M. S. Bansal, J. G. Burleigh, and O. Eulenstein. Duptree: A program for large-scale phylogenetic analyses using gene tree parsimony. *Amer. Jour. Bot.*, 24(13):1540–1541, 2008.

25 Y. Yu, T. Warnow, and L. Nakhleh. Algorithms for MDC-based multi-locus phylogeny inference. In *Proc. RECOMB*, 2011.

26 Y. Yu, T. Warnow, and L. Nakhleh. Algorithms for MDC-based multi-locus phylogeny inference: Beyond rooted binary gene trees on single alleles. *J. Comp. Biol.*, 18(11):1543–1559, 2011.

27 L. Zhang. On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies. *J. Comp. Biol.*, 4(2):177–188, 1997.

28 L. Zhang. From gene trees to species trees II: Species tree inference by minimizing deep coalescence events. *IEEE/ACM Trans. Comp. Biol. Bioinf.*, 8(9):1685–1691, 2011.

Better Greedy Sequence Clustering with Fast Banded Alignment[*†]

Brian Brubach[1], Jay Ghurye[2], Mihai Pop[3], and Aravind Srinivasan[4]

1 **Department of Computer Science, University of Maryland, College Park, USA**
`bbrubach@cs.umd.edu`
2 **Department of Computer Science, University of Maryland, College Park, USA**
`jayg@cs.umd.edu`
3 **Department of Computer Science, University of Maryland, College Park, USA**
`mpop@umiacs.umd.edu`
4 **Department of Computer Science, University of Maryland, College Park, USA**
`srin@cs.umd.edu`

Abstract

Comparing a string to a large set of sequences is a key subroutine in greedy heuristics for clustering genomic data. Clustering 16S rRNA gene sequences into operational taxonomic units (OTUs) is a common method used in studying microbial communities. We present a new approach to greedy clustering using a trie-like data structure and Four Russians speedup. We evaluate the running time of our method in terms of the number of comparisons it makes during clustering and show in experimental results that the number of comparisons grows linearly with the size of the dataset as opposed to the quadratic running time of other methods. We compare the clusters output by our method to the popular greedy clustering tool UCLUST. We show that the clusters we generate can be both tighter and larger.

1998 ACM Subject Classification B.2.4 Algorithms

Keywords and phrases Sequence Clustering, Metagenomics, String Algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.3

1 Introduction

The problem of comparing a string against a large set of sequences is of central importance in domains such as computational biology, information retrieval, and databases. Solving this problem is a key subroutine in many *greedy* clustering heuristics, wherein we iteratively choose a cluster center and form a cluster by recruiting all strings which are similar to the center. In computational biology, sequence similarity search is used to group biological sequences that are closely related. We will use this domain as a motivating example throughout the paper.

Traditionally, clustering 16S rRNA gene [11] sequences involved building a multiple sequence alignment of all sequences, computing a pairwise distance matrix of sequences based on the multiple sequence alignment, and clustering this matrix [17]. However, finding the best multiple sequence alignment is computationally intractable and belongs to the class of

* AS and BB were supported in part by NSF Awards CNS 1010789 and CCF 1422569. MP and BB were supported in part by the NIH, grant R01-AI-100947 to MP. MP and JG were supported in part by the Bill and Melinda Gates Foundation (PI Jim Nataro, subcontract to MP). AS was supported in part by a research award from Adobe, Inc.
† The authors wish to thank the anonymous reviewers for their helpful comments.

NP-hard problems[16]. Another naive way of clustering sequences is to perform all-versus-all comparison to compute a similarity metric such as edit distance and perform hierarchical clustering to merge closely related sequences together. However, the resulting running time is at least quadratic in the total number of sequences. With the development of faster and cheaper DNA sequencing technologies, metagenomic sequencing datasets can contain over 1 billion short reads [2]. At this scale, both strategies can prove to be very expensive and take months to generate clusters. To counter this, heuristic-based methods like greedy clustering are used. While these methods can have worst case quadratic running time, they can run faster in practice[8, 3, 5].

Here, we show a new method for reducing that worst case quadratic running time in practice when the distance metric is the Levenshtein distance [7] and similarity is determined by a maximum distance of d. Our algorithm improves the speed of the recruitment step wherein we seek all strings within d distance of a chosen center. In addition to promising experimental results, we give slightly weaker, but provable, guarantees for our techniques while many existing methods do not. Finally, we analyze the quality of the clusters output by our method in comparison to the popular greedy clustering tool UCLUST. We show that the clusters we generate can be both tighter and larger.

1.1 Related Work

The problem of comparing a query string against a large string database has been widely studied for at least the past twenty years. For similarity metrics like the edit distance, a dynamic programming algorithm [14] can be used to compare two sequences in $O(m^2)$ time, where m is the length of the sequences. When we only wish to identify strings which are at most edit distance d apart, the running time for each comparison can be reduced to $O(md)$ [12] using a modified version of the standard dynamic programming algorithm. This type of sequence alignment is referred to as *banded alignment* in the literature since we only need to consider a diagonal "band" through the dynamic programming table. The simple dynamic programming approach can also be sped up by using the Four Russians method [10, 9], which divides the alignment matrix into small square blocks and uses a lookup table to perform the alignment quickly within each block. This brings the running time down to $O(m^2 \log(\log(m))/\log(m))$ and $O(m^2/\log m)$ for arbitrary and finite alphabets, respectively. Myers [13] considered the similar problem of finding all locations at which a query string of length m matches a substring of a text of length M with at most d differences. They used the bit vector parallelism in hardware to achieve a running time of $O(mM/w)$ where w is the machine word size. However, when used for clustering sequences, these methods need to perform pairwise comparison of all sequences, thereby incurring the high computational cost of $O(n^2)$ comparisons where n is the total number of sequences.

Sequence search against a database is a crucial subroutine in sequence clustering in general and greedy clustering in particular. In greedy approaches, we choose some sequences to be cluster centers and clusters are formed by recruiting other sequences which are similar to the centers. Depending on the approach, we may compare a sequence to recruit against a set of cluster centers or compare a single cluster center against all other sequences, recruiting those within some specified distance. The comparison can be done using any of the methods mentioned above, but in the worst case, existing algorithms may need to perform all pairs banded alignment resulting in $O(n^2md)$ running time on arbitrary input data. However, the interesting property of sequencing data is that most of the sequences generated by the experiments are highly similar to each other. To exploit sequence similarity and reduce the computation performed in dynamic programming, the DNACLUST [5] algorithm lexicographically sorts the sequences and compares sequences against the center sequence in sorted order. Since the adjacent sequences in sorted order share a long prefix, the part of the dynamic programming table corresponding to their longest common prefix remains unchanged, allowing the "free" reuse of that part of the table for further alignments. This method fails when two sequences differ at the start but are otherwise similar. In this case, the entire dynamic programming table needs to be recomputed. The UCLUST [3] algorithm uses the USEARCH [3] algorithm to compare a query sequence against a database of cluster centers. However, the algorithm used by UCLUST makes several heuristic choices in order to speed up the calculation of clusters and thus, the resulting clusters are not guaranteed to satisfy any specific requirement. For example, the distances between sequences assigned to the same cluster should ideally satisfy triangle inequality (ensuring that the cluster diameter is at most twice the radius) and the cluster diameters should be both fairly uniform and within the bounds specified by the user.

1.2 Preliminaries

Let the multiset $\mathcal{S}$ be the set of n sequences to be clustered. Let m be the maximum length of any sequence in $\mathcal{S}$. For simplicity of exposition and analysis, we will assume throughout most of this paper that all sequences have length exactly m. We also assume m is much smaller than n.

1.2.1 Distance metric

We use the same edit distance-based similarity metric as DNACLUST[5], namely

$$\text{similarity} = 1 - \frac{\text{edit distance}}{\text{length of the shorter sequence}}$$

Here, we define edit distance to be Levenshtein distance with uniform penalties for insertions, deletions, and substitutions. The "length of the shorter sequence" refers to the original sequences' lengths without considering gaps inserted by the alignment. We say that two sequences are *similar* if their alignment meets or exceeds a given similarity threshold. Let d be the maximum edit distance between two sequences aligned to the same cluster. This distance is usually computed from a similarity threshold provided by the user, e.g., 97%. Both of our algorithms will be performing *banded alignment* with d as the maximum allowable distance. In this case, if we determine that two sequences have distance greater than d, we need not report their actual distance.

1.2.2 Intervals

Our algorithm involves dividing each sequence into overlapping substrings of length k at regular *intervals*. We formalize the definition of an interval as follows. Given a *period length* p such that $k = p + d + 1$, we divide each sequence into $\lfloor m/p \rfloor$ intervals of length k. For $i \in \{0, 1, \ldots, \lfloor m/p \rfloor - 1\}$, the i^{th} interval starts at index ip inclusive and extends to index $ip + k$ exclusive. We will see in Section 2.1.2 that we must choose p to be at least d. However, choosing a larger p may give a better speedup when dealing with highly similar sequences. Further, for an interval i, we define b_i to be the number of *distinct* substrings for interval i over all sequences in $\mathcal{S}$ and we define $b = \max_i b_i$. We will show in Section 2.1.3 that when b is much smaller than n we get some theoretical improvement on the running time. Figure 1 shows an example of how a sequence is partitioned into a set of overlapping substrings. We

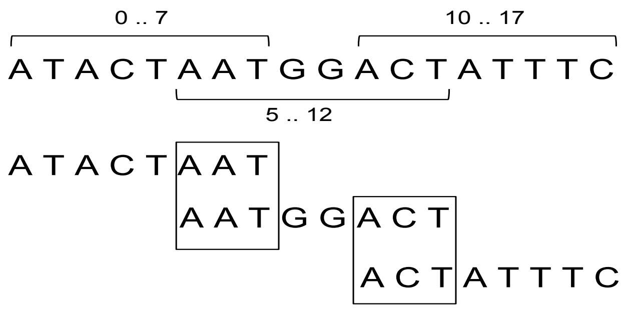


Figure 1 An example of how a string is divided in overlapping substrings called intervals. In this case, the length of each substring (k) is 8. Since the substrings must overlap by $d + 1$ characters, which in this case is 3, the period length (p) is 5.

store these intervals in a data structure we call an *Edit Distance Interval Trie (EDIT)* which is described in detail in Section 2.2.

1.2.3 Greedy clustering

The greedy clustering approach (similar to CD-HIT[8], UCLUST, and DNACLUST) can be described at a high level as follows. De-replicate the multiset $\mathcal{S}$ to get the set $\mathcal{U}$ of distinct sequences. Optionally, impose some ordering on $\mathcal{U}$. Then, iteratively remove the top sequence from $\mathcal{U}$ to form a new cluster center s_c. Recruit all sequences $s \in \mathcal{U}$ that are within d distance from s_c. When we recruit a sequence s, we remove it from $\mathcal{U}$ and add it to the cluster centered at s_c. If s_c does not recruit any sequences, we call it a singleton and add it to a list of singletons, rather than clusters. We continue this process until $\mathcal{U}$ is empty.

We order the sequences of $\mathcal{U}$ in decreasing order of their abundance/multiplicity in $\mathcal{S}$. This is also the default ordering used by UCLUST. Alternatively, DNACLUST uses decreasing order of sequence length. The reason for ordering by abundance is that assuming a random error model, the abundant sequences should be more likely to be "true" centers of a cluster. The reason for DNACLUST ordering by length is to preserve triangle inequality among sequences in a cluster when performing semi-global alignment allowing gaps at the end with no penalty. Semi-global alignment is necessary for comparing reads generated by specific sequencing technologies such as 454. However, since we perform global alignment, triangle inequality is guaranteed regardless of the ordering and thus, ordering by abundance is preferred.

1.3 Our Contributions

We developed a method for recruiting in exact greedy clustering inspired by the classical Four Russians speedup. In Section 2, we describe our algorithm and prove that the worst case theoretical running time is better than naive all-versus-all banded alignment under realistic assumptions on the sequencing data used for clustering. In section 3, we present experimental results from using our method to cluster a real 16S rRNA gene dataset containing about 2 million distinct sequences. We show that on real data the asymptotic running time of the algorithm grows linearly with the size of the input data. We also evaluated the quality of the clusters generated by our method and compared it with UCLUST, which is one of the widely used methods. We show that our method generates tighter and larger clusters at 99% similarity both when considering edit distance and evolutionary distance. At 97%

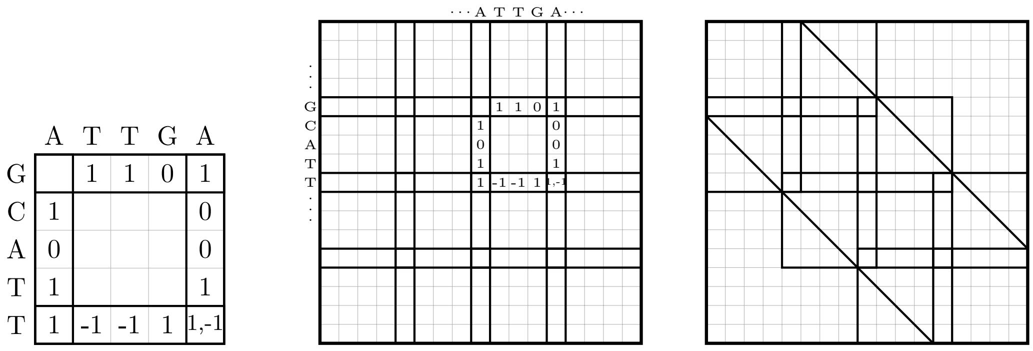


Figure 2 Example of classic Four Russians. **Left:** a single block. Notice that for any input in the upper left corner, we can sum that value with one path along the edges of the block to recover the value in the lower right corner. Note that the offset value in the lower right corner may be different for the row and column vectors overlapping at that cell. In this case, the lower right cell is one more than its left neighbor and one less than its above neighbor. **Center:** the full dynamic programming table divided into nine 5×5 blocks. Note that the offset values in the example block may not correspond to the optimal alignment of the two substrings shown since they depend on the global alignment between the two full length strings. **Right:** blocks covering only the diagonal band in the context of banded alignment.

similarity, we show that the our method produces clusters with a much tighter edit distance diameter compared to UCLUST. While UCLUST runs faster at similarities 97% and less, our approach is faster at higher similarities. In particular, we highlight that UCLUST does not scale linearly at the 99% similarity threshold while our approach does.

2 Recruiting algorithm

We show two ways in which the classical Four Russians speedup can be adapted to banded alignment. Then, we describe a trie-like data structure for storing sequences. Finally, we use this data structure to recruit similar sequences to a given center sequence using our Four Russians method.

2.1 Banded Four Russians approach

We present two ways to extend the Four Russians speedup of edit distance computation to banded alignment. The first is a very natural extension of the classical Four Russians speedup. The second is useful for tailoring our algorithm to meet the needs of 16S rRNA gene clustering. Specifically, we exploit the fact that the strings are similar and the maximum edit distance is small.

2.1.1 Warm-up: classic Four Russians speedup

In the classical Four Russians speedup of edit distance computation due to [10, 9], the dynamic programming table is broken up into square *blocks* as shown in the center of Fig. 2. These blocks are tiled such that they overlap by one column/row on each side (for a thorough description of this technique see [6]). When computing banded alignment, we only need to tile the area within the band as in the righthand of Fig. 2. Let the maximum edit distance be d and the string lengths be m. Then our block size k can be as small as $d + 1$ and we require roughly $2m/k$ blocks in total.

The high level idea of the Four Russians speedup is to precompute all possible solutions to a *block function* and store them in a lookup table (In our implementation we use lazy computation and store the lookups in a hash table instead of precomputing for all inputs). The block function takes as input the two substrings to be compared in that block and the first row and column of the block itself in the dynamic programming table. It outputs the last row and column of the block. We can see in the Fig. 2 that given the two strings and the first row and column of the table, such a function could be applied repeatedly to compute the lower right cell of the table and therefore, the edit distance. Note that cells outside the band will not be used since any alignment visiting those cells must have distance larger than d.

There are several tricks that reduce the number of inputs to the block function to bound the time and space requirements of the lookup table. For example, the input row and column for each block can be reduced to vectors in $\{-1, 0, 1\}^d$. These *offset vectors* encode only the difference between one cell and the next (see Fig. 2) which is known to be at most 1 in the edit distance table. It has also been shown that the upper left corner does not need to be included in the offset vectors. This bounds the number of possible row and column inputs at 3^d each [10].

Notice that for the banded alignment problem, this may not provide any speedup for comparing just two strings of length m. Indeed, building and querying the lookup table may take more time than simply running the classical dynamic programming algorithm restricted to the band of width $2d + 1$. However, our final algorithm will do many such comparisons between different pairs of strings using the same lookup table. In practice, we also populate the lookup table as needed rather than pre-computing it. This technique, known as *lazy computation*, allows us to avoid adding unnecessary entries for comparisons that don't appear in our dataset. Additionally, decomposing sequences into blocks will be a crucial step in building the data structure in Section 2.2.

2.1.2 Our approach to the Four Russians speedup

Notice that the previous approach will not offer much benefit in practice when d is small (e.g. $d = 2$). The overhead of looking up block functions and stitching them together may even be slower than simply running dynamic programming on a block. Further, our dataset may not require us to build a lookup table comparing all possible strings of length k.

Here we consider a different block function. This function is designed for situations in which we wish to use a block size k that is larger than $d + 1$. The blocks now overlap on a square of size $d + 1$ at the upper left and lower right corners. We will call these overlapping regions *overlap squares.* Our block function now takes as input the two substrings to be compared and the first row and column of the the upper left overlap square. It outputs the first row and column of the lower right overlap square as well as the difference between the upper left corners of the two overlap squares.

Thus, we can move directly from one block to the next, storing a sum of the differences between the upper left corners. In this case, reaching the final lower right cell of the table requires an additional $O(d^2)$ operation to fill in the last overlap square, but this adds only a negligible factor to the running time.

This approach succeeds when the number of possible substring inputs to the block function is limited by some properties of the dataset as opposed to an absolute theoretical upper bound such as $O(|\sigma|^k)$ based on the number of possible strings of length k for an alphabet σ. Rather than computing and storing all possible inputs, we simply store the inputs encountered by our algorithm. The advantage is that a larger block size reduces the number of lookups needed to compare two strings which is $m/(k - d - 1)$ for this approach. Naturally, the same

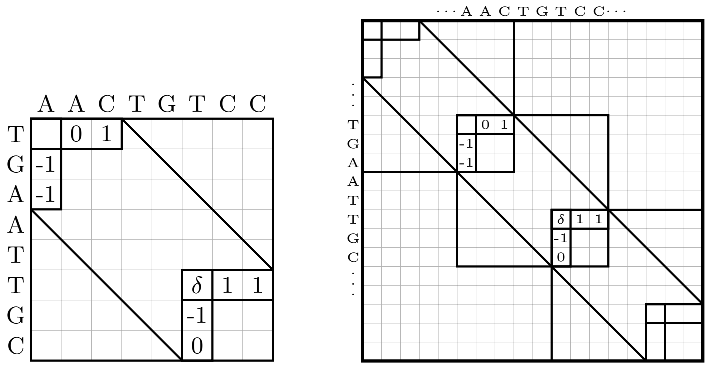


Figure 3 Example of our approach to the Four Russians speedup. **Left:** a block for maximum edit distance $d = 2$. The output δ represents the offset from the upper left corner of the current block to the upper left corner of the next block. Note that we only need to consider a diagonal band of the block itself. **Right:** using these blocks to cover the diagonal band of the dynamic programming table in the context of banded alignment.

tricks such as offset encoding of the input rows and columns as some vector in $\{-1, 0, 1\}^d$ can be applied in this case.

Another benefit of this approach is that it is more straightforward to implement in practice. Each block depends on the full output of one previous block. In contrast, the classical approach requires combining partial input from two previous blocks and also sending output to two separate blocks.

2.1.3 Theoretical bound on the running time of our approach

To give some intuition, we prove a theoretical bound on the running time under the assumption of at most b distinct substrings per interval in the dataset. This is a reasonable assumption for certain application in computational biology. For example, the 16s rRNA gene is highly conserved and thus b is much smaller than n for such datasets. While standard banded alignment takes $O(n^2md)$, we show that for small enough b this can be reduced to $O(n^2m)$. We prove this bound for our approach to using the Four Russians speedup for banded alignment, but it extends to the classical approach as well.

▶ **Theorem 1.** *If $b \leq \frac{n}{3^d\sqrt{d}}$, we can find all pairs of distance at most d in $O(n^2m)$ time.*

Proof. To simplify, we will assume the lookup table is pre-computed. Then, we must show that if $b \leq \frac{n}{3^d\sqrt{d}}$, then building the lookup table and doing the actual string comparisons can each be done in $O(n^2m)$ time. We further assume $k \approx 2d$ (in practice we choose a larger k).

First, we show that there are at most $\frac{m}{k-d}b^2 3^{2d}$ entries in the lookup table. There are at most $\frac{m}{k-d}$ intervals and since each interval has at most b distinct strings, there are at most b^2 relevant string comparisons. Each distinct string comparison must be computed for all 3^{2d} offset vector inputs. The cost of generating each lookup entry is simply the cost of computing banded alignment on a block, kd. Thus, the lookup table can be built in time $\frac{m}{k-d}b^2 3^{2d}kd$. Keeping our goal in mind we see that

$$\frac{m}{k-d}b^2 3^{2d}kd \leq n^2m \qquad \text{is true when} \qquad b \leq \frac{n}{3^d\sqrt{d}} \qquad \text{since } k \approx 2d$$

To bound the running time of the string comparisons, notice that comparing two strings requires computing $\frac{m}{k-d}$ block functions. The time spent at each block will be $O(k + d)$ to

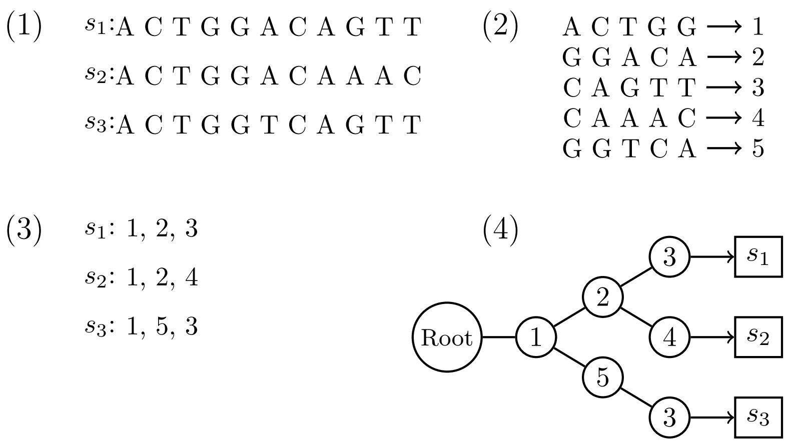


Figure 4 Example illustrating the steps of Algorithm 1 with $d = 1$ and $k = 5$.

look up the output of the block function and update our sum for the next corner. Thus, building the lookup table and computing the edit distance between all pairs using the lookup table each take $O(n^2m)$ time. ◀

2.2 The Edit Distance Interval Trie (EDIT)

To facilitate the crucial step of identifying all strings within edit distance d of a chosen cluster center, we construct a trie-like data structure on the intervals. This structure will be built during a pre-processing stage. Then, during recruitment, any recruited sequences will be deleted from the structure. The procedure for building this structure is summarized in Algorithm 1 and illustrated in Figure 4. The main benefit of this data structure, like any trie, is that it exploits prefix similarity to avoid duplicating work.

The mapping in step 2 of Algorithm 1 is a one-to-one mapping to integers from one to the number of distinct substrings. Here, it serves to reduce the size of the data structure since the number of distinct substrings will typically be much less than all possible length k strings on the given alphabet. This mapping also speeds up calls to the lookup table during the recruitment subroutine summarized in the next section.

Algorithm 1: BUILD-EDIT

```
1 Partition each sequence into overlapping intervals of length $k$, such that each interval
    overlaps on exactly $d + 1$ characters.
2 Map each distinct substring of length $k$ appearing in our list of interval strings to an
    integer.
3 Assign an integer vector signature to each sequence by replacing each block with its
    corresponding integer value.
4 Insert these signatures into a trie with the leaves being pointers to the original
    sequences.
```

2.3 Recruiting to a center

Given a center sequence s_c, we can recruit all sequences of distance at most d from s_c by simply traversing the trie in depth first search order and querying the block function of each

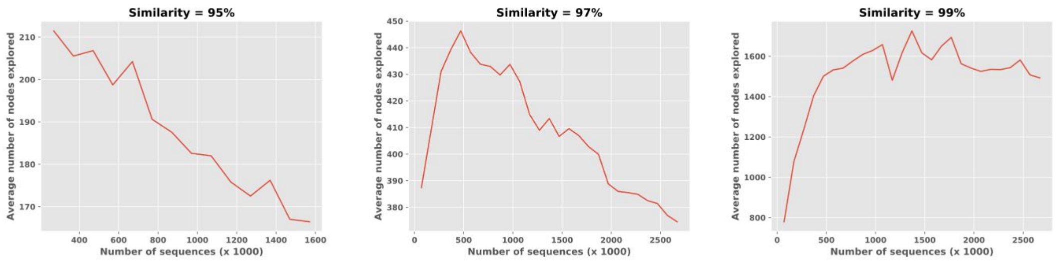


Figure 5 Plots for the average number of nodes explored in the tree while recruiting sequences to a cluster center.

node we encounter. The input to each block function is the substring of that node in the trie, the substring at the same depth within the signature of s_c, and the offset vectors output by the previous block function. As we traverse a path from the root toward a leaf, we store a sum of the edit distance as provided by the output of each block function. If this sum ever exceeds the maximum distance d, we stop exploring that path and backtrack. Whenever we reach a leaf ℓ, we retrieve its corresponding sequence s_ℓ. Then, we align the remaining suffixes and compute the true similarity threshold $d' \leq d$ based on the length of the shorter sequence. If the final edit distance is less than d', we add s_ℓ to the cluster centered at s_c and prune/remove any nodes in the the trie corresponding only to s_ℓ.

3 Experimental results

3.1 Properties of our recruitment algorithm and data structure

In this section, we highlight some key features of our recruitment algorithm and the EDIT data structure. To evaluate our method, we used a dataset consisting of about 57 million 16S rRNA amplicon sequencing reads with 2.7 million distinct sequences. To understand the impact of the number of input sequences to cluster on the average number of comparisons in each recruitment step, we ran our algorithm on different input sizes at different similarity thresholds. We counted the average number of tree nodes explored while recruiting a particular center sequence and used it as a quantitative representation of the amount of comparisons made since all nodes represent a substring of fixed length k. Figure 5 shows the plots for the average number of tree nodes explored for different similarity thresholds. For the 95% and 97% similarity thresholds, the average number nodes explored decreases as more sequences are clustered. This happens because of the fact that although more sequences are clustered, due to the lower similarity threshold a large number of sequences get clustered in each traversal of the tree. For 99% similarity threshold, the average number of nodes explored increases initially with the number of sequences, but becomes uniform after about $100,000$ sequences. The strict increase in the number of nodes can be explained by the high similarity threshold. However, in all cases, the number of nodes explored by each center does not increase linearly with the number of input sequences. Thus the total number of comparisons made for given dataset is observed to be increasing as function of n rather than the worst case n^2.

To understand the likelihood of backtracking at each level of the tree, we clustered a sample of 1.07 million distinct sequences at three different similarity thresholds (95%, 97%, and 99%). The backtracking probability for a given node was calculated as the ratio of the number of times we stopped exploring a path at that node to the total number of times

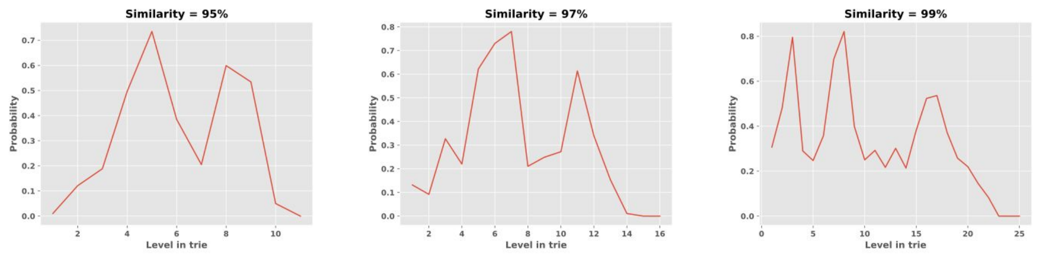


Figure 6 Plots for the probability of backtracking at a particular level in the tree. Note that the number of levels is different for different similarity thresholds since our substring length k is dependent on the maximum distance d.

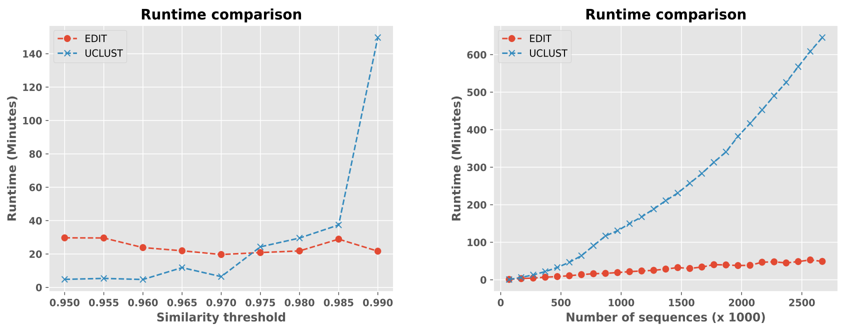


Figure 7 Running time comparison of EDIT and UCLUST as a function of similarity threshold and number of sequences.

that node was explored. We aggregated this probability for all of the nodes belonging to the same level of the tree. Figure 6 shows this likelihood for all levels of the tree at the different similarity thresholds. As we define block size based on the similarity, there are a different number of levels in the tree corresponding to different similarity thresholds. For all three similarity thresholds, the probability of backtracking decreases as we go deeper into the tree except a couple of sharp peaks at intermediate levels. These peaks can be attributed to the sequencing artifacts. The 16S rRNA gene reads are sequenced using the Illumina paired-end sequencing protocol. These paired reads are then merged to make a single read which we use for clustering. The reads contain sequencing errors concentrated near their ends. Due to such sudden errors along the sequence, while recruiting, the edit distance can easily go above the threshold and backtracking needs to be performed. Since we're using lazy computation, our first encounter with a particular input to the block function requires us to explicitly perform the dynamic programming for that block and store it in the lookup table. However, we observed that this explicit dynamic programming computation is rare and most block functions can be computed by simply querying the lookup table (data not shown).

3.2 Comparison with UCLUST

Here, we evaluate EDIT against UCLUST, a highly used tool for analyzing 16S rRNA gene datasets.

3.2.1 Running time analysis

We compared the running time of EDIT and UCLUST on a subsample 1.07 million distinct sequences at different similarity thresholds. Figure 7 shows the plot for running time at

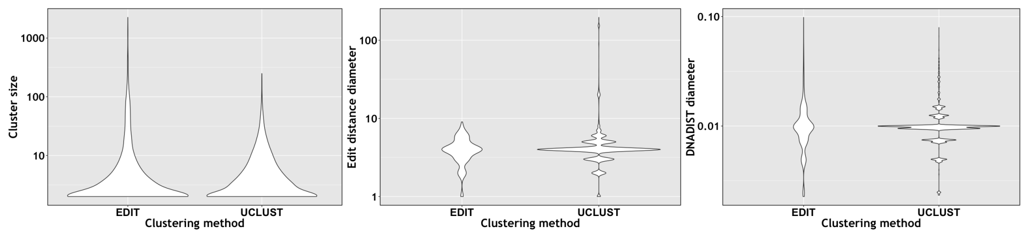


Figure 8 Evaluation at similarity threshold of 99%. All of the plots are log scaled.

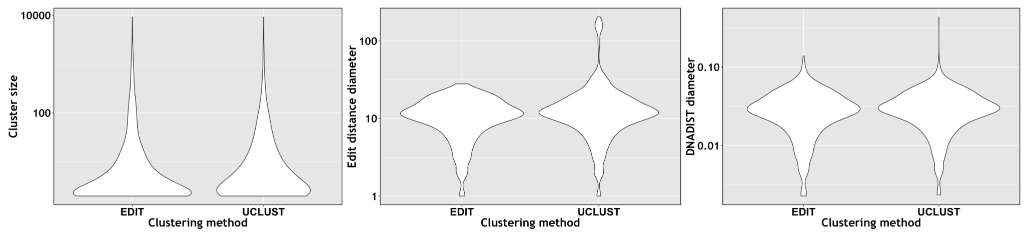


Figure 9 Evaluation at similarity threshold of 97%. All of the plots are log scaled.

different similarity thresholds. We observed that the running time of EDIT stays fairly constant at different similarity thresholds whereas the running time of UCLUST was very low for lower similarity thresholds, but increased non-linearly at higher similarity thresholds. Especially, between 98.5% to 99%, the running time of UCLUST grows 5 folds. We did further analysis of running time at 99% similarity threshold using different sample sizes as input. Figure 7 shows the running time comparison of UCLUST and EDIT. It can be observed that, the running time of UCLUST on large sample sizes (> 1 million) grows much faster that the running time of EDIT, which scales almost linearly. For the largest sample of 2.7 million sequences, UCLUST running time was ten times greater than EDIT running time. This evaluation implies that higher similarity thresholds ($> 98\%$), EDIT was faster compared to UCLUST. Also, the running time of EDIT showed low variance compared to UCLUST for different similarity thresholds.

3.2.2 Evaluation of clusters

We subsampled 135,880 distinct sequences from the entire dataset and ran both methods at the 97% and 99% similarity thresholds. We then compared the outputs of both methods using three metrics: the cluster size, the cluster diameter based on the sequence similarity, and the cluster diameter based on the evolutionary distance. To compute the cluster diameter based on sequence similarity, we computed the maximum edit distance between any two sequences in each cluster. To compute the cluster diameter based on evolutionary distance, we first performed a multiple sequence alignment of the sequences in each cluster using clustalW [15]. Once the multiple sequence alignment was computed, we used the DNADIST program from the phylip [4] package to compute a pairwise evolutionary distance matrix. The maximum distance between any pair of sequences is defined as the DNADIST diameter. Using two orthogonal notions of cluster diameter helps to define the "tightness" of clusters. Figures 8 and 9 show violin plots for different comparison metrics at the 99% and 97% similarity thresholds, respectively. At the 99% similarity threshold, EDIT is able to produce larger clusters compared to UCLUST. The edit distance diameters for the clusters generated by EDIT is fairly well constrained. However, the edit distance diameters for the clusters generated by UCLUST had a large variance, implying that several dissimilar sequences may be getting clustered together. The DNADIST diameter for both methods was comparable. At the 97% similarity threshold, both EDIT and UCLUST generated similar sized clusters. Even in this case, the edit distance diameter for UCLUST clusters showed a larger variance compared to the edit distance diameter for EDIT clusters. The DNADIST diameter for UCLUST has slightly more variance compared to that of EDIT clusters, implying some of the clusters generated by UCLUST had sequences with a large evolutionary distance between them. This validation confirms that the sequences in the clusters produced by EDIT at different similarity thresholds are highly similar to each other.

At the 99% similarity threshold, we observed a stark difference between the cluster sizes of EDIT and UCLUST. For example, the two largest clusters produced by EDIT had sizes 7,978 and 3,383 respectively whereas the two largest clusters produced by UCLUST were of sizes 249 and 233, which is almost 30 times smaller than the largest EDIT cluster. To investigate this further, we used BLAST[1] to align all clusters of UCLUST against the top two largest clusters of EDIT. We only considered the alignments with 100% alignment identity and alignment coverage. We observed that 765 distinct UCLUST clusters had all of their sequences aligned to the largest EDIT cluster and 837 distinct clusters had at least 80% of their sequences aligned to the largest EDIT cluster. Only 82 UCLUST clusters out of 16,968 total (not including singletons) had less than 80% of their sequences mapped to the largest EDIT cluster. Those 82 clusters accounted for only 255 sequences, roughly 30 times fewer than the number of the sequences in the largest EDIT cluster alone. As far as singletons (the clusters with only one sequence) are concerned, EDIT generated 22,318 singleton clusters whereas UCLUST generated 33,519 singleton clusters. For the size of the sample considered in this analysis, this difference is very significant. This evaluation implies that at a high similarity threshold, heuristic based methods like UCLUST tend to produce fragmented clusters whereas EDIT was able to capture a higher number of similar sequences in a single cluster.

4 Conclusion and future directions

The datasets analyzed by biologists are rapidly increasing in size due to the advancements in sequencing technologies and efficient clustering are needed to analyze these datasets in reasonable memory and running time. In this paper, we proposed a first step towards this goal by designing a novel data structure to perform banded sequence alignment. We extended the traditional Four Russian's method to perform banded alignment of highly similar sequences and use that to perform greedy clustering of 16S rRNA amplicon sequencing reads. We compared our method to UCLUST and showed that our method generates tight clusters at different similarity thresholds when both string similarity and evolutionary distance are considered. We focused our discussion of results around high similarity clustering ($> 97\%$) because there is no fixed threshold that can create biologically meaningful clusters. Our method can generate mathematically well-defined and tight clusters, which can serve as representative clusters from the original data and thus can be used to perform downstream computationally intensive analysis.

Although we use clustering as a motivating example throughout the paper, our algorithm could be used in a variety of different contexts where highly similar sequences need to be identified from the data. We plan to extend our algorithm to make it parallelized by performing the traversal of each tree branch in parallel. Most the the tree is explored by sequences which end up becoming singletons and this dominated the running time. We plan to explore different methods such as k-mer filters and locality sensitive hashing to flag singletons and exclude them from the recruiting process.

References

1 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

2 J. Gregory Caporaso, Christian L. Lauber, William A Walters, Donna Berg-Lyons, James Huntley, Noah Fierer, Sarah M. Owens, Jason Betley, Louise Fraser, Markus Bauer, et al. Ultra-high-throughput microbial community analysis on the Illumina HiSeq and MiSeq platforms. *The ISME journal*, 6(8):1621–1624, 2012.

3 Robert C. Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, 2010.

4 J. Felsenstein. PHYLIP-phylogeny inference package (version 3.2). *cladistics*, 5:164–166, 1989.

5 Mohammadreza Ghodsi, Bo Liu, and Mihai Pop. DNACLUST: accurate and efficient clustering of phylogenetic marker genes. *BMC bioinformatics*, 12(1):271, 2011.

6 Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

7 Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

8 Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

9 William J. Masek and Michael S. Paterson. How to compute string-edit distances quickly. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, pages 337–349. Addison-Wesley Publ. Co., Mass., 1983.

10 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980. `doi:10.1016/0022-0000(80)90002-1`.

11 Gerard Muyzer, Ellen C. De Waal, and Andre G. Uitterlinden. Profiling of complex microbial populations by denaturing gradient gel electrophoresis analysis of polymerase chain reaction-amplified genes coding for 16S rRNA. *Applied and environmental microbiology*, 59(3):695–700, 1993.

12 Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986.

13 Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)*, 46(3):395–415, 1999.

14 Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

15 Julie D. Thompson, Toby Gibson, Des G. Higgins, et al. Multiple sequence alignment using ClustalW and ClustalX. *Current protocols in bioinformatics*, pages 2–3, 2002.

16 Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

17 James R. White, Saket Navlakha, Niranjan Nagarajan, Mohammad-Reza Ghodsi, Carl Kingsford, and Mihai Pop. Alignment and clustering of phylogenetic markers-implications for microbial diversity studies. *BMC bioinformatics*, 11(1):152, 2010.

Optimal Computation of Overabundant Words

Yannis Almirantis[1], Panagiotis Charalampopoulos[2], Jia Gao[3], Costas S. Iliopoulos[4], Manal Mohamed[5], Solon P. Pissis[6], and Dimitris Polychronopoulos[7]

1 **National Center for Scientific Research Demokritos, Athens, Greece**
`yalmir@bio.demokritos.gr`
2 **Department of Informatics, King's College London, London, UK**
`panagiotis.charalampopoulos@kcl.ac.uk`
3 **Department of Informatics, King's College London, London, UK**
`jia.gao@kcl.ac.uk`
4 **Department of Informatics, King's College London, London, UK**
`costas.iliopoulos@kcl.ac.uk`
5 **Department of Informatics, King's College London, London, UK**
`manal.mohamed@kcl.ac.uk`
6 **Department of Informatics, King's College London, London, UK**
`solon.pissis@kcl.ac.uk`
7 **Computational Regulatory Genomics Group, MRC London Institute of Medical Sciences, Imperial College London, Hammersmith Hospital Campus, London, UK**
`dpolychr@imperial.ac.uk`

Abstract

The observed frequency of the longest proper prefix, the longest proper suffix, and the longest infix of a word w in a given sequence x can be used for classifying w as *avoided* or *overabundant*. The definitions used for the expectation and deviation of w in this statistical model were described and biologically justified by Brendel et al. (J Biomol Struct Dyn 1986). We have very recently introduced a time-optimal algorithm for computing all avoided words of a given sequence over an integer alphabet (Algorithms Mol Biol 2017). In this article, we extend this study by presenting an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all overabundant words in a sequence x of length n over an integer alphabet. Our main result is based on a new *non-trivial* combinatorial property of the suffix tree $\mathcal{T}$ of x: the number of distinct factors of x whose longest infix is the label of an explicit node of $\mathcal{T}$ is no more than $3n-4$. We further show that the presented algorithm is time-optimal by proving that $\mathcal{O}(n)$ is a tight upper bound for the number of overabundant words. Finally, we present experimental results, using both synthetic and real data, which justify the *effectiveness* and *efficiency* of our approach in practical terms.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases overabundant words, avoided words, suffix tree, DNA sequence analysis

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.4

1 Introduction

Brendel et al. in [6] initiated research into the linguistics of nucleotide sequences that focused on the concept of words in continuous languages – languages devoid of blanks – and introduced an operational definition of words. The authors suggested a method to measure, for each possible word w of length k, the deviation of its observed frequency $f(w)$ from the expected frequency $E(w)$ in a given sequence x. The observed frequency of the longest proper prefix, the longest proper suffix, and the longest infix of w in x were used to measure $E(w)$. The values of the deviation, denoted by $dev(w)$, were then used to identify words that are *avoided* or *overabundant* among all possible words of length k. The typical length of avoided (or of overabundant) words of the nucleotide language was found to range from 3 to 5 (tri- to pentamers). The statistical significance of the avoided words was shown to reflect their biological importance. That work, however, was based on the very limited sequence data available at the time: only DNA sequences from two viral and one bacterial genomes were considered. Also note that the range of typical word length k might change when considering eukaryotic genomes, the complex dynamics and function of which are expected to impose more demanding roles to avoided or overabundant words of nucleotides.

To this end, in [1], we presented an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all avoided words of length k in a sequence of length n over a fixed-sized alphabet. For words over an integer alphabet of size σ, the algorithm requires time $\mathcal{O}(\sigma n)$, which is optimal for sufficiently large σ. We also presented a time-optimal $\mathcal{O}(\sigma n)$-time algorithm to compute all avoided words (of any length) in a sequence of length n over an integer alphabet of size σ. We provided a tight asymptotic upper bound for the number of avoided words over an integer alphabet and the expected length of the longest one. We also proved that the same asymptotic upper bound is tight for the number of avoided words of fixed length k when the alphabet is sufficiently large. The authors in [3, 2, 4] studied a similar notion of *unusual words* – based on different definitions than the ones Brendel et al. use for expectation and deviation – focusing on the factors of a sequence; based on Brendel et al.'s definitions, we focus on any word over the alphabet. More recently, space-efficient detection of unusual words has also been considered [5]; such avoidances is becoming an interesting line of research [18].

In this article, we wish to complement our study in [1] by focusing on overabundant words. The motivation comes from molecular biology. Genome dynamics, i.e. the molecular mechanisms generating random mutations in the evolving genome, are quite complex, often presenting self-enhancing features. Thus, it is expected to often give rise to words of nucleotides which will be overabundant, i.e. being present at higher amounts than expected on the basis of their longest proper prefix, longest proper suffix, and longest infix frequencies. One specific such mechanism, which might generate overabundant words, is the following: it is well-known that in a genomic sequence of an initially random composition, the existing relatively long homonucleotide tracts present a higher frequency of further elongation than the frequency expected on the basis of single nucleotide mutations [15]; that is, they present a sort of autocatalytic self-elongation. This feature, in combination with the much higher frequency of transition *vs.* transversion mutation events, generates overabundant words which are homopurinic or homopurimidinic tracts. It is also anticipated that the overabundance of homonucleotide tracts will strongly differentiate between conserved and non-conserved parts of the genome. While this phenomenon is largely free to act within the non-conserved genomic regions, and thus it is expected to generate there large amounts of overabundant words, it is hindered in the conserved genomic regions due to selective constraints.

Our Contributions. Analogously to avoided words [6, 11, 1], many different models and algorithms exist for identifying words that are in abundance in a given sequence; see for instance [7, 9]. In this article, we make use of the biologically justified model introduced by Brendel et al. [6] and, by proving non-trivial combinatorial properties, we show that it admits *efficient* computation for overabundant words as well. We also present experimental results, using both synthetic and real data, which further highlight the *effectiveness* of this model. The computational problem can be described as follows. Given a sequence x of length

n and a real number $\rho > 0$, compute the set of ρ-overabundant words, i.e. all words w for which $dev(w) \geq \rho$. We present an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all ρ-overabundant words (of any length) in a sequence x of length n over an integer alphabet. This result is based on a combinatorial property of the suffix tree $\mathcal{T}$ of x that we prove here: the number of distinct factors of x whose longest infix is the label of an explicit node of $\mathcal{T}$ is no more than $3n - 4$. We further show that the presented algorithm is time-optimal by proving that $\mathcal{O}(n)$ is a tight upper bound for the number of ρ-overabundant words. Finally, we pose an open question of combinatorial nature on the maximum number $\mathsf{OW}(n, \sigma)$ of overabundant words that a sequence of length n over an alphabet of size $\sigma > 1$ can contain.

2 Terminology and Technical Background

2.1 Definitions and Notation

We begin with basic definitions and notation, generally following [8]. Let $x = x[0]x[1] \dots x[n-1]$ be a *word* of *length* $n = |x|$ over a finite ordered *alphabet* Σ of size σ, i.e. $\sigma = |\Sigma|$. In particular, we consider the case of an *integer alphabet*; in this case each letter is replaced by its rank such that the resulting word consists of integers in the range $\{1, \dots, n\}$. In what follows we assume without loss of generality that $\Sigma = \{0, 1, \dots, \sigma - 1\}$. We also define Σ_x to be the alphabet of word x and $\sigma_x = |\Sigma_x|$. For two positions i and j on x, we denote by $x[i \mathinner{..} j] = x[i] \dots x[j]$ the *factor* (sometimes called *subword*) of x that starts at position i and ends at position j (it is empty if $j < i$), and by ε the *empty word*, word of length 0. We recall that a *prefix* of x is a factor that starts at position 0 ($x[0 \mathinner{..} j]$) and a *suffix* is a factor that ends at position $n-1$ ($x[i \mathinner{..} n-1]$), and that a factor of x is a *proper* factor if it is not x itself. A factor of x that is neither a prefix nor a suffix of x is called an *infix* of x. We denote the *reverse* word of x by $\mathsf{rev}(x)$, i.e. $\mathsf{rev}(x) = x[n-1]x[n-2] \dots x[1]x[0]$. We say that x is *a power* of a word y if there exists a positive integer k, $k > 1$, such that x is expressed as k consecutive concatenations of y; we denote that by $x = y^k$.

Let $w = w[0]w[1] \dots w[m-1]$ be a word, $0 < m \leq n$. We say that there exists an *occurrence* of w in x, or, more simply, that w *occurs in* x, if w is a factor of x, which we denote by $w \preceq x$. Every occurrence of w can be characterised by a starting position in x. Thus we say that w occurs at position i in x when $w = x[i \mathinner{..} i + m - 1]$. Further, let $f(w)$ denote the *observed frequency*, that is, the number of occurrences of a non-empty word w in word x. If $f(w) = 0$ for some word w, then w is called *absent* (which is denoted by $w \not\preceq x$), otherwise, w is called *occurring*.

By $f(w_p)$, $f(w_s)$, and $f(w_i)$ we denote the observed frequency of the longest proper prefix w_p, suffix w_s, and infix w_i of w in x, respectively. We can now define the *expected frequency* of word w, $|w| > 2$, in x as in Brendel et al. [6]:

$$E(w) = \frac{f(w_p) \times f(w_s)}{f(w_i)}, \text{ if } \ f(w_i) > 0; \text{ else } E(w) = 0. \tag{1}$$

The above definition can be explained intuitively as follows. Suppose we are given $f(w_p)$, $f(w_s)$, and $f(w_i)$. Given an occurrence of w_i in x, the probability of it being preceded by $w[0]$ is $\frac{f(w_p)}{f(w_i)}$ as $w[0]$ precedes exactly $f(w_p)$ of the $f(w_i)$ occurrences of w_i. Similarly, this occurrence of w_i is also an occurrence of w_s with probability $\frac{f(w_s)}{f(w_i)}$. Although these two events are not always independent, the product $\frac{f(w_p)}{f(w_i)} \times \frac{f(w_s)}{f(w_i)}$ gives a good approximation of the probability that an occurrence of w_i at position j implies an occurrence of w at position $j - 1$. It can be seen then that by multiplying this product by the number of occurrences of w_i we get the above formula for the expected frequency of w.

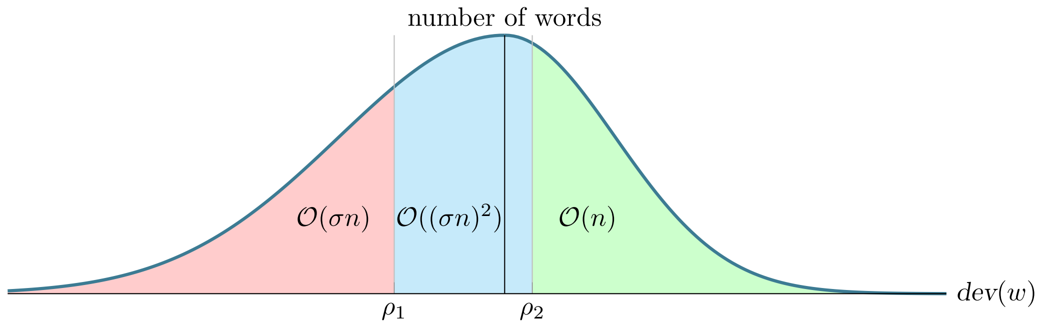


Figure 1 For a word x, the words for which $dev(w)$ is defined are the ones of the form $w = aub$, where u is a factor of x and $a, b \in \Sigma$, not necessarily distinct. There are $\mathcal{O}(n^2)$ distinct factors in a word of length n and for each of these we obtain σ^2 words of this form. We have shown that the ρ_1-avoided words are $\mathcal{O}(\sigma n)$ [1]. In this article, we show that the ρ_2-overabundant ones are $\mathcal{O}(n)$.

Moreover, to measure the deviation of the observed frequency of a word w from its expected frequency in x, we define the *deviation* (χ^2 test) of w as:

$$dev(w) = \frac{f(w) - E(w)}{\max\{\sqrt{E(w)}, 1\}}. \tag{2}$$

For more details on the *biological* justification of these definitions see [6] and [1].

Using the above definitions and two given thresholds, we can classify a word w as either *avoided*, *common*, or *overabundant* in x. In particular, for two given thresholds $\rho_1 < 0$ and $\rho_2 > 0$, a word w is called ρ_1-*avoided* if $dev(w) \leq \rho_1$, ρ_2-*overabundant* if $dev(w) \geq \rho_2$, and (ρ_1, ρ_2)-*common* otherwise (see Figure 1). We have very recently shown that the number of ρ_1-avoided words is $\mathcal{O}(\sigma n)$, and have introduced a time-optimal algorithm for computing all of them in a given sequence over an integer alphabet [1]. In this article, we show that the number of ρ_2-overabundant words is $\mathcal{O}(n)$, and study the following computational problem.

AllOverabundantWordsComputation
Input: A word x of length n and a real number $\rho > 0$
Output: All ρ-overabundant words in x

2.2 Suffix Trees

In our algorithms, suffix trees are used extensively as computational tools. For a general introduction to suffix trees see [8].

The *suffix tree* $\mathcal{T}(x)$ of a non-empty word x of length n is a compact trie representing all suffixes of x. The nodes of the trie which become nodes of the suffix tree are called *explicit* nodes, while the other nodes are called *implicit*. Each edge of the suffix tree can be viewed as an upward maximal path of implicit nodes starting with an explicit node. Moreover, each node belongs to a unique path of that kind. Then, each node of the trie can be represented in the suffix tree by the edge it belongs to and an index within the corresponding path.

We use $\mathcal{L}(v)$ to denote the *path-label* of a node v, i.e., the concatenation of the edge labels along the path from the root to v. We say that v is path-labelled $\mathcal{L}(v)$. Additionally, $\mathcal{D}(v) = |\mathcal{L}(v)|$ is used to denote the *word-depth* of node v. Node v is a *terminal* node if and only if $\mathcal{L}(v) = x[i \,.\,. n-1]$, $0 \leq i < n$; here v is also labelled with index i. It should be clear that each occurring word w in x is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(x)$. The *suffix-link* of a node v with path-label $\mathcal{L}(v) = \alpha y$ is a pointer to the node path-labelled y, where $\alpha \in \Sigma$ is a single letter and y is a word. The suffix-link of v exists if v is a non-root internal node of $\mathcal{T}(x)$.

In any standard implementation of the suffix tree, we assume that each node of the suffix tree is able to access its parent. Note that once $\mathcal{T}(x)$ is constructed, it can be traversed in a depth-first manner to compute the word-depth $\mathcal{D}(v)$ for each node v. Let u be the parent of v. Then the word-depth $\mathcal{D}(v)$ is computed by adding $\mathcal{D}(u)$ to the length of the label of edge (u, v). If v is the root then $\mathcal{D}(v) = 0$. Additionally, a depth-first traversal of $\mathcal{T}(x)$ allows us to count, for each node v, the number of terminal nodes in the subtree rooted at v, denoted by $\mathcal{C}(v)$, as follows. When internal node v is visited, $\mathcal{C}(v)$ is computed by adding up $\mathcal{C}(u)$ of all the nodes u, such that u is a child of v, and then $\mathcal{C}(v)$ is incremented by 1 if v itself is a terminal node. If a node v is a leaf then $\mathcal{C}(v) = 1$.

We assume that the terminal nodes of $\mathcal{T}(x)$ have suffix-links as well. We can either store them while building $\mathcal{T}(x)$ or just traverse it once and construct an array $node[0\,..\,n-1]$ such that $node[i] = v$ if $\mathcal{L}(v) = x[i\,..\,n-1]$. We further denote by $\textsc{Parent}(v)$ the parent of a node v in $\mathcal{T}(x)$ and by $\textsc{Child}(v, \alpha)$ the explicit node that is obtained from v by traversing the outgoing edge whose label starts with $\alpha \in \Sigma$. A batch of q $\textsc{Child}(v, \alpha)$ queries can be answered off-line in time $\mathcal{O}(n + q)$ for a word x over an integer alphabet (via radix sort).

3 Combinatorial Properties

In this section, we prove some properties that are useful for designing the time-optimal algorithm presented in the next section.

▶ **Fact 1.** *Given a word x of length n over an alphabet of size σ, the number of words w for which $dev(w)$ is defined is $\mathcal{O}((\sigma n)^2)$.*

Proof. For a word w over Σ, $dev(w)$ is only defined if $w_i \preceq x$. Hence the words w for which $dev(w)$ is defined are of the form aub for some non-empty $u \preceq x$ and $a, b \in \Sigma$. For each distinct factor $u \neq \varepsilon$ of x there are σ^2 words of the form aub, $a, b \in \Sigma$. Since there are $\mathcal{O}(n^2)$ distinct factors in a word of length n, the fact follows. ◀

▶ **Fact 2.** *Every word w that does not occur in x and for which $dev(w)$ is defined has $dev(w) \leq 0$.*

Proof. For such a word we have that $E(w) \geq 0$ and that $f(w) = 0$ and hence $dev(w) = \frac{f(w)-E(w)}{\max\{\sqrt{E(w)},1\}} \leq 0$. ◀

Naïve algorithm. By using Fact 2, we can compute $dev(w)$, for each factor w of x, thus solving Problem $\textsc{AllOverabundantWordsComputation}$. There are $\mathcal{O}(n^2)$ such factors, however, which make this computation inefficient.

▶ **Fact 3.** *Given a factor w of a word x, if w_i corresponds to an implicit node in the suffix tree $\mathcal{T}(x)$, then so does w_p.*

Proof. A factor w' of x corresponds to an implicit node $\mathcal{T}(x)$ if and only if every occurrence of it in x is followed by the same unique letter $b \in \Sigma$. Hence, since $w_p = aw_i$ for some $a \in \Sigma$, if w_i is always followed by, say, $b \in \Sigma$, every occurrence of w_p in x must also always be followed by b. Thus w_p corresponds to an implicit node as well. ◀

▶ **Lemma 4.** *If w is a factor of a word x and w_i corresponds to an implicit node in $\mathcal{T}(x)$, then $dev(w) = 0$.*

Proof. If a word $w' \preceq x$ corresponds to an implicit node along the edge (u, v) in $\mathcal{T}(x)$ and $\mathcal{L}(v) = w$ then the number of occurrences of w' in x is equal to that of w.

If w_i corresponds to an implicit node on edge (u, v) it follows immediately that $f(w_i) = f(w_s)$, as either w_s also corresponds to an implicit node in the same edge or $w_s = \mathcal{L}(v)$. In addition, from Fact 3 we have that w_p is an implicit node as well and it similarly follows that $f(w_p) = f(w)$. We thus have $E(w) = \frac{f(w_p)\times f(w_s)}{f(w_i)} = f(w)$ and hence $dev(w) = \frac{f(w)-E(w)}{\max\{\sqrt{E(w)},1\}} = 0$. ◀

Based on these properties, the aim of the algorithm in the next section is to find the factors of x whose longest infix corresponds to an explicit node and check if they are ρ-overabundant. More specifically, for each explicit node v in $\mathcal{T}(x)$, such that $\mathcal{L}(v) = y$, we aim at identifying the factors of x that have y as their longest infix (i.e. factors of the form ayb, $a, b \in \Sigma$). We will do that by identifying the factors of x that have y as their longest proper suffix (i.e. factors of the form ay, $a \in \Sigma$) and then checking for each of these the different letters that succeed it in x. Then we can check in time $\mathcal{O}(1)$ if each of these words is ρ-overabundant.

Note that the algorithm presented in Section 4 is fundamentally different and in a sense more involved than the one presented in [1] for the computation of *occurring* ρ-avoided words (note that a ρ-avoided word can be *absent*). This is due to the fact that for occurring ρ-avoided words we have the stronger property that w_p must correspond to an explicit node.

▶ **Theorem 5.** *Given a word x of length n, the number of distinct factors of x of the form ayb, where $a, b \in \Sigma$ and $y \neq \varepsilon$ is the label of an explicit node of $\mathcal{T}(x)$, is no more than $3n - 2 - 2\sigma_x$.*

Proof. Let S be the set of all explicit or implicit nodes in $\mathcal{T}(x)$ of the form yb such that y is represented by an explicit node other than the root. We have at most $2n - 2 - \sigma_x$ of them; there are at most $2n - 2$ edges in $\mathcal{T}(x)$, but σ_x of them are outgoing from the root. For such a word yb, the number of factors of x of the form ayb is equal to the degree of the node representing $\mathsf{rev}(yb)$ in $\mathcal{T}(\mathsf{rev}(x))$.

For every node in S, we obtain a distinct node in $\mathcal{T}(\mathsf{rev}(x))$. Let us suppose that k_1 of these nodes are non-root internal explicit nodes, k_2 are leaves, and the rest $2n-2-\sigma_x-k_1-k_2$ are implicit nodes. Each internal explicit node u contributes at most $deg(u)$ factors, where $deg(u)$ is the number of outgoing edges of node u, each leaf contributes 0 factors, and each implicit node contributes at most 1 factor.

Hence the number of such factors would be maximised if we obtained all the non-root internal explicit nodes and no leaves in $\mathcal{T}(\mathsf{rev}(x))$. Let $\mathcal{T}(\mathsf{rev}(x))$ have m non-root internal explicit nodes. The resulting upper bound then is $\sum_{u\in\mathcal{T}(\mathsf{rev}(x))\setminus\{root\}} deg(u) + (2n - 2 - \sigma_x - m) \leq n + m - \sigma_x + (2n - 2 - \sigma_x - m) = 3n - 2 - 2\sigma_x$.

Note that $\sum_{u\in\mathcal{T}(\mathsf{rev}(x))\setminus\{root\}} deg(u) \leq n + m - \sigma_x$ since there are at most n edges from explicit internal nodes to leaves and m edges to other internal nodes; σ_x of these are outgoing from the root. ◀

▶ **Corollary 6.** *The ρ-overabundant words in a word x of length n are at most $3n - 2 - 2\sigma_x$.*

Proof. By Fact 2, Lemma 4, and symmetry, it follows that the ρ-overabundant words in x are factors of x of the form ayb, where $a, b \in \Sigma$, such that $y \neq \varepsilon$ is represented by an explicit node in $\mathcal{T}(x)$ and $\mathsf{rev}(y)$ represented by an explicit node in $\mathcal{T}(\mathsf{rev}(x))$. Hence they are a subset of the set of words considered in Theorem 5. ◀

▶ **Lemma 7.** *The ρ-overabundant words in a word x of length n over a binary alphabet (e.g. $\Sigma = \{\mathtt{a}, \mathtt{b}\}$) are no more than $2n - 4$.*

Proof. For every internal explicit node u of $\mathcal{T}(x)$, other than the root, let $deg'(u)$ be $deg(u)+1$ if node u is terminal and $deg(u)$ otherwise. The sum of $deg'(u)$ over the internal explicit non-root nodes of $\mathcal{T}(x)$ is no more than $2n-4$ (ignoring the case when $x = \alpha^n, \alpha \in \Sigma$). We will show that, for each such node, the ρ-overabundant words with $w_i = \mathcal{L}(u)$ as their longest proper infix are at most $deg'(u)$.

- *Case I: $deg'(u) = 2$.*
  - *Subcase 1: $deg(u) = 1$.* Node u is terminal and it has an edge with label α. We can then have at most 2 ρ-overabundant words with w_i as their longest proper infix: $\mathtt{a}w_i\alpha$ and $\mathtt{b}w_i\alpha$.
  - *Subcase 2: $deg(u) = 2$.* Node u is not terminal and it has an edge with label $\mathtt{a}$ and an edge with label $\mathtt{b}$. If only one of $\mathtt{a}w_i$ and $\mathtt{b}w_i$ occurs in x we are done. If both of them occur in x we argue as follows (irrespective of whether w_i is also a prefix of x):
    If $\mathtt{a}w_i\mathtt{a}$ is ρ-overabundant, then
    $f(\mathtt{a}w_i\mathtt{a}) - f(\mathtt{a}w_i) \times f(w_i\mathtt{a})/f(w_i) \geq \rho > 0 \Rightarrow f(\mathtt{a}w_i\mathtt{a})/f(\mathtt{a}w_i) > f(w_i\mathtt{a})/f(w_i) \Leftrightarrow 1 - f(\mathtt{a}w_i\mathtt{a})/f(\mathtt{a}w_i) < 1 - f(w_i\mathtt{a})/f(w_i) \Leftrightarrow f(\mathtt{a}w_i\mathtt{b})/f(\mathtt{a}w_i) < f(w_i\mathtt{b})/f(w_i) \Leftrightarrow f(\mathtt{a}w_i\mathtt{b}) - f(\mathtt{a}w_i) \times f(w_i\mathtt{b})/f(w_i) < 0$
    and hence $\mathtt{a}w_i\mathtt{b}$ is not ρ-overabundant. (Similarly for $\mathtt{b}w_i\mathtt{a}$ and $\mathtt{b}w_i\mathtt{b}$.)
- *Case II: $deg'(u) = 3$.* Node u is terminal and it has an edge with label $\mathtt{a}$ and an edge with label $\mathtt{b}$. If only one of $\mathtt{a}w_i$ and $\mathtt{b}w_i$ occurs in x or if both of them occur in x, but w_i is not a prefix of x, we can have at most 2 ρ-overabundant words with w_i as the proper longest infix; this can be seen by looking at the node representing $\mathsf{rev}(w_i)$ in $\mathcal{T}(\mathsf{rev}(x))$, which falls in *Case I*.
  So we only have to consider the case where both $\mathtt{a}w_i$ and $\mathtt{b}w_i$ occur in x and w_i is a prefix of x. For this case, we assume without loss of generality that $\mathtt{a}w_i$ is a suffix of x.
  If $\mathtt{a}w_i\mathtt{a}$ is ρ-overabundant, then
  $f(\mathtt{a}w_i\mathtt{a}) - f(\mathtt{a}w_i) \times f(w_i\mathtt{a})/f(w_i) \geq \rho > 0 \Rightarrow f(\mathtt{a}w_i\mathtt{a})/f(\mathtt{a}w_i) > f(w_i\mathtt{a})/f(w_i) \Leftrightarrow 1 - f(\mathtt{a}w_i\mathtt{a})/f(\mathtt{a}w_i) < 1 - f(w_i\mathtt{a})/f(w_i) \Leftrightarrow (f(\mathtt{a}w_i\mathtt{b})+1)/f(\mathtt{a}w_i) < (f(w_i\mathtt{b})+1)/f(w_i) \Rightarrow f(\mathtt{a}w_i\mathtt{b})/f(\mathtt{a}w_i) < (f(w_i\mathtt{b})/f(w_i) \Leftrightarrow f(\mathtt{a}w_i\mathtt{b}) - f(\mathtt{a}w_i) \times f(w_i\mathtt{b})/f(w_i) < 0$
  and hence $\mathtt{a}w_i\mathtt{b}$ is not ρ-overabundant. Thus in this case we can have at most $3 = deg'(u)$ ρ-overabundant words.

We can thus have at most $deg'(u)$ ρ-overabundant words for each internal explicit non-root node of $\mathcal{T}(x)$. This concludes the proof. ◀

▶ **Lemma 8.** *The ρ-overabundant words in a word of length n are $\mathcal{O}(n)$ and this bound is tight. There exists a word over the binary alphabet with $2n-6$ ρ-overabundant words.*

Proof. The asymptotic bound follows directly from Corollary 6. The tightness of the asymptotic bound can be seen by considering word $x = ba^{n-2}b$, $a, b \in \Sigma$, of length n and some ρ such that $0 < \rho < 1/n$. Then for every prefix w of x of the form ba^k and for every suffix w' of x of the form a^kb, $2 \leq k \leq n-2$, we have that $f(w_p) = f(w'_s) = 1$, $f(w_s) = f(w'_p) = n-k-1$, and $f(w_i) = f(w'_i) = n-k$. Hence for any w we have $dev(w) = 1 - \frac{1\times(n-k-1)}{n-k} = \frac{1}{n-k} > \rho$. For instance, for $w = ba^{n-2}$, we have $dev(w) = 1/2$. There are $2n-6 = \Omega(n)$ such factors and hence at least these many ρ-overabundant words. ◀

▶ **Corollary 9.** *The (ρ_1, ρ_2)-common words in a word of length n over an alphabet of size σ are $\mathcal{O}((\sigma n)^2)$.*

Proof. By Fact 1 we know that $dev(w)$ is defined for $\mathcal{O}((\sigma n)^2)$ words. The ρ_1-avoided ones are $\mathcal{O}(\sigma n)$ [1], while the ρ_2-overabundant are $\mathcal{O}(n)$ by Corollary 6. Hence the (ρ_1, ρ_2)-common words are $\mathcal{O}((\sigma n)^2)$. ◀

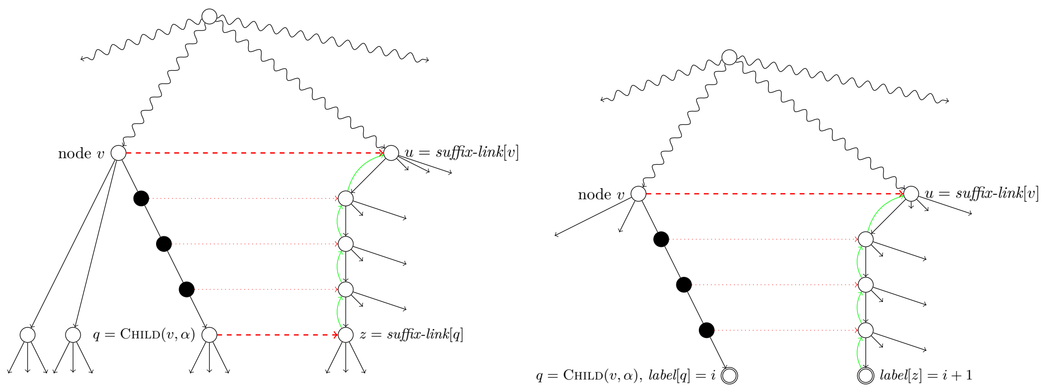

node v
$u = \textit{suffix-link}[v]$
$q = \text{CHILD}(v, \alpha)$
$z = \textit{suffix-link}[q]$
node v
$u = \textit{suffix-link}[v]$
$q = \text{CHILD}(v, \alpha)$, $\textit{label}[q] = i$
$\textit{label}[z] = i + 1$

Figure 2 The above figures illustrate the nodes (implicit or explicit) considered in a step (lines 6–37) of Algorithm 1. The figure on the left presents the case where $\text{CHILD}(v, \alpha)$ is an internal node, while the right one the case that it is a leaf. Black nodes represent implicit nodes along the edge (v, q) that we have to consider as potential w_p, and the red dotted line joins them with the respective (white) explicit node that represents the longest suffix of this w_p, i.e. w_i.

4 Algorithm

Based on Fact 2 and Lemma 4 all ρ-overabundant words of a word x are factors of x of the form ayb, where $a, b \in \Sigma$ and y is the label of an explicit node of $\mathcal{T}(x)$. It thus suffices to consider these words and check for each of them whether it is ρ-overabundant. We can find the ones that have their longest proper prefix represented by an explicit node in $\mathcal{T}(x)$ easily, by taking the suffix-link from that node during a traversal of the tree. To find the ones that have their longest proper prefix represented by an implicit node we use the following fact, which follows directly from the definition of the suffix-links of the suffix tree.

▶ **Fact 10.** *Suppose aw, where $a \in \Sigma$ and $w \in \Sigma^*$, is a factor of a word x and that w is represented by an explicit node v in $\mathcal{T}(x)$, while aw by an implicit node along the edge (u_1, u_2) in $\mathcal{T}(x)$. The suffix-link from u_2 points to a node in the subtree of $\mathcal{T}(x)$ rooted at v.*

The algorithm first builds the suffix tree of word x, which can be done in time and space $\mathcal{O}(n)$ for words over an integer alphabet [10]. It is also easy to compute $\mathcal{D}(v)$ and $\mathcal{C}(v)$, for each node v of $\mathcal{T}(x)$, within the same time complexity (lines 2–5 in Algorithm 1).

The algorithm then performs a traversal of $\mathcal{T}(x)$. When it first reaches a node v, it considers $\mathcal{L}(v)$ as a potential longest proper prefix of ρ-overabundant words – i.e. $\mathcal{L}(v) = w_p = aw_i$, where $a \in \Sigma$. By following the suffix-link to node u, which represents the respective w_i, and based on the first letter of the label of each outgoing edge (v, q) from v, it computes the deviation for all possible factors of x of the form $w_p b$, where $b \in \Sigma$. (Note that we can answer all the $\text{CHILD}(u, \alpha)$ queries off-line in time $\mathcal{O}(n)$ in total for integer alphabets.) It is clear that this procedure can be implemented in time $\mathcal{O}(n)$ in total (lines 7–19).

Then, while on node v and based on Fact 10, the algorithm considers for every outgoing edge (v, q), the implicit nodes along this edge that correspond to words (potential w_p's) whose proper longest suffix (the respective w_i) is represented by an explicit node in $\mathcal{T}(x)$.

Hence, when $\mathcal{D}(q) - \mathcal{D}(v) > 1$ the algorithm follows the suffix-link from node q to node z. It then checks whether $\text{PARENT}(z) = u$. If not, then the word $\mathcal{L}(q)[0 \,..\, \mathcal{D}(\text{PARENT}(z))]$ is represented by an implicit node along the edge (v, q) and hence $\mathcal{L}(q)[0 \,..\, \mathcal{D}(\text{PARENT}(z)) + 1]$ has to be checked as a potential ρ-overabundant word. After the check is completed, the algorithm sets $z = \text{PARENT}(z)$ and iterates until $\text{PARENT}(z) = u$. This is illustrated in

Algorithm 1 Compute all ρ-overabundant words

```
 1: procedure ComputeOverabundantWords(word $x$, real number $\rho$)
 2:     $\mathcal{T}(x) \leftarrow$ BuildSuffixTree($x$)
 3:     for each node $v \in \mathcal{T}(x)$ do
 4:         $\mathcal{D}(v) \leftarrow$ word-depth of $v$
 5:         $\mathcal{C}(v) \leftarrow$ number of terminal nodes in the subtree rooted at $v$
 6:     for each node $v \in \mathcal{T}(x)$ do                              ▷ prefix node
 7:         ▷ Report $\rho$-overabundant words $w$ such that $w_p$ is explicit
 8:         $u \leftarrow$ suffix-link[$v$]                                    ▷ infix node
 9:         if $\mathcal{D}(v) > 1$ and IsInternal($v$) then
10:             $f_p \leftarrow \mathcal{C}(v)$, $f_i \leftarrow \mathcal{C}(u)$
11:             if $f_i > f_p$ and $u \neq$ Root($\mathcal{T}(x)$) then
12:                 for each child $y$ of node $v$ do
13:                     if not(IsTerminal($y$) and $\mathcal{D}(y) = \mathcal{D}(v) + 1$) then
14:                         $f_w \leftarrow \mathcal{C}(y)$
15:                         $\alpha \leftarrow \mathcal{L}(y)[\mathcal{D}(v) + 1]$
16:                         $f_s \leftarrow \mathcal{C}$(Child($u, \alpha$))
17:                         $E \leftarrow f_p \times f_s / f_i$
18:                         if $(f_w - E)/(\max\{1, \sqrt{E}\}) \geq \rho$ then
19:                             Report($\mathcal{L}(y)[0 \mathinner{.\,.} \mathcal{D}(v)]$)
20:         ▷ Report $\rho$-overabundant words $w$ such that $w_p$ is implicit
21:         for each child $y$ of node $v$ do
22:             if $\mathcal{D}(y) > \mathcal{D}(v) + 1$ then
23:                 if IsInternal($y$) then
24:                     $z \leftarrow$ suffix-link[$y$]
25:                 else                                               ▷ $y$ is a terminal node
26:                     $i \leftarrow$ label[$y$]
27:                     $z \leftarrow$ node[$i + 1$]
28:                     if $\mathcal{D}(z) = \mathcal{D}$(Parent($z$)) $+ 1$ then
29:                         $z \leftarrow$ Parent($z$)
30:                 $f_w \leftarrow f_p \leftarrow \mathcal{C}(y)$
31:                 while Parent($z$) $\neq u$ do
32:                     $f_i \leftarrow \mathcal{C}$(Parent($z$))
33:                     $f_s \leftarrow \mathcal{C}(z)$
34:                     $E \leftarrow f_p \times f_s / f_i$
35:                     if $(f_w - E)/(\max\{1, \sqrt{E}\}) \geq \rho$ then
36:                         Report($\mathcal{L}(y)[0 \mathinner{.\,.} \mathcal{D}$(Parent($z$)) $+ 1]$)
37:                     $z \leftarrow$ Parent($z$)
```

Figure 2. By Theorem 5, the Parent(z) $= u$ check will fail $\mathcal{O}(n)$ times in total. All other operations take time $\mathcal{O}(1)$ and hence this procedure takes time $\mathcal{O}(n)$ in total (lines 20–37).

We formalise this procedure in Algorithm 1, where we assume that the suffix tree of $x\$$ is built, where \$ is a special letter, $\$ \notin \Sigma$. This forces all terminal nodes in $\mathcal{T}(x)$ to be leaf nodes. We thus obtain the following result; optimality follows directly from Lemma 8.

▶ **Theorem 11.** *Algorithm 1 solves problem* AllOverabundantWordsComputation *in time and space $\mathcal{O}(n)$, and this is time-optimal.*

5 Experimental Results: Effectiveness, Efficiency, and Applications

Algorithm 1 was implemented as a program to compute the ρ-overabundant words in one or more input sequences. The program was implemented in the `C++` programming

Table 1 The deviation of the randomly generated inserted word w, as well as the word $w_{\max}$ with the maximum deviation. The length of each of the 25 randomly generated sequences over $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ was $n = 80,000$, the length of w was $m = 6$, and $\rho = 0.000001$. In green are the cases when the word with the maximum deviation was w itself or one of its factors.

Times t of inserting w	20	40	80	160	320
w	TTACAA	GTGCCC	CACTTT	AGTTAC	AAACAG
$dev(w)$	2.233313	4.143015	5.623615	6.010327	5.674220
$w_{\max}$	CTCCTATG	GTGCCC	CACTTT	AGTTA	ACAG
$dev(w_{\max})$	3.354102	4.143015	5.623615	6.900740	9.617803
w	AATCTG	AGTCGA	GAAGTC	TATCTT	CAAAAA
$dev(w)$	2.034233	2.888529	4.456468	5.073860	11.071170
$w_{\max}$	ATTGGGG	TCTGTATG	GAAGTC	ATCTT	CAAAAA
$dev(w_{\max})$	3.265609	3.272727	4.456468	6.115612	11.071170
w	GTACCA	GGCGTG	AAGGAT	GGGTCC	TTCCGG
$dev(w)$	2.187170	3.658060	4.428189	5.467296	5.256409
$w_{\max}$	TCTGTGCG	ACGATACC	AAGGAT	GGTCC	TTCCG
$dev(w_{\max})$	3.548977	4.000000	4.428189	6.787771	9.105009
w	CCATAG	GTTGAT	TGAGCG	ACATTT	CTTGTA
$dev(w)$	2.470681	2.467858	4.214544	5.755475	5.362435
$w_{\max}$	CAGTGGTC	TTTTCCT	TGAGC	ACATT	TTGTA
$dev(w_{\max})$	3.333333	3.368226	5.072968	6.376277	9.467110
w	TCGACA	CGCTTT	TACAAC	TATTAG	TGAGAT
$dev(w)$	1.531083	2.789220	3.552902	4.959926	5.124976
$w_{\max}$	CTTTGCT	ATTACC	ACAAC	ATTAG	GACAT
$dev(w_{\max})$	3.308195	3.322163	5.653479	6.837628	10.012316

language and developed under GNU/Linux operating system. Our program makes use of the implementation of the *compressed suffix tree* available in the Succinct Data Structure Library [12]. The input parameters are a (Multi)FASTA file with the input sequence(s) and a real number $\rho > 0$. The output is a file with the set of ρ-overabundant words per input sequence. The implementation is distributed under the GNU General Public License, and it is available at `http://github.com/solonas13/aw`. The experiments were conducted on a Desktop PC using one core of Intel Core i5-4690 CPU at 3.50GHz under GNU/Linux. The program was compiled with `g++` version 4.8.4 at optimisation level 3 (-O3). We also implemented a brute-force approach to confirm the correctness of our implementation. Here we do not plot the results of the brute-force approach as it is easily understood that it is orders of magnitude slower than our linear-time approach.

Experiment I (Effectiveness). In the first experiment, our task was to establish the effectiveness of the statistical model in identifying overabundant words. To this end, we generated 25 random sequences of length $n = 80,000$ over the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (uniform distribution). Then for each of these sequences, we inserted a random word w of length $m = 6$ in t random positions. We varied the value of t based on the fact that in a random sequence of length n over an alphabet of size $\sigma = |\Sigma|$, where letters are independent, identically uniformly distributed random variables, a specific word of length m is expected to occur roughly $r = n/\sigma^m$ times. We hence considered t equal to r, $2r$, $4r$, $8r$, and $16r$. We then ran our program for each resulting sequence to identify the ρ-overabundant words with

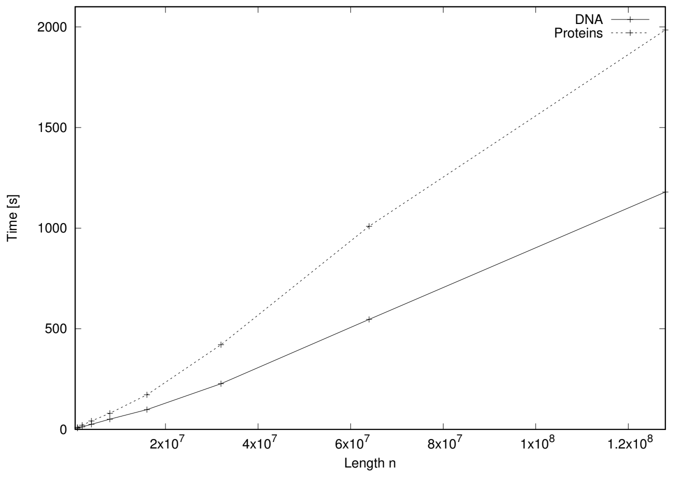


Figure 3 Elapsed time of Algorithm 1 using synthetic DNA ($\sigma = 4$) and proteins ($\sigma = 20$) sequences of length 1M to 128M.

$\rho = 0.000001$, and output the deviation of the inserted word w, as well as the word $w_{\max}$ with the maximum deviation. The inserted word w was reported as a ρ-overabundant word in *all* cases. Furthermore, in many cases the word with the maximum deviation was w itself and in many other cases one of its factors; this was true in *all* cases for $t \geq 80 \approx 4r$. Hence, the model is effective in identifying words that are overabundant. The full results of this experiment are presented in Table 1.

Experiment II (Efficiency). Our task here was to establish the fact that the elapsed time of the implementation grows linearly with n, the length of the input sequence. As input datasets, for this experiment, we used synthetic DNA ($\sigma = 4$) and proteins ($\sigma = 20$) sequences ranging from 1 to 128 M (Million letters). For each sequence we used a constant value of $\rho = 10$. The results are plotted in Fig. 3. It becomes evident from the results that the elapsed time of the program grows linearly with n. The longer time required for the proteins sequences compared to the DNA sequences for increasing n is explained by the dependence of the time required to answer queries of the form $\textsc{Child}(v, \alpha)$ on the size of the alphabet ($\sigma = 20$ *vs.* $\sigma = 4$) in the implementation of the compressed suffix tree we used.

Experiment III (Real Application). Here we proceed to the examination of seven collections of Conserved Non-coding Elements (CNEs) obtained through multiple sequence alignment between the human and other genomes. Despite being located at the non-coding part of genomes, CNEs can be extremely conserved on the sequence level across organisms. Their genesis, functions and evolutionary dynamics still remain enigmatic [16, 13]. The detailed description of how those CNEs were identified can be found in [17]. For each CNE of these datasets, a sequence stretch (surrogate sequence) of non-coding DNA of equal length and equal `GC` content was taken at random from the repeat-masked human genome. The CNEs of each collection were concatenated into a single long sequence and the same procedure was followed for the corresponding surrogates. We have determined through the proposed algorithm the overabundant words for $k = 10$ (decamers) and $\rho = 3$ for these fourteen datasets and the results are presented in Table 2. Likewise, in Table 3, we show all overabundant words (i.e. $k > 2$) for $\rho = 3$.

Table 2 Number of overabundant words for $k = 10$ and $\rho = 3$.

$k = 10$, $\rho = 3$	CNEs 75–80	CNEs 80–85	CNEs 85–90	CNEs 90–95	CNEs 95–100	Mammalian	Amniotic
Surr	1,144	718	473	297	469	15,470	2,874
CNEs	331	181	100	59	71	491	149
Ratio	**3.46**	**3.97**	**4.73**	**5.03**	**6.61**	**31.51**	**19.29**

Table 3 Number of overabundant words for $k > 2$ and $\rho = 3$.

$k > 2$, $\rho = 3$	CNEs 75–80	CNEs 80–85	CNEs 85–90	CNEs 90–95	CNEs 95–100	Mammalian	Amniotic
Surr	5,925	3,798	2,770	1,948	2,405	69,022	12,913
CNEs	1,373	778	512	390	403	7,549	1,401
Ratio	**4.32**	**4.88**	**5.41**	**4.99**	**5.97**	**9.14**	**9.22**

The first five CNE collections have been composed through multiple sequence alignment of the same set of genomes (human vs. chicken; mapped on the human genome) and they differ only in the thresholds of sequence similarity applied between the considered genomes: from 75% to 80% (the least conserved CNEs, which thus are expected to serve less demanding functional roles) to 95–100% which represent the extremely conserved non-coding elements (UCNEs or CNEs 95–100) [17]. The remaining two collections have been composed under different constraints and have been derived after alignment of Mammalian and Amniotic genomes. In Tables 2 and 3, the last line shows the ratios formed by the numbers of overabundant words of each concatenate of surrogates divided by the numbers of overabundant words of the corresponding CNE dataset.

Inspecting data contained in Tables 2 and 3, first we observe in all cases that absolute numbers of overabundant words drop from low- to high-conserved CNE concatenates. This feature is shared by the corresponding concatenates of surrogate sequences as evidenced along table rows from CNEs 75–80 to CNEs 95–100. This is due to the considerable decrease in absolute numbers of the corresponding elements in the human genome, which is reflected to the length of their concatenates. Note that in genomic sequences, extreme conservation is always clearly less frequent than medium conservation. As the studied sequences decrease in length, the numbers of overabundant words also drop in each category (CNEs or surrogates). Consequently, the important quantity is the ratio of these numbers between CNE and surrogate dataset. As amniotic and mammalian CNEs are classes characterized by different conservation thresholds (the former being much more conserved), they also present disparate overabundant word numbers, again the corresponding ratios being the relevant quantities.

Two results directly related to our analysis stem from inspection of Tables 2 and 3:

1. In all cases, the number of overabundant words from the surrogate concatenate of sequences *far exceeds* the corresponding number derived from the CNE dataset.
2. In the case of datasets with increasing degree of similarity between aligned genomes (from 75–80 to 95–100), the ratios of the numbers of overabundant words show a clear, *increasing trend.*

Both these findings can be understood on the basis of the difference in functionality between CNE and surrogate datasets. As we briefly describe in Section 1, this systematic difference (finding 1 above) is expected on the basis of the self-enhancing elongation of relatively long homonucleotide tracts [14, 15], which occurs mainly in the non-constrained parts of the genome, here the surrogate datasets. Therefore, we expect and we do find that CNE datasets always have less overabundant words than their corresponding surrogate. Moreover, finding 2 corroborates the proposed mechanism of overabundance, as in CNE datasets 1–5 depletion in overabundant words quantitatively follows the degree of sequence conservation. Inspection of the individual overabundant words found in the surrogate datasets verifies that they largely consist of short repeats of the types described in [14] and in [15]. There is an analogy of this finding with a corresponding one, concerning the occurrence of avoided words in the same sequence sets, which is described in [1].

6 Open Question

By Corollary 6 and Lemma 8, we have the following bounds on the maximum number $\mathsf{OW}(n, \sigma)$ of overabundant words in a sequence of length n over an alphabet of size $\sigma > 1$:

$$2n - 6 \leq \mathsf{OW}(n, \sigma) \leq 3n - 2 - 2\sigma.$$

We have conducted computational experiments, and for $\sigma > 2$ we obtained sequences with more than $2n$ overabundant words. An open problem is thus to find $\mathsf{OW}(n, \sigma)$.

References

1 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.

2 Alberto Apostolico, Mary Ellen Bock, and Stefano Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10(3-4):283–311, 2003.

3 Alberto Apostolico, Mary Ellen Bock, Stefano Lonardi, and Xuyan Xu. Efficient detection of unusual words. *Journal of Computational Biology*, 7(1-2):71–94, 2000.

4 Alberto Apostolico, Fang-Cheng Gong, and Stefano Lonardi. Verbumculus and the discovery of unusual words. *Journal of Computer Science and Technology*, 19(1):22–41, 2004.

5 Djamal Belazzougui and Fabio Cunial. Space-efficient detection of unusual words. In *SPIRE*, volume 9309 of *LNCS*, pages 222–233. Springer, 2015.

6 Volker Brendel, Jacques S Beckmann, and Edward N Trifonov. Linguistics of nucleotide sequences: morphology and comparison of vocabularies. *Journal of Biomolecular Structure and Dynamics*, 4(1):11–21, 1986.

7 Chris Burge, Allan M. Campbello, and Samuel Karlin. Over- and under-representation of short oligonucleotides in DNA sequences. *Proc Natl Acad Sci USA*, 89(4):1358–1362, 1992.

8 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. 2007.

9 Alain Denise, Mireille Régnier, and Mathias Vandenbogaert. Assessing the statistical significance of overrepresented oligonucleotides. In *WABI*, volume 2149 of *LNCS*, pages 85–97. Springer Berlin Heidelberg, 2001.

10 Martin Farach. Optimal suffix tree construction with large alphabets. In *FOCS*, pages 137–143. IEEE, 1997.

11 Mikhail S. Gelfand and Eugene V. Koonin. Avoidance of palindromic words in bacterial and archaeal genomes: a close connection with restriction enzymes. *Nucleic Acids Research*, 25(12):2430–2439, 1997.

12 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *SEA*, volume 8504 of *LNCS*, pages 326–337. Springer, 2014.

13 Nathan Harmston, Anja Barešić, and Boris Lenhard. The mystery of extreme non-coding conservation. *Phil. Trans. R. Soc. B*, 368(1632):20130021, 2013.

14 Suzanne E. Hile and Kristin A. Eckert. Positive correlation between DNA polymerase α-primase pausing and mutagenesis within polypyrimidine/polypurine microsatellite sequences. *Journal of Molecular Biology*, 335(3):745–759, 2004.

15 G. Levinson and G. A. Gutman. Slipped-strand mispairing: a major mechanism for DNA sequence evolution. *Molecular Biology and Evolution*, 4(3):203–221, 1987.

16 Dimitris Polychronopoulos, Diamantis Sellis, and Yannis Almirantis. Conserved noncoding elements follow power-law-like distributions in several genomes as a result of genome dynamics. *PloS One*, 9(5):e95437, 2014.

17 Dimitris Polychronopoulos, Emanuel Weitschek, Slavica Dimitrieva, Philipp Bucher, Giovanni Felici, and Yannis Almirantis. Classification of selectively constrained DNA elements using feature vectors and rule-based classifiers. *Genomics*, 104(2):79–86, 2014.

18 Ivan Rusinov, Anna Ershova, Anna Karyagina, Sergey Spirin, and Andrei Alexeevski. Lifespan of restriction-modification systems critically affects avoidance of their recognition sites in host genomes. *BMC Genomics*, 16(1):1, 2015.

Detecting Locus Acquisition Events in Gene Trees*

Michał Aleksander Ciach[1], **Anna Muszewska**[2], **and Paweł Górecki**[3]

1 **Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Warsaw, Poland; and**
Institute of Biochemistry and Biophysics, Polish Academy of Sciences, Warsaw, Poland
`m_ciach@student.uw.edu.pl`

2 **Institute of Biochemistry and Biophysics, Polish Academy of Sciences, Warsaw, Poland**

3 **Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Warsaw, Poland**
`gorecki@mimuw.edu.pl`

Abstract

Horizontal Gene Transfer (HGT), a process of acquisition and fixation of foreign genetic material, is an important biological phenomenon. Several approaches to HGT inference have been proposed. However, most of them either rely on approximate, non-phylogenetic methods or on the tree reconciliation, which is computationally intensive and sensitive to parameter values. In this work, we investigate the Locus Tree Inference problem as a possible alternative that combines the advantages of both approaches. We show several algorithms to solve the problem in the parsimony framework. We introduce a novel tree mapping, which allows us to obtain a heuristic solution to the problems of locus tree inference and duplication classification. Our approach allows not only for faster comparisons of gene and species trees but also to improve known algorithms for duplication inference in the presence of polytomies in the species trees.

1998 ACM Subject Classification J.3 [Life and Medical Sciences] Genomics

Keywords and phrases rank, taxon, ranked species tree, speciation, gene duplication, gene loss, horizontal gene transfer

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.5

1 Introduction

Horizontal Gene Transfer (HGT) is the process of acquisition and fixation of foreign genetic material. It can lead to substantial changes in the ecology and evolution of recipient organism, sometimes leading to the emergence of new pathogens [8]. HGT is interesting both from biological and computational perspective. Several methods of detecting horizontally transferred genes have been proposed, which can be roughly divided into two categories [14]. So-called *surrogate methods* are computationally efficient, yet often imprecise. The other group are the the *phylogenetic methods*, most notably the tree reconciliation [5].

HGT and gene duplication are examples of evolutionary events in which an organism gains a new locus, i.e. a fragment of a chromosome with a specific gene. The new locus evolves more or less independently of other loci. This observation leads to the concept of a *locus tree* [13],

* The support was provided by NCN grants #2015/19/B/ST6/00726 and 2012/07/D/NZ2/04286.

which represents the evolutionary history of the loci. A corresponding gene tree represents evolutionary histories of alleles found in those loci, including populational effects like the incomplete lineage sorting. Therefore, a locus tree is an "intermediate" concept between the gene and the species tree. When population effects are negligible, a locus tree is equivalent to a gene tree with some branches labelled as locus gain events. Such labelling allows to "decompose" the gene tree into a set of independent evolutionary histories of different loci. The concept of gene tree decomposition has been investigated earlier in the context of tree comparison [10]. Distinguishing between different locus gain events is challenging, as their effects on gene trees are topologically similar. In reconciliation, weights of events have to be specified; these are, however, rarely known. The fact that the results depend strongly on those unknown parameters may undermine the credibility of biological conclusions. To properly estimate the weights, high-quality training datasets are needed, in which inferred events are biologically supported.

Many cases of HGT were found by manual inspection of incongruences in gene trees [15]. Inferring a locus tree facilitates such analyses, as it allows to automatically detect the incongruences. This approach has several advantages over reconciliation. It allows to restrict to only two parameters: the locus gain and the locus loss weight. It is also more robust to imprecise data, as improperly placed branches will only be locally detected as new loci, without interfering with the global evolutionary scenario. This allows to disregard the noise when analyzing the tree, and instead focus on several important events. The locus tree inference has been addressed in populational genetics setting [13]. However, this approach requires several difficult to obtain parameters, like speciation times or population sizes.

Our contribution: In this work, we address the problem of Locus Tree Inference when populational effects are negligible. This allows addressing the locus tree inference problem in a parsimony framework. We propose to solve it by decomposing a binary gene tree into a forest of subtrees that can be embedded into a possibly polytomic species tree, in a way that minimizes the weighted sum of the forest size and the number of loss events. We propose two variants of the problem: the *Locus Tree Inference*, *LTI*, in which forest elements are subtrees of the species tree, and the *Conditional Locus Tree Inference*, *CLTI*, where each forest element is a subtree of some binarization of the species tree. We show a dynamic programming algorithm that solves LTI in $O(|G||S|m)$ time and $O(|G||S|)$ space, where m is the maximal degree of a node from the species tree. To solve CLTI, we propose a new mapping, called the Highest Separating Rank. Based on the mapping, we show an $O(d|G| + |S|)$ time and $O(|G| + |S|)$ space algorithm, where d is the height of S, for inferring required and conditional duplications in gene trees, which improves $O(|G|(d+m) + |S|)$ time solution from [18]. Finally, we propose an efficient heuristic to solve CLTI, and present a comparative study on simulated and empirical data.

2 Definitions

Let $T = \langle V_T, E_T \rangle$ be a rooted directed tree. For $a, b \in V_T$, by $\text{lca}_T(a, b)$ we denote the lowest common ancestor of a and b in T. We also use the binary order relation $a \preceq b$ if b is a node on the path between a and the root of T (note that $a \preceq a$). Two nodes a and b are called *siblings* if they are children of $\text{lca}_T(a, b)$. We call a and b *comparable* if $a \preceq b$ or $b \preceq a$, otherwise a and b are called *incomparable*. The parent of a node a is denoted as $\mathsf{parent}(a)$. The subtree of T rooted at v is denoted by $T(v)$. By $L(T)$ we denote the set of all leaves in a tree T and we use $L(v)$ instead of $L(T(v))$. By $\text{root}(T)$ we denote the root of tree T. A *species tree* S is a rooted directed tree in which nodes are called *taxa*. A *gene tree* G is a rooted directed binary tree, such that every leaf of G is labeled by a leaf-taxon from S, i.e., an element of $L(S)$. For a node g in G, by $\mathrm{tax}(g) \subset L(S)$ we denote the set of all labels of leaves from $L(g)$.

The *lowest common ancestor mapping*, or lca-mapping, between G and S is a function $M\colon V_G \to V_S$ such that $M(g) = t$ if g is a leaf labelled by the leaf-taxon t, or $M(g) = \mathrm{lca}_S(M(g_1), M(g_2))$ if g is has two children g_1 and g_2. An internal node g in G is a *duplication* if $M(g) = M(g_i)$ for any child g_i of g. Every other node, i.e. a leaf or an internal node satisfying $M(g) \succ M(g_i)$ for every child g_i of g, is called a *speciation* [12, 7, 4].

A node with more than two children is called a *polytomy*. For a polytomy s in a tree S, let $H(s)$ be the set of all possible binary trees whose leaves are the children of s. For instance, if s is the polytomy node present in S from Fig. 2, then $H(s) = \{(d,(e,f)),(e,(d,f)),(f,(d,e))\}$. Let $H^*(S)$ be the set of all possible binary trees obtained from S by replacing each polytomy s with a tree from $H(s)$. An element of $H^*(S)$ is called a *binarization* of S.

3 Locus gain Problems

In this section we introduce the parsimony framework for the (Conditional) Locus Tree Inference problem and a dynamic programming formula for solving the problem. We say that a gene tree G is *embeddable* (respectively, *conditionaly embeddable*) into a species tree S, if each node of G is a speciation (respectively, a speciation in some binarization of S). For instance, $(a,(b,c))$ is embeddable into $(a,(b,c,e),f)$, while $(a,(a,b))$ is not. Since the polytomies in S can be resolved independently, we get the following result:

▶ **Lemma 1.** *G is conditionally embeddable into S if and only if there is a binarization S' of S such that G is embeddable into S'.*

Every internal node g of G induces a set of *loss* events defined as nodes of the species tree strictly between $M(g)$ and $M(\mathsf{parent}(g))$, plus $M(g)$ if $M(g)$ is a polytomy. The above definition yields a notion of the *loss cost*, denoted by $\Lambda(G,S)$, and defined as the total number of loss events required to embed G into S.

A gene tree may not be embeddable into a species tree due to duplications or HGTs. Our goal is to decompose a gene tree into a set of embeddable subtrees in the most parsimonious way. We say that a forest F is a *decomposition* of G, if $\bigcup_{T\in F} L(T) = L(G)$, and for every $T \in F$, $G|_{L(T)} = T$, where, for $A \subseteq L(G)$, $G|_A$ is a tree induced by A and the set of internal nodes $\{\mathrm{lca}_G(a,b) | a,b \in A\}$ and inheriting the ancestor relation from G. Decompositions can be equivalenty obtained by tree edit operations as follows. Given a gene tree G, $X \subseteq E_G$ is called a *set of cuts* if no two edges in X share their top nodes.

▶ **Lemma 2.** *Let X be a set of cuts from G. Let G^X be a graph obtained from G by: removing all cuts from G, contracting all nodes with one parent and one child, and then removing all roots having exactly one child. Then G^X is a decomposition of G.*

In the context of X every node of G can be uniquely associated with a node from G^X by a mapping σ^X that maps a node g to the first non-removed node below g. Formally, $\sigma^X(g) = \sigma^X(g_1)$ if $\langle g, g_2\rangle \in X$ and g_1 is the sibling of g_2, and $\sigma^X(g) = g$, otherwise. By $M^X : G \to S$ we denote the "locus-aware" lca-mapping, given by $M^X(g) = M'(\sigma^X(g))$, where M' is the lca-mapping between $T \in G^X$ and S such that $\sigma^X(g) \in T$ (see Fig. 1).

Consider a set of cuts X in G. We say that X *detaches* $g \in G$, or is *g-detaching*, if $\sigma^X(g)$ is the root of some tree from G^X. For example, in Fig. 1, the cuts from the right example

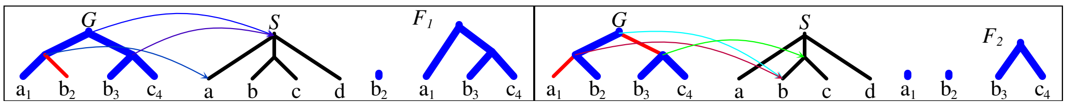


Figure 1 An example of locus trees for $G = ((a_1, b_2), (b_3, c_4))$ with two decompositions F_1 and F_2 consistent with $S = (a, (b, c), d)$. These decompositions are created by cuts indicated with red color. $M^X : G \to S$ is depicted for every set of cuts X (only for internal nodes). Here, $\Lambda(F_1, S) = \Lambda(F_2, S) = 0$, $\Delta(F_1) = 2 \cdot \mathbf{GAIN}$ and $\Delta(F_2) = 3 \cdot \mathbf{GAIN}$, i.e., F_1 is optimal.

detach the parent of leaf a (as $\sigma^X(\mathsf{parent}(a)) = b$ is a root in G^X), while the cuts from the left one do not ($\sigma^X(g) = a$ is below the root).

We say that a decompositon is *consistent* (respectively, *conditionally consistent*) with a species tree S if for every $T \in F$, T is (respectively, conditionally) embeddable into S. From the definition of σ^X we have:

▶ Remark. Let G^X be a decomposition consistent with S. Then, a set of cuts X detaches $g \in G$ if and only if every tree in G^X is either disjoint with or entirely contained in $G(g)$.

Given a species tree S and a gene tree G we define a *locus tree* with respect to S as a pair (G, X), where X is a set of cuts such that the decompositon G^X is consistent with S. Locus trees which induce the same decompositions are considered equivalent. From Lemma 2 it follows that for each set of cuts there is a unique decomposition induced by this set. Conversely, for every decomposition F of G there exists a set of cuts X such that $F = G^X$. Inferring such a set from a given decomposition is straighforward by a bottom-up traversal of the gene tree. Therefore, we can consider decompositions as equivalent to locus trees. From computational point of view, it is more natural to seek for optimal decompositions rather than sets of cuts.

Given a decompositon F, we define the *total loss cost* as $\Lambda(F, S) = \sum_{T \in F} \Lambda(T, S)$. We can now define the *Locus Tree Inference* problem (*LTI*) in the parsimony framework:

▶ **Problem** (Locus Tree Inference, LTI). *Given a gene tree G and a species tree S. Find the decomposition F^* of G consistent with S having the minimal weighted cost $\Delta(G, S) = \mathbf{GAIN} \cdot |F^*| + \mathbf{LOSS} \cdot \Lambda(F^*, S)$ in the set of all decompositions of G consistent with S, where $\mathbf{GAIN} \geq 0$ and $\mathbf{LOSS} \geq 0$ are the weights of locus gain and locus loss events, resp.*

Such decompositions we call *optimal*. In the same way, for conditional consistency, we define the *Conditional Locus Tree Inference* problem (*CLTI*). The problems are equivalent if the input species tree is binary. From the algorithmic point of view, LTI is similar to the reconciliation with DTL scenarios [1] with no duplications. A transfer event corresponds to the creation of a tree in a decomposition forest. Additionally, we do not count loss event at the root of a new tree.

Our algorithm consists of several functions of $g \in G$, $s \in S$ and $\iota \in \{0, 1\}$ which denotes whether a set of cuts detaches g:

D1. $\delta(g, s, 0)$ is the minimal partial cost contribution of $G(g)$ in the set of all g-detaching sets X such that $M^X(g) = s$.

D2. $\delta(g, s, 1)$ as above but for non-g-detaching sets of cuts.

D3. $\delta^{\triangle}(g, s, \iota)$ is the minimal value of $\delta(g, s', \iota)$ for $s' \preceq s$.

D4. $\delta^{\uparrow}(g, s, \iota)$ is the minimal partial cost contribution of $G(g)$ in the set of all g-detaching sets of cuts X such that $M^X(g) \preceq s$. For $\iota = 1$, the cost additionally includes all losses created by $\sigma^X(g)$ and associated with every species node s' satisfying $M^X(g) \prec s' \preceq s$.

Let $c(v)$ be the set of children of v ($\emptyset$ for leaves). By $\mathbf{1}$ we denote the indicator function, that is, $\mathbf{1}[p]$ is 1 if p is satisfied and 0 otherwise. Then, we have the following dynamic programming formula (*DP algorithm*) that solves LTI:

$$\delta(g, s, \iota) = \begin{cases} 0 & \text{if } g \text{ is a leaf and } M(g) = s, \\ \min\{\alpha, \gamma\} & \text{if } g \text{ is not a leaf}, \\ +\infty & \text{otherwise}, \end{cases}$$

where

$$\begin{aligned} \alpha &= \mathbf{1}[c(s) \geq 3] \cdot \mathbf{LOSS} \cdot \iota + \min_{s', s'' \in c(s) \text{ and } s' \neq s''} \delta^{\uparrow}(g', s', 1) + \delta^{\uparrow}(g'', s'', 1), \\ \gamma &= \mathbf{GAIN} + \min(\delta^{\triangle}(g', M(g'), 0) + \delta^{\uparrow}(g'', s, \iota), \delta^{\triangle}(g'', M(g''), 0) + \delta^{\uparrow}(g', s, \iota)), \\ \delta^{\uparrow}(g, s, \iota) &= \begin{cases} \delta(g, s, \iota) & \text{if } s \text{ is a leaf}, \\ \min\{\delta(g, s, \iota), \mathbf{1}[|c(s)| > 1] \cdot \mathbf{LOSS} \cdot \iota + \min_{x \in c(s)} \delta^{\uparrow}(g, x, \iota)\} & \text{otherwise}, \end{cases} \\ \delta^{\triangle}(g, s, \iota) &= \min\{\delta(g, s, \iota), \min_{x \in c(s)} \delta(g, x, \iota)\}. \end{aligned}$$

▶ **Theorem 3.** *For every G and S: $\Delta(G, S) = \min_{s \in S} \delta^{\triangle}(\mathrm{root}(G), s, 0)) + \mathbf{GAIN}$.*

Proof. The proof is by induction on the structure of G and S, where the properties D1–D4 of all δ's are proved. We omit technical details. ◀

▶ **Theorem 4.** *The optimal cost can be computed in $O(|G||S|m)$ time and $O(|G||S|)$ space, where m is the maximal degree of a node from S.*

Proof. *Time:* We show that all values of δ functions can be computed in $O(m)$ time. This is straighforward for all values except α, where computing min potentially requires $O(m^2)$ time. This can be done, however, in $O(m)$ time, by finding for each node g' of G, the two children of s with the minimal and the second minimal value of $\delta^{\uparrow}$ and choosing the minimal pair one among all four variants. *Space:* Obvious. ◀

CLTI can be solved by an algorithm similar to the one presented above. It requires additional case in δ for resolving duplications. In order to model a proper binarization of a polytomy in $M(g)$, both children of g have to be mapped into a disjoint sets of children of $M(g)$. Such solution requires extending all δ's by a set of species nodes allowed for the mappings. In consequence, this approach has an exponential time and space complexity. In general we do not know if there is a polynomial time algorithm for CLTI. However, when the locus gain weight (**GAIN**) is much greater than the loss weight (**LOSS**), an efficient heuristic can be constructed, based on a mapping introduced in the next section.

4 Ranked Trees and Rank-based Mappings

Usually, when comparing trees, mappings based on their topologies are used (e.g. the lca-mapping). However, biological species trees contain additional useful structure: the *taxonomic ranks*, like species, genus, or family. Several major ranks are common to almost all living organisms. In this section, we propose a mathematical formalization of ranks and two rank-based mappings, which are useful in duplication inference and CLTI.

A *ranked species tree* is a species tree S in which every node s of S has assigned a small positive integer called *rank*, denoted $R(s)$, such that, for every s and s', if $s \prec s'$ then $R(s) < R(s')$. We assume that the rank of the root of is $d > 0$ and every leaf has rank 1.

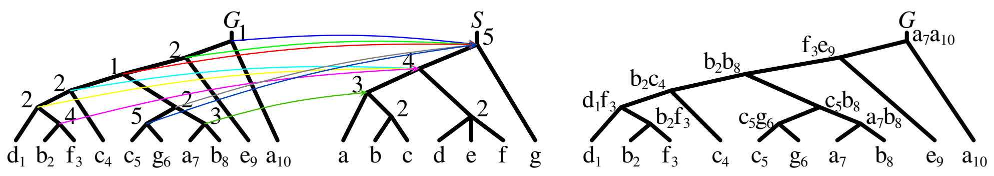


Figure 2 An example of a gene tree G and a species tree S. *Left:* The lca-mapping M. Each internal node of S is decorated with its rank based on the height of the corresponding subtree. Each internal node of G is decorated with the value of mapping P. *Right:* G in which each internal node is decorated by a pair of gene leaves that induces the value of mapping P in Alg. 1 (line 7). For example, for the left child of the root, say x, f_3e_9 yields $P(x) = R(\text{lca}_S(f, e)) = 2$.

Let G be a gene tree and S be a ranked species tree. For a rank r and a leaf t in S, the unique directed path in S consisting of all taxa comparable with t having the rank lower than r will be called *an (evolutionary) r-lineage* of t. Note that every 1-lineage is empty. We say that leaf-taxa t and t' are *separated* by the rank r if for every x from the r-lineage of t and every y from the r-lineage of t', x and y are incomparable. Observe that every pair of leaf-taxa is separated by the rank of 1. Moreover, if r separates t and t' then every rank lower than r also separates t and t'. For example, in Fig. 2 leaf-taxa a and c are separated by ranks 1, 2 and 3, but not by rank 4.

Let g be an internal node in G with children g_1 and g_2. The *highest separating rank* mapping $P\colon V_G \to \{1, 2, \ldots, d\}$ is defined as

$$P(g) = \max\{r\colon r \text{ separates every pair of leaf-taxa from } tax(g_1) \times tax(g_2)\}. \tag{1}$$

The *lowest common rank* mapping $I\colon V_G \to \{1, 2, \ldots, d\}$ is defined as $I(g) = R(M(g))$. We now present some basic properties of both mappings. Simple proofs are omitted for brevity.

▶ **Lemma 5.** *Let $\rho(t, t')$ be the highest rank that separates leaf-taxa t and t' and let g be an internal node of G with two children g_1 and g_2. Then:*

(A) *For every leaf-taxa t and t', $\rho(t, t') = R(\text{lca}_S(t, t'))$.*
(B) *$P(g) = \min\{\rho(t, t') : \langle t, t'\rangle \in tax(g_1) \times tax(g_2)\}$.*
(C) *$I(g) = \max\{R(\text{lca}_S(t, t')) : \langle t, t'\rangle \in tax(g_1) \times tax(g_2)\}$.*
(D) *$P(g) = \min\{R(\text{lca}_S(t, t')) : \langle t, t'\rangle \in tax(g_1) \times tax(g_2)\}$.*
(E) *$P(g) = 1$ if and only if $\text{tax}(g_1) \cap \text{tax}(g_2) \neq \emptyset$.*

Taxonomic ranks have been used earlier for HGT detection [11]. In this work, the authors decorated nodes of the gene tree with the rank of the lowest taxon shared by each descendant leaf, equivalent with the I mapping. A high difference between the rank of a node and the one of its parent was one of the premisses for HGT. To the best of our knowledge, no mapping equivalent with the highest separating rank has been proposed to day.

4.1 Computing mappings

Given a species tree S and a gene tree G, to compute I we can use the classical algorithm for lca-queries, in which, after a linear-time preprocessing, computing lca-queries can be completed in constant time [2]. We conclude that I can be computed in $O(|G| + |S|)$ time.

A naïve algorithm for computing P, based on Lemma 5, requires $O(|G||S|^2)$ time. Here, we propose an $O(d|G| + |S|)$ time solution. For two distinct leaves l_1 and l_2 of G, we write $l_1 <_p l_2$ if l_1 is visited earlier than l_2 in prefix traversal of G. For instance, in Fig. 2 the leaves are linearly ordered starting from the left, i.e., $d_1 <_p b_2 <_p f_3 <_p \ldots$.

Algorithm 1 Computing P

```
1: Input: A ranked tree $S$ with maximal rank $d$, a gene tree $G$ such that every leaf of $G$ has an
   attribute map equal to its label, i.e., a leaf from $S$. A parent of a node is denoted by attr. parent
   (None for the root).
   Output: Values of $P$ stored in attr. P of each internal node of $G$.
2: For $s$ in $S$.nodes: Let $s$.lastvisited := None # initialize last visited leaf
3: For $g$ in $G$.nodes: If $g$ is a leaf Then $v$.smap := $v$.map Else $g$.P := None # init P and smap
4: Initialize data structure in $G$ for lca-queries $\text{lca}_G(x, y)$ in $G$.
5: For rank in $1, 2, \dots, d$: For $v$ in $G$.leaves in prefix order such that $v$.smap.rank = rank:
6:         If not $v$.smap.lastvisited = None Then
7:             $g$ := $\text{lca}_G(v, v$.smap.lastvisited)
8:             If $g$.P = None Then $g$.P := rank # $P$ assignment
9:         $v$.smap.lastvisited := $v$ # update last visited
10:        $v$.smap := $v$.smap.parent # climb in $S$
```

▶ **Lemma 6.** *For a fixed $s \in S$, the sequence of all assignment evaluations in line 9 such that v.smap $= s$ induces a sequence of values v, denoted by $v_1, v_2, \dots, v_k$ such that:* (I) *the assignment s.lastvisited $:= v_i$ is executed only when* rank $= s$.rank, (II) $v_1 <_p v_2 <_p \cdots <_p v_k$, *and* (III) $\{v_1, v_2, \dots, v_k\} = M^{-1}(L(s))$.

Proof. (I) is obvious by the condition in the second loop. By the condition in the inner loop, the order of leaves induced by a sequence of such assignments follows $<_p$. For every gene leaf v, v.smap is initially set to the label of v, i.e., $M(v)$ (see line 3). Thus, if s is a leaf, i.e., s.rank $= 1$, then the assignment in line 9 sets the value of v.lastvisited if and only if the label of v is s. Thus for the leaves, (II) is satisfied. For (III), note that the line 10, ensures that every leaf v is assigned once to s.lastvisited of every node s of a species tree that is present on the path starting from $M(v)$ and terminating in the root. Hence, $M^{-1}(L(s)) \subseteq \{v_1, v_2, \dots, v_k\}$. The other inclusion follows trivially from the fact that for a leaf v, v.smap is originally set to $M(v)$ and v cannot be assigned to a node incomparable with $M(v)$. ◀

▶ **Lemma 7.** *For every internal node g, $P(g) = g$.P.*

Proof. Let g' and g'' be the left and the right child of g, respectively. The proof is by induction on the rank $r = 1, 2, \dots, d$. Let $r = 1$. Assume that $P(g) = 1$, we show that g.P $= 1$. Let $s \in \text{tax}(g') \cap \text{tax}(g'')$. Then, by Lemma 6, let Λ_s be the sequence $\{v_1, v_2, \dots, v_k\}$ of all leaves assigned to s.lastvisited such that $M(v_i) = s$ and ordered by $<_p$. Clearly, the list has the leaves from both subtrees of g, thus there is an index $j < k$, such that $v_j \in L(g')$ and $v_{j+1} \in L(g'')$. Thus $\text{lca}_G(v_j, v_{j+1}) = g$. Now, in line 8, when v is v_{j+1} then v.smap.lastvisited is v_j. In such a case, either g.P is None and g.P will be set to 1, or g.P is already set, however, it can be only 1. This completes the first part of the proof.

Assume that $P(g) = r$ and for every q, such that $P(q) < r$, we have $P(q) = q.P$. For every $v \in L(g')$ and $w \in L(g'')$, $R(\text{lca}_S(M(v), M(w))) \geq r$, thus $g.P =$ None, when Alg. 1 starts the main loop with rank $= r$. From Lemma 5, there is a pair taxa $\langle t_1, t_2 \rangle \in \hat{g}$ such that $s = \text{lca}_S(t, t')$ and $R(s) = r$. Thus, there are two leaves a_1 and a_2 in G such that for each i, $M(a_i) = t_i$ and $\text{lca}(a_1, a_2) = g$, i.e. $a_1 \preceq g'$ and $a_2 \preceq g''$. Similarly, to the first step, the leaves from $M^{-1}(L(s))$ are all visited and set to s.lastvisited according to the order $<_p$. The sequence contains elements a_1 and a_2, therefore again there is j separating leaves from both subtrees of g. The rest of the proof is analogous: in line 8 either $g.P$ is already set to r (if there was other s', processed before s, with $R(s') = r$ satisfying the same properties as s) or it will be set in to r. This completes the proof. ◀

▶ **Lemma 8.** *Alg. 1 requires $O(d|G| + |S|)$ time and $O(|G| + |S|)$ space.*

Proof. *Time:* Lines 2–5 have $O(|G| + |S|)$ time complexity, while the body of the inner loop needs $O(1)$ time. *Space:* Alg. 1 uses only a few node attributes plus the lca-query data structure of the size $O(|G|)$. ◀

5 Classification of Gene Duplications

Several methods for reconciliation with non-binary gene trees have been proposed [20, 16, 19, 6, 3, 9]. However, reconciliation with non-binary species trees is harder to model. This is because a polytomy may represent a lack of knowledge about the order of speciations, and therefore some duplication nodes may correspond to biological speciations. This motivates a further classification of duplication nodes into conditional and required duplications [18].

When reconciling a gene tree G with every binarization of S, if g from G is a duplication in every reconciliation, then g is called a *required duplication*. Similarly, if g is a duplication in at least one but not all reconciliations, we say that g is a *conditional duplication*. Note that G is conditionally embeddable in S if and only if each node in G is either a speciation or a conditional duplication.

In this section, we show how P and I can be used to solve the problem of gene duplication classification when the species tree has possible polytomies.

▶ **Lemma 9.** *For an internal node g from a gene tree G, the following conditions are equivalent:*

(A1) *$P(g) = I(g)$,*

(A2) *every subtree rooted below $M(g)$ contains taxa from at most one child of g, i.e., for every $s \prec M(g)$, if $L(s) \cap \mathrm{tax}(g_1) \neq \emptyset$ then $L(s) \cap \mathrm{tax}(g_2) = \emptyset$, where g_1 and g_2 are the children of g, and*

(A3) *for every $\langle t, t' \rangle \in tax(g_1) \times tax(g_2)$, $\mathrm{lca}_S(t, t') = M(g)$.*

Proof. *(A1) ⇒ (A2).* Assume that $s \prec M(g)$ and there are two leaves t and t' in $L(s)$ such that $t \in \mathrm{tax}(g_1)$ and $t' \in \mathrm{tax}(g_2)$. Hence, $\langle t, t' \rangle \in tax(g_1) \times tax(g_2)$ and $\mathrm{lca}_S(t, t') \preceq s \prec M(g)$. Thus, $P(g) < I(g)$, a contradiction. *(A2) ⇒ (A3).* Let $\langle t, t' \rangle \in \hat{g}$. Then, t and t' are leaves from two different subtrees rooted below $M(g)$. Therefore, $\mathrm{lca}_S(t, t') = M(g)$. *(A3) ⇒ (A1).* It follows immediately from Lemma 5. ◀

Note that the above Lemma holds also when $P(g) = I(g) = 1$, i.e. when an internal node g is mapped to a leaf of S. In such a case the condition (A2) is satisfied trivially.

▶ **Lemma 10.** *Let $P(g) = I(g)$. Then, g is a speciation iff $M(g)$ is an internal node and there are exactly two subtrees rooted at children of $M(g)$ having nodes from $\mathrm{tax}(g)$.*

Proof. (⇒). We have that g is an internal node. In such a case $I(g) > 1$ and $M(g)$ is an internal node. Then, by (A2) from Lemma 9, every child of $M(g)$ has taxa present in at most one child of g. Clearly, there are at least two children of $M(g)$ satisfying this property. If there are more than two, then one child of g, say g_1, has taxa from at least two children of $M(g)$. Hence, $M(g_1) = M(g)$ and g is a duplication node, a contradiction. (⇐). Similarly, if $M(g)$ is an internal node, then by (A2), the mappings of the children of g are incomparable and located below $M(g)$, therefore g cannot be a duplication. ◀

We have a symmetric property whose proof is similar to the previous one.

▶ **Lemma 11.** *Assume that $P(g) = I(g)$. Then, g is a duplication node if and only if either $M(g)$ is a leaf or $M(g)$ is an internal node and there are at least three subtrees rooted at a child of $M(g)$ having nodes from $\operatorname{tax}(g)$.*

Finally, we have the main property.

▶ **Theorem 12** (Classification Theorem). *Let g be an internal node of G. Then:*
(C1) *If $P(g) = I(g) = 1$ or $P(g) < I(g)$ then g is a required duplication.*
(C2) *If $P(g) = I(g) > 1$, then g is a duplication if and only if g is a conditional duplication.*

Proof. *(C1)* If $P(g) = I(g) = 1$, then g is mapped to a leaf. Hence, every leaf below g has the same label. Thus, in every binarization of S, g is a duplication. Assume that $P(g) < I(g)$. Then $M(g)$ is an internal node in S, having at least three taxa in $L(M(g))$ (otherwise, the two children of $M(g)$ are leaves and $P(g) = I(g) = 2$). We can assume that there are three leaves $t, t' \in \operatorname{tax}(g_1)$ and $t'' \in \operatorname{tax}(g_2)$ such that $\operatorname{lca}_S(t', t'') \prec \operatorname{lca}_S(t, t', t'')$. Clearly, this property holds for every binarization T of S, where the possible polytomy $M(g)$ is resolved. Moreover, in every T, $M(g_1) \succeq \operatorname{lca}_T(t, t', t'')$, thus $M(g_1)$ is comparable with $M(g_2) \succeq t''$. Thus, $M(g) = \max(M(g_1), M(g_2))$ and g is a duplication node.

(C2,⇐). If g is a conditional duplication, then it is a duplication by the definition. *(C2,⇒).* Assume that g is a duplication, then by condition (A2) from Lemma 9, the children of $M(g)$ can be clustered into three disjoint sets X, X' and X'' such that every node from X has no taxa present in $\operatorname{tax}(g)$, every node of X' has taxa from $\operatorname{tax}(g')$ but not from $\operatorname{tax}(g'')$ and analogously every node of X'' has taxa from $\operatorname{tax}(g'')$ but not from $\operatorname{tax}(g')$, where g' and g'' are the children of g. In addition, by Lemma 11 at least one among X' and X'', say X', has at least two elements. Consider a binary tree T in $H(M(g))$, such that all elements of X' and X'' are located in the left and the right subtree of T, respectively. Then, $\operatorname{lca}_T(X')$ and $\operatorname{lca}_T(X'')$ are incomparable. Thus, in such a binarization of S, g' and g'' maps below $M(g)$, and g is a speciation node. Similarly, it can be shown that there exist a tree in $H(M(g))$ in which g is a duplication. ◀

Based on Alg. 1, Classification Theorem leads to a natural $O(d|G| + |S|)$ time solution for the inference of required and conditional duplications when reconciling a given binary gene tree with a species tree. This improves known $O(|G|(d + m) + |S|)$ time algorithm from [18], where m is the maximal degree of a node from S. The improvement is beneficial for highly polytomic species trees. For example, as of 04.28.2017, the genus *Aspergillus* has 1950 children species in the NCBI Taxonomy.

6 Heuristic for CLTI

In this section, we propose the heuristic algorithm for CLTI when the locus gain weight is much greater than the loss weight. The algorithm is based on the following lemma, which follows directly from Theorem 12:

▶ **Lemma 13.** *Tree G is conditionally embeddable in S if and only if for all internal nodes g in G, $I(g) = P(g) > 1$.*

Alg. 2 is a greedy approach that iteratively finds the minimal nodes g such that $P(g) < I(g)$ or $I(g) = 1$ and detaches an embeddable subtree below each node. Note the following:

▶ Remark. Let G be conditionally embeddable in S. Let $\hat{\Lambda}(G, S) = |L(M(\operatorname{root}(G)))| - |L(G)|$. Then, $\Lambda(G, S) \leq \hat{\Lambda}(G, S)$.

Algorithm 2 Heuristic algorithm for CLTI

```
1: Input: A ranked tree $S$ and a gene tree $G$.
   Output: A decomposition forest of $G$ conditionally consistent with $S$.
   Initialize: $F := \emptyset$
2: Compute $I$ and $P$ in $G$.
3: Let $Z$ be the set of all minimal nodes $g$ in $G$ such that $G(g)$ is not conditionally embeddable
   (i.e. $P(g) < I(g)$ or $I(g) = 1$). If $Z$ is empty Then Return $F \cup \{G\}$
4: For every $g$ in $Z$:
5:     Let $\langle v, w\rangle$ be the edge incident to a child of $g$ with minimal $\hat{\Lambda}(G(w), S) + \hat{\Lambda}(G(g) \setminus G(w), S)$
       such that $G(w)$ and $G(g) \setminus G(w)$ are conditionally embeddable.
       Add $G(w)$ to $F$, remove $e$ and $G(w)$ from $G$.
6: Repeat steps 2-6 until tree $G$ is empty.
```

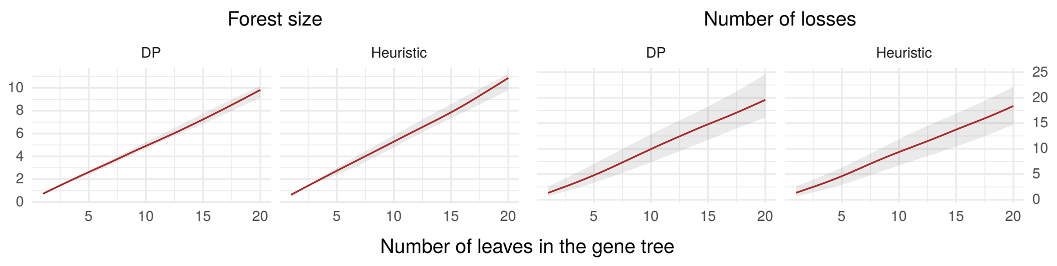


Figure 3 Comparison of DP and heuristic algorithms for binary species trees in terms of forest size $|F|$ and numbers of losses $\Lambda(F, S)$. The brown line depicts the median cost; the grey ribbon depicts the 90% confidence interval. The weights in the DP algorithm have been set to $\mathbf{GAIN} = 1000, \mathbf{LOSS} = 1$. The plots have been smoothed with cubic splines.

Let $T_1 \setminus T_2$ denote tree T_1 with detached subtree T_2. Then, $\hat{\Lambda}(G', S) + \hat{\Lambda}(G(g) \setminus G', S)$ is an estimate of the partial loss cost induced by detaching subtree G'. The detached subtree in Alg. 2 is chosen so as to minimize this estimate. To limit the complexity of a single step, we consider only subtrees rooted at vertices at a close neighbourgood of g.

A major advantage of Alg. 2 is the space complexity, which is $O(|G| + |S|)$. This makes the heuristic suitable for trees with hundreds or thousands of nodes. The time complexity is $O(ad|G| + a|S|)$, where a is the number of recomputations of I and P mappings. Pessimistically, $a = O(|G|)$, which makes this algorithm assymptotically quadratic. However, in applications this number is expected to be a small integer.

6.1 Experimental validation

In the case of binary species trees, conditional embeddability is identical to strict embeddability, and both locus tree inference algorithms can be compared experimentally.

For each $|L(G)| = 1, \ldots, 20$ and $|L(S)| = 10$ we have generated 100 pairs of random trees under the Yule-Harding model. The leaves of G have been assigned to leaves of S randomly. The numbers of losses for heuristic algorithm have been computed using a modification of the DP algorithm. The inferred costs are shown in Fig. 3.

The costs are similar for both algorithms. The forest size is approximately half the number of leaves in G. Using linear regression, we have determined that, on average, the inferred forest size is equal to $0.47|L(G)|$ for DP and $0.53|L(G)|$ for the heuristic. The number of losses is slightly smaller for the heuristic algorithm (on average $0.98|L(G)|$ for DP and $0.90|L(G)|$ for the heuristic). The reason for this is that the greedy approach tends to detach more concise trees.

7 Example of evolutionary history decomposition

We have compared our approach with a state-of-the-art reconciliation program, Notung 2.9 [17]. We have analyzed the evolution of the gene family of an aminotransferase from a fungus *Penicillium lilacinoechinulatum*[1] (GenBank: ABV48733.1), which has been earlier reported to undergo a HGT [15]. The gene tree has been reconciled with NCBI Taxonomy with loss weight 1, duplication weight 1.5 and transfer weight varying from 3 to 8. The result of decomposition by our heuristic approach is depicted in Fig. 4. Depending on the transfer weight, Notung 2.9 reported from 1 to 7 transfers and numerous duplications, while our heuristic inferred a unique result.

Several biological conclusions can be drawn from the decomposition. There are two probable HGTs: into the ancestors of family Clavicipitaceae (blue subtree) and genus *Fusarium* (turquoise subtree). Those groups are distantly related to species in their "mother locus" subtrees, which makes multiple duplications and losses less likely than a HGT. Since both recipient groups are pathogenic (as well as *Aspergillus fumigatus*, in which the gene has been extensively duplicated), we may expect that the protein ABV48733.1 plays a role in their pathogenesis. Both transfers are consistent with reconciliation results.

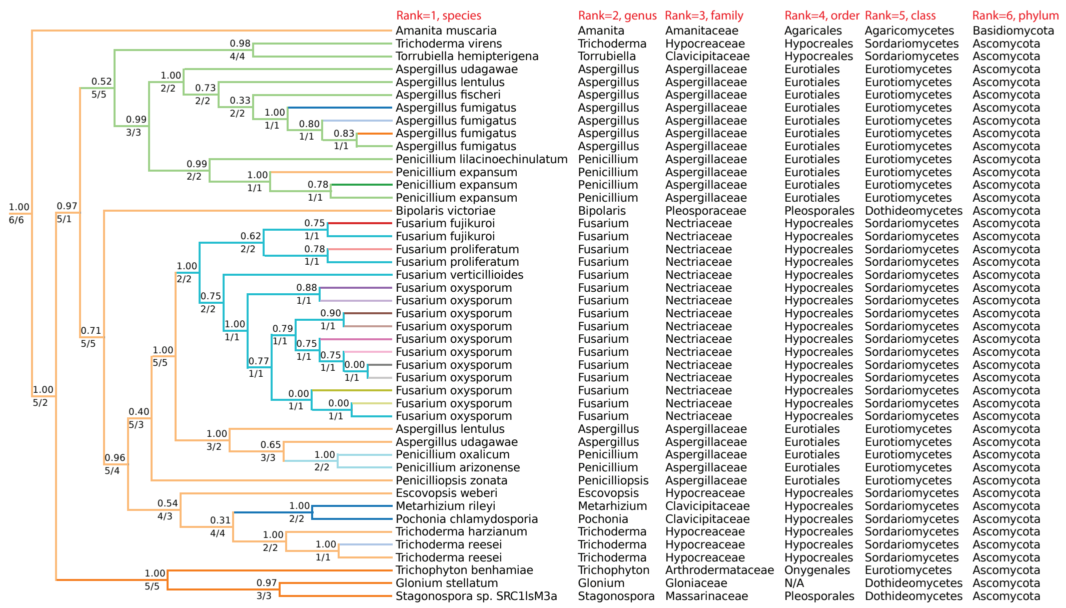
Note that the P mappings of parents of turquoise and blue subtrees are much higher than the ones of their children, consistent with a HGT hypothesis. On the contrary, the P mapping of parent of the green subtree is much lower than the ones of its children, consistent with a duplication. This observation can be a basis for an event scoring system to aid the classification of the events. Other locus gain events are ambiguous, both in the case of history decomposition analysis and the tree reconciliation.

8 Discussion

In this work, we have investigated two new problems for locus tree inference in a parsimony framework. We have proposed and analyzed a new mapping, called the Highest Separating Rank, which has been applied to the problems of duplication classification and locus tree inference. The solution to the duplication classification problem has improved the current one by removing dependence on the maximum node degree in species tree from the time complexity. A prototype implementation is publicly availiable at `https://github.com/mciach/LocusTreeInference`. The presented approach will be used to obtain a manually curated dataset of horizontally transferred genes.

Future outlooks. The influence of potential contradictory binarizations on the decomposition needs to be elucidated. LTI should be generalized for non-binary gene trees, as it would allow to collapse nodes with low support, possibly decreasing the forest size. The P mapping can be applied to obtain efficient solutions to the problem of gene tree rooting and supertree construction. Finally, application of automatic event scoring system should be investigated.

[1] Homologs of the protein sequence have been found in 20 fungal species using BLASTp suite. The sequences have been aligned using MAFFT program and trimmed with TrimAL. The phylogenetic tree has been created using PhyML and rooted by setting *Amanita muscaria* as the outgroup. Nodes of the species tree have been collapsed to represent only the following taxonomic ranks: species, genus, family, order, class, phylum, kingdom.



Rank=1, species	Rank=2, genus	Rank=3, family	Rank=4, order	Rank=5, class	Rank=6, phylum
Amanita muscaria	Amanita	Amanitaceae	Agaricales	Agaricomycetes	Basidiomycota
Trichoderma virens	Trichoderma	Hypocreaceae	Hypocreales	Sordariomycetes	Ascomycota
Torrubiella hemipterigena	Torrubiella	Clavicipitaceae	Hypocreales	Sordariomycetes	Ascomycota
Aspergillus udagawae	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus lentulus	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus fischeri	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus fumigatus	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus fumigatus	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus fumigatus	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus fumigatus	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicillium lilacinoechinulatum	Penicillium	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicillium expansum	Penicillium	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicillium expansum	Penicillium	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicillium expansum	Penicillium	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Bipolaris victoriae	Bipolaris	Pleosporaceae	Pleosporales	Dothideomycetes	Ascomycota
Fusarium fujikuroi	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium fujikuroi	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium proliferatum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium proliferatum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium verticillioides	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Fusarium oxysporum	Fusarium	Nectriaceae	Hypocreales	Sordariomycetes	Ascomycota
Aspergillus lentulus	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Aspergillus udagawae	Aspergillus	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicillium oxalicum	Penicillium	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicillium arizonense	Penicillium	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Penicilliopsis zonata	Penicilliopsis	Aspergillaceae	Eurotiales	Eurotiomycetes	Ascomycota
Escovopsis weberi	Escovopsis	Hypocreaceae	Hypocreales	Sordariomycetes	Ascomycota
Metarhizium rileyi	Metarhizium	Clavicipitaceae	Hypocreales	Sordariomycetes	Ascomycota
Pochonia chlamydosporia	Pochonia	Clavicipitaceae	Hypocreales	Sordariomycetes	Ascomycota
Trichoderma harzianum	Trichoderma	Hypocreaceae	Hypocreales	Sordariomycetes	Ascomycota
Trichoderma reesei	Trichoderma	Hypocreaceae	Hypocreales	Sordariomycetes	Ascomycota
Trichoderma reesei	Trichoderma	Hypocreaceae	Hypocreales	Sordariomycetes	Ascomycota
Trichophyton benhamiae	Trichophyton	Arthrodermataceae	Onygenales	Eurotiomycetes	Ascomycota
Glonium stellatum	Glonium	Gloniaceae	N/A	Dothideomycetes	Ascomycota
Stagonospora sp. SRC1lsM3a	Stagonospora	Massarinaceae	Pleosporales	Dothideomycetes	Ascomycota

Figure 4 Gene tree of homologs of protein $ABV48733.1$ and evolutionary lineages of organisms. Numbers above branches represent node supports, numbers below branches represent the values of I/P mappings before decomposition. The separate histories of different loci have been highlited by different colors.

References

1. Mukul S. Bansal, Eric J. Alm, and Manolis Kellis. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28(12):i283–i291, 2012.
2. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94. Springer, 2000.
3. Ann-Charlotte Berglund-Sonnhammer, Pär Steffansson, Matthew J. Betts, and David A. Liberles. Optimal gene trees from sequences and species trees using a soft interpretation of parsimony. *Journal of Molecular Evolution*, 63(2):240–250, 2006.
4. P. Bonizzoni, G. Della Vedova, and R. Dondi. Reconciling a gene tree to a species tree under the duplication cost model. *Theoretical Computer Science*, 347(1-2):36–53, 2005.
5. Jean-Philippe Doyon, Vincent Ranwez, Vincent Daubin, and Vincent Berry. Models, algorithms and programs for phylogeny reconciliation. *Briefings in Bioinformatics*, 12(5):392, 2011.
6. O. Eulenstein, S. Huzurbazar, and D. A. Liberles. *Evolution after Gene Duplication*, chapter Reconciling Phylogenetic Trees, pages 185–206. John Wiley & Sons, Inc., 2010.
7. P. Górecki and J. Tiuryn. DLS-trees: A model of evolutionary scenarios. *Theoretical Computer Science*, 359(1-3):378–399, 2006.
8. Patrick J. Keeling and Jeffrey D. Palmer. Horizontal gene transfer in eukaryotic evolution. *Nature Reviews Genetics*, 9(8):605–618, 2008.
9. Manuel Lafond, Krister M. Swenson, and Nadia El-Mabrouk. An optimal reconciliation algorithm for gene trees with polytomies. In *International Workshop on Algorithms in Bioinformatics*, pages 106–122. Springer, 2012.
10. Marina Marcet-Houben and Toni Gabaldón. Treeko: a duplication-aware algorithm for the comparison of phylogenetic trees. *Nucleic Acids Research*, page gkr087, 2011.

11 Miguel A Naranjo-Ortíz, Matthias Brock, Sascha Brunke, Bernhard Hube, Marina Marcet-Houben, and Toni Gabaldón. Widespread inter-and intra-domain horizontal gene transfer of d-amino acid metabolism enzymes in eukaryotes. *Frontiers in Microbiology*, 7, 2016.

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

13 Matthew D. Rasmussen and Manolis Kellis. Unified modeling of gene duplication, loss, and coalescence using a locus tree. *Genome Research*, 22(4):755–765, 2012.

14 Matt Ravenhall, Nives Škunca, Florent Lassalle, and Christophe Dessimoz. Inferring horizontal gene transfer. *PLOS Computational Biology*, 11(5):1–16, 05 2015.

15 Thomas A. Richards, Guy Leonard, Darren M. Soanes, and Nicholas J. Talbot. Gene transfer into the fungi. *Fungal Biology Reviews*, 25(2):98–110, 2011.

16 M. J. Sanderson and M. M. McMahon. Inferring angiosperm phylogeny from EST data with widespread gene duplication. *BMC Evolutionary Biology*, 7 (Suppl 1): S3, 2007.

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

18 Benjamin Vernot, Maureen Stolzer, Aiton Goldman, and Dannie Durand. Reconciliation with non-binary species trees. *Journal of Computational Biology*, 15(8):981–1006, 2008.

19 Tandy Warnow. Large-scale multiple sequence alignment and phylogeny estimation. In *Models and Algorithms for Genome Evolution*, pages 85–146. Springer, 2013.

20 Yu Zheng and Louxin Zhang. Reconciliation with non-binary gene trees revisited. In *International Conference on Research in Computational Molecular Biology*, pages 418–432. Springer, 2014.

An IP Algorithm for RNA Folding Trajectories*

Amir H. Bayegan[1] and Peter Clote[2]

1 **Boston College, Biology Department, Chestnut Hill, MA, USA**
`a.h.bayegan@gmail.com`
2 **Boston College, Biology Department, Chestnut Hill, MA, USA**
`clote@bc.edu`

Abstract

Vienna RNA Package software `Kinfold` implements the Gillespie algorithm for RNA secondary structure folding kinetics, for the move sets MS_1 [resp. MS_2], consisting of base pair additions and removals [resp. base pair addition, removals and shifts]. In this paper, for arbitrary secondary structures s, t of a given RNA sequence, we present the first optimal algorithm to compute the shortest MS_2 folding trajectory $s = s_0, s_1, \ldots, s_m = t$, where each intermediate structure s_{i+1} is obtained from its predecessor by the addition, removal or shift of a single base pair. The shortest MS_1 trajectory between s and t is trivially equal to the number of base pairs belonging to s but not t, plus the number of base pairs belonging to t but not s. Our optimal algorithm applies integer programming (IP) to solve (essentially) the minimum feedback vertex set (FVS) problem for the "conflict digraph" associated with input secondary structures s, t, and then applies topological sort, in order to generate an optimal MS_2 folding pathway from s to t that maximizes the use of shift moves. Since the optimal algorithm may require excessive run time, we also sketch a fast, near-optimal algorithm (details to appear elsewhere). Software for our algorithm will be publicly available at `http://bioinformatics.bc.edu/clotelab/MS2distance/`.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Integer programming, RNA secondary structure, folding trajectory, feedback vertex problem, conflict digraph

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.6

1 Introduction

In this paper, we introduce the first algorithm to compute the MS_2 distance between two secondary structures s and t of a given RNA sequence $a_1, \ldots, a_n$; i.e. the length m of the shortest refolding trajectory $s = s_0, s_1, \ldots, s_m = t$, in which each intermediate secondary structure s_{i+1} is obtained from s_i by a single base pair addition, removal or shift. Here a *shift* transforms a base pair (x, y) to the base pair (x', y'), where either $x \in \{x', y'\}$ or $y \in \{x', y'\}$, but not both; see Figure 1 for an illustration of all possible types of shift moves. Although shifts are considered in the secondary structure folding kinetics program `Kinfold` [12] as well as in theoretical work on RNA molecular structure evolution [15], most papers on RNA secondary structure do not consider shift moves, presumably due to the sometimes tremendous additional complications even though the shift moves for helix zippering and defect diffusion are supported by experimental data [14]. Indeed, while our algorithm to compute the expected number of nearest neighbors with respect to MS_1

* This research was supported in part by National Science Foundation grant DBI-1262439 (PC). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

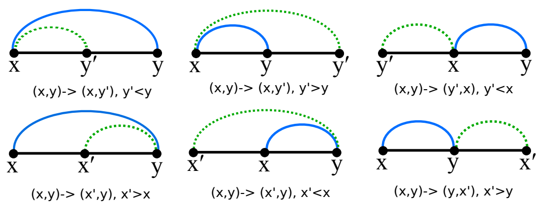


Figure 1 Shift moves from solid base pair to dotted base pair. Image taken from [2].

moves [1] is highly non-trivial, our analogous algorithm for MS_2 moves is far more complex [2]. Moreover, the current paper illustrates the enormous computational complexity that arises when considering MS_2 distance rather than MS_1 distance – while MS_1 distance, also known as *base pair distance*, is trivial to compute, we conjecture that MS_2 distance is NP-complete, where this problem can be formalized as a decision problem to determine, for any given secondary structures s, t and integer m, whether there is an MS_2 trajectory $s = s_0, s_1, \ldots, s_m = t$ of length $\leq m$, in which each intermediate secondary structure s_{i+1} is obtained from s_i by a single base pair addition, removal or shift. Here, we describe an exact (possibly exponential time) integer programming (IP) algorithm, and in a sequel to this paper, we will describe a fast, *near-optimal* algorithm, a greedy algorithm and a slow, exact branch-and-bound algorithm (details of these algorithm cannot be given here, due to space constraints). We conclude the current paper by a benchmarking comparison between the optimal IP algorithm and the near-optimal algorithm, and compare the values of MS_1, MS_2 and Hamming distance on a data set of 3800 random RNA sequences having random initial and random target structures of length n, computed for a range of values of n.

Since our algorithms involve the *feedback vertex set* problem for *RNA conflict digraphs*, we now provide a bit of background about this problem. Given a directed graph, or digraph, $G = (V, E)$, a *feedback vertex set* (FVS) is a subset $V' \subseteq V$ which contains at least one vertex from every directed cycle in G, thus rendering G acyclic. Similarly, a *feedback arc set* (FAS) is a subset $E' \subseteq E$ which contains at least one directed edge (arc) from every directed cycle in G. The FVS [resp. FAS] problem is the problem to determine a minimum size feedback vertex set [resp. feedback arc set] which renders G acyclic. Both the FVS and FAS are NP-complete for arbitrary digraphs, as well as for tournaments; indeed the FVS and FAS problems both appear in the list of 21 problems shown by R.M. Karp to be NP-complete [10]. We now introduce some necessary definitions.

Although the notion of secondary structure is well-known, we give three distinct but equivalent definitions, that will allow us to overload secondary structure notation to simplify presentation of our algorithms.

▶ **Definition 1** (Secondary structure as set of ordered base pairs). Let $[1, n]$ denote the set $\{1, 2, \ldots, n\}$. A secondary structure for a given RNA sequence $a_1, \ldots, a_n$ of length n is defined to be a set s of <u>ordered</u> pairs (i, j), with $1 \leq i < j \leq n$, such that the following conditions are satisfied.

1. *Watson-Crick and wobble pairs:* If $(i, j) \in s$, then $a_i a_j \in \{GC, CG, AU, UA, GU, UG\}$.
2. *No base triples:* If (i, j) and (i, k) belong to s, then $j = k$; if (i, j) and (k, j) belong to s, then $i = k$.
3. *Nonexistence of pseudoknots:* If (i, j) and (k, ℓ) belong to s, then it is not the case that $i < k < j < \ell$.
4. *Threshold requirement for hairpins:* If (i, j) belongs to s, then $j - i > \theta$, for a fixed value $\theta \geq 0$; i.e. there must be at least θ unpaired bases in a hairpin loop. Following standard convention, we set $\theta = 3$ for steric constraints.

If s is a secondary structure (set of ordered pairs), then $|s|$ denotes the size of s, i.e. the number of base pairs belonging to s.

Without risk of confusion, it will be convenient to overload the concept of secondary structure s with two alternative, equivalent notations, for which context will determine the intended meaning.

▶ **Definition 2** (Secondary structure as set of unordered base pairs)**.** A secondary structure s for the RNA sequence $a_1, \ldots, a_n$ is a set of unordered pairs $\{i, j\}$, with $1 \leq i, j \leq n$, such that the corresponding set of ordered pairs

$$\{i, j\}_< \overset{\text{def}}{=} (\min(i, j), \max(i, j)) \tag{1}$$

satisfies Definition 1. If s is a secondary structure (set of unordered pairs), then $|s|$ denotes the size of s, i.e. the number of base pairs belonging to s.

▶ **Definition 3** (Secondary structure as an integer-valued function)**.** A secondary structure s for $a_1, \ldots, a_n$ is a function $s : [1, \ldots, n] \to [0, \ldots, n]$, such that $\left\{\{i, s[i]\}_< : 1 \leq i \leq n, s[i] \neq 0\right\}$ satisfies Definition 1; i.e.

$$s[i] = \begin{cases} 0 & \text{if } i \text{ is unpaired in } s \\ j & \text{if } (i, j) \in s \text{ or } (j, i) \in s \end{cases} \tag{2}$$

▶ **Definition 4** (Secondary structure distance measures)**.** Let s, t be secondary structures of length n. Base pair distance is defined by equation (3) below, and Hamming distance is defined by equation (4) below.

$$d_{BP}(s, t) = |\{(x, y) : ((x, y) \in s \wedge (x, y) \notin t) \vee ((x, y) \in t \wedge (x, y) \notin s)\}| \tag{3}$$

$$d_H(s, t) = |\{i \in [1, n] : s[i] \neq t[i]\}| \tag{4}$$

We next define some primitive notions used later to define a central concept of *RNA conflict directed graph* (digraph). Let $[1, n]$ denote the set $\{1, \ldots, n\}$. Given secondary structure s on RNA sequence $\{a_1, \ldots, a_n\}$, we say that a position $x \in [1, n]$ is *touched* by s, or equivalently that the structure s *touches* the position x, if x belongs to a base pair of s, or equivalently $s[x] \neq 0$. Let BP_1 [resp. BP_2] denote the set of base pairs of s [resp. t] which are not touched by any base pair of t [resp. s]; i.e.

$$BP_1 = \{(i, j) \in s : t[i] = 0 = t[j]\} \tag{5}$$

$$BP_2 = \{(i, j) \in t : s[i] = 0 = s[j]\} \tag{6}$$

2 MS_2 distance between secondary structures

In this section, we present an integer programming (IP) algorithm to compute the MS_2 distance between any two secondary structures s, t, i.e. the minimum length of a trajectory from s to t, involving only base pair additions, removals and shifts. Since any shift move, such as $(x, y) \to (x, z)$ can be simulated by removal of the base pair (x, y) followed by addition of the base pair (x, z), our strategy to produce a minimum length MS_2 trajectory is to use graph-theoretic methods to *maximize* the number of shift moves and *minimize* the number of base pair additions and removals. The validity of this approach is formalized in the following simple theorem, whose proof is straightforward.

▶ **Theorem 5.** *Suppose that the MS_1 distance between secondary structures s, t is k, i.e. base pair distance $d_{BP}(s,t) = |s - t| + |t - s| = k$. Suppose that ℓ is the number of shift moves occurring in the shortest MS_2 trajectory $s = s_0, s_1, \ldots, s_m = t$ from s to t. Then the MS_2 distance between s and t equals*

$$d_{MS_2}(s,t) = \ell + (k - 2\ell) = k - \ell\,. \tag{7}$$

2.1 RNA conflict digraph

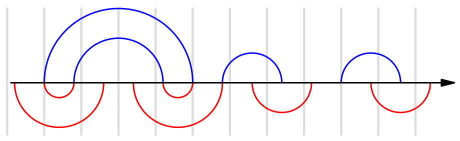
Throughout this section, we take s, t to be two arbitrary, distinct, but fixed secondary structures of the RNA sequence $a_1, \ldots, a_n$. To determine a minimum length MS_2 folding trajectory from secondary structure s to secondary structure t, we must maximize the number of shift moves and minimize the number of base pair additions and removals. To that end, note that the base pairs in s that do not touch any base pair of t must be removed in any MS_2 path from s to t, since there is no shift of such base pairs to a base pair of t – such base pairs are exactly those in BP_1, defined in equation (5). Similarly, note that the base pairs in t that do not touch any base pair of s must occur must be added, in the transformation of s to t, since there is no shift of any base pair from s to obtain such base pairs of t – such base pairs are exactly those in BP_2, defined in equation (6). We now focus on the remaining base pairs of s, all of which touch a base pair of t, and hence could theoretically allow a shift move in transforming s to t, *provided* that there is no base triple or pseudoknot introduced by performing such a shift move. Examples of all six possible types of shift move are illustrated in Figure 1. To handle such cases, we define the notion of *RNA conflict digraph*, solve the *feedback vertex set* (FVS) problem by integer programming (IP), apply topological sorting [3] to the acyclic digraph obtained by removing a minimum set of vertices occurring in feedback loops, then apply shift moves in topologically sorted order. We now formalize this argument.

Define the digraph $G = (V, E)$, whose vertices (or nodes) $n \in V$ are defined in the following Definition 6 and whose directed edges are defined in Definition 7.

▶ **Definition 6** (Vertex in an RNA conflict digraph). If s, t are distinct secondary structures for the RNA sequence $a_1, \ldots, a_n$, then a vertex in the RNA conflict digraph $G = G(s,t)$ is a triplet node, or more simply, node $v = (x, y, z)$ consisting of integers x, y, z, such that the base pair $\{x, y\}_< = (\min(x,y), \max(x,y))$ belongs to t, and the base pair $\{y, z\}_< = (\min(y,z), \max(y,z))$ belongs to s. Let $v.t$ [resp. $v.s$] denote the base pair $\{x, y\}_<$ [resp. $\{y, z\}_<$] belonging to t [resp. s]. The middle integer y of node $v = (x, y, z)$ is called the *pivot* position, since it is common to both s and t. Nodes are ordered by the integer ordering of their pivot positions: $(x, y, z) \preceq (x', y', z')$ if and only if $y \le y'$ (or $y = y'$ and $x < x'$, or $y = y'$, $x = x'$, and $z < z'$). If $v = (x, y, z)$ is a node, then $flatten(v)$ is defined to be the set $\{x, y, z\}$ of its coordinates.

Nodes are representations of a potential shift move, and can be categorized into six types, as shown in Figure 1.

▶ **Definition 7** (Directed edge in an RNA conflict digraph). The base pair $\{a, b\}_<$ is said to *cross* the base pair $\{c, d\}_<$ if either $\min(a,b) < \min(c,d) < \max(a,b) < \max(c,d)$ or $\min(c,d) < \min(a,b) < \max(c,d) < \max(a,b)$; in other words, base pairs cross if they form a pseudoknot. Base pairs $\{a, b\}_<$ and $\{c, d\}_<$ are said to touch if $|\{a, b\} \cap \{c, d\}| = 1$; in other words, base pairs touch if they form a base triple. There is a directed edge from node $n_1 = (x_1, y_1, z_1)$ to node $n_2 = (x_2, y_2, z_2)$, denoted $n_1 \to n_2$, if either $z_1 = x_2$ or the base pair $\{y_1, z_1\}_< \in s$ from n_1 crosses base pair $\{x_2, y_2\}_< \in t$ from n_2.



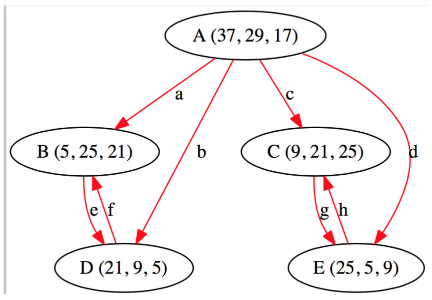


Figure 2 Rainbow diagram (left) and conflict digraph (right) for the toy example of initial structure s consisting of the six base pairs $(1, 13)$, $(5, 9)$, $(17, 29)$, $(21, 25)$, $(33, 41)$, $(49, 57)$ and of the target structure t consisting of the four base pairs $(5, 25)$, $(9, 21)$, $(29, 37)$, $(45, 53)$. The corresponding conflict digraph consists of 5 vertices, 8 directed edges, and 2 directed cycles. Edges are labeled for discussion in the text.

The motivation behind the definition of *edge* $\mathbf{v_1} \to \mathbf{v_2}$ is that either a base triple or pseudoknot will result if one applies the shift corresponding to $\mathbf{v_2}$ before the shift corresponding to $\mathbf{v_1}$. To that end, it is natural to define the edge $\mathbf{v_1} \to \mathbf{v_2}$ if the base pair $\mathbf{v_1}.s$ either touches or crosses the base pair $\mathbf{v_2}.t$. This approach is valid, but may lead to larger conflict digraphs with many more cycles than necessary, resulting in additional computational cost in the enumeration of all directed cycles as well as in application of the IP solver. Consider the example of initial structure $s = \{(5, 9)\}$ and target structure $t = \{(1, 5), (9, 13)\}$, for which an optimal 2-step MS_2 trajectory consists of shifting base pair $(5, 9)$ to $(1, 5)$, followed by adding the base pair $(9, 13)$. Vertices of the conflict digraph $G = (V, E)$ are clearly $\mathbf{v_1} = (1, 5, 9)$ and $\mathbf{v_2} = (13, 9, 5)$. Since $\mathbf{v_1}.s = (5, 9)$ touches $\mathbf{v_2}.t = (9, 13)$, we would have $\mathbf{v_1} \to \mathbf{v_2}$; since $\mathbf{v_2}.s = (5, 9)$ touches $\mathbf{v_1}.t = (1, 5)$, we would have $\mathbf{v_2} \to \mathbf{v_1}$. However, if we apply the shift move corresponding to $\mathbf{v_1}$, then we cannot subsequently apply the shift move corresponding to $\mathbf{v_2}$, since $\mathbf{v_1}.s = \mathbf{v_2}.s$. A similar issue arises when $\mathbf{v_1}.t = \mathbf{v_2}.t$.

For another example, consider the initial structure $s = \{(5, 9), (21, 25)\}$ and target structure $t = \{(5, 25), (9, 21)\}$. Vertices of the conflict digraph are B, C, D, E, where B is $(5, 25, 21)$, D is $(21, 9, 5)$, C is $(9, 21, 25)$, E is $(25, 5, 9)$. Supposing that $\mathbf{v_1} = (x_1, y_1, z_1)$ and $\mathbf{v_2} = (x_2, y_2, z_2)$, if $\mathbf{v_1} \to \mathbf{v_2}$ were defined by $\mathbf{v_1}.s$ touches or crosses $\mathbf{v_2}.t$, then the resulting conflict digraph would be the complete digraph on vertices B, C, D, E, leading to many more cycles than necessary. By defining $\mathbf{v_1} \to \mathbf{v_2}$ by either $z_1 = x_2$ or $\mathbf{v_1}.s$ crosses $\mathbf{v_2}.t$, the resulting conflict digraph consists of only $B \to C$, $C \to B$, $D \to E$, $E \to D$, which appears as a portion of Figure 2.

▶ **Definition 8** (Conflict digraph $G = (V, E)$)**.** Let s, t be distinct secondary structures for the RNA sequence $a_1, \ldots, a_n$. The RNA confict digraph $G(s, t) = (V(s, t), E(s, t))$, or $G = (V, E)$ when s, t are clear from context, is defined by

$$V = \{(x, y, z) : x, y, z \in [1, n] \wedge \{x, y\} \in t \wedge \{y, z\} \in s\}, \tag{8}$$

$$\begin{aligned} E = \Big\{ &(n_1, n_2) : n_1 = (x_1, y_1, z_1) \in V \wedge n_2 = (x_2, y_2, z_2) \in V \wedge \Big(z_1 = x_2 \vee \\ &\Big([\min(y_1, z_1) < \min(x_2, y_2) < \max(y_1, z_1) < \max(x_2, y_2)] \vee \\ &[\min(x_2, y_2) < \min(y_1, z_1) < \max(x_2, y_2) < \max(y_1, z_1)] \Big) \Big\}. \end{aligned} \tag{9}$$

Notice that Definition 8 establishes a partial ordering on vertices of the conflict digraph $G = (V, E)$, in that edges determine the order in which shift moves should be performed.

Indeed, if $n_1 = (x, y, z)$, $n_2 = (u, v, w)$ and $(n_1, n_2) \in E$, which we denote from now on by $n_1 \to n_2$, then the shift move in which $\{y, z\} \in s$ shifts to $\{x, y\} \in t$ must be performed before the shift move where $\{v, w\} \in s$ shifts to $\{u, v\} \in t$ – indeed, if shifts are performed in the opposite order, then after shifting $\{v, w\} \in s$ to $\{u, v\} \in t$ and before shifting $\{y, z\} \in s$ to $\{x, y\} \in t$, we would create either a base triple or a pseudoknot. Our strategy to efficiently compute the MS_2 distance between secondary structures s and t will be to (1) enumerate all simple cycles in the conflict digraph $G = (V, E)$ and to (2) apply an integer programming (IP) solver to solve the minimum feedback arc set problem $V' \subset V$. Noticing that the *induced digraph* $\overline{G} = (\overline{V}, \overline{E})$, where $\overline{V} = V - V'$ and $\overline{E} = E \cap (\overline{V} \times \overline{V})$, is acyclic, we then (3) topologically sort $\overline{G}$, and (4) perform shift moves from $\overline{V}$ in topologically sorted order. In an initial implementation of our algorithms, we used the `simple_cycles()` function from the `NetworkX` python library to enumerate all simple cycles.

There is a first important technical deviation from this strategy, corresponding to an additional IP constraint (‡) in line 7 of the pseudocode below, necessary to address a possible *overlap* between triplet nodes. It can happen, for instance, that base pair $(x, y) \in t$, and that base pairs $(u, x) \in s$ and $(y, z) \in s$, for which we have triplet nodes $v_1 = (y, x, u)$ and $v_2 = (x, y, z)$. If we detect node v_1 [resp. v_2] in a simple cycle C_1 [resp. C_2], then in the absence of (‡), the first IP constraint (†) would *remove* both nodes v_1 and v_2, whereby IP variables x_{v_1} and x_{v_2} would both be set to 0, resulting in the *removal of both* base pairs (u, x) and (y, z) from s in line 16 of the pseudocode. This causes an additional base pair *addition* of (x, y) to the folding pathway in line 18 of the pseudocode. In contrast, if (for instance) only the node v_1 had been removed, resulting in the base pair removal of (u, x) from s, then it would have been possible to shift base pair (y, z) to (x, y), rather than removing both (u, x) and (y, z) from s with subsequent base pair addition of (x, y). Such situations are avoided by the IP constraint (‡) below.

One might (incorrectly) surmise that it is possible to immediately remove the base pair $\{y, z\}$ from s for every node $v = (x, y, z) \notin \overline{V}$. The fallacy of doing this can be illustrated as follows. Suppose, for instance, that base pair $(x, y) \in s$, and that base pairs $(u, x) \in t$ and $(y, z) \in t$, for which we have triplet nodes $v_1 = (u, x, y)$ and $v_2 = (z, y, x)$. Since v_1, v_2 overlap in two positions, by the constraint (‡) in line 7 of the pseudocode below, it cannot be that both v_1 and v_2 both belong to $\overline{V}$. If neither v_1 nor v_2 belongs to $\overline{V}$, then it is safe to immediately remove base pair $\{x, y\}$ from s. However, if (say) $v_1 \in \overline{V}$ and $v_2 \notin \overline{V}$, then it would be a mistake to remove $\{x, y\}$ from s if we could instead later shift base pair $\{x, y\}$ to base pair $\{u, v\}$, *provided* that so doing does not create a base triple. Such a shift is possible if position u is not touched by s. A clean treatment of such situations requires the following definition.

▶ **Definition 9** (Base pair (x, y) is covered by $\overline{V}$)**.** Suppose that $G = (V, E)$ is the RNA conflict digraph for RNA sequence $a_1, \ldots, a_n$ and secondary structures s, t. Let $\overline{V} \subset V$. Base pair $(x, y) \in t$ is covered by $\overline{V}$ if there exists $v \in \overline{V}$ such that $v.t = (x, y)$, i.e. the t base pair portion of v equals (x, y). Base pair $(x, y) \in s$ is covered by $\overline{V}$ if there exists a base pair $(u, v) \in t$ such that (x, y) touches (u, v) and (u, v) is covered.

It will now follow that we can remove all base pairs $(x, y) \in s$ that are not covered by $\overline{V}$, as indicated in line 11 of the following pseudocode.

▶ **Algorithm 1** (MS_2 distance from s to t)**.**
INPUT: *Secondary structures s, t for RNA sequence $a_1, \ldots, a_n$*
OUTPUT: *Folding trajectory $s = s_0, s_1, \ldots, s_m = t$, where $s_0, \ldots, s_m$ are secondary structures, m is the minimum possible value for which s_i is obtained from s_{i-1} by a single base pair addition, removal or shift for each $i = 1, \ldots, m$.*

First, initialize the variable numMoves to 0, and the list moveSequence to the empty list [ ]. Recall that $BP_2 = \{(x,y) : (x,y) \in t, (s-t)[x] = 0, (s-t)[y] = 0\}$. Bear in mind that s is constantly being updated, so actions performed on s depend on its current value.

```
    //remove base pairs from $s$ that are untouched by $t$
 1.  $BP_1 = \{(x,y) : (x,y) \in s, (t-s)[x] = 0, (t-s)[y] = 0\}$
 2.  for $(x,y) \in BP_1$
 3.      remove $(x,y)$ from $s$; numMoves = numMoves+1
    //define conflict digraph $G = (V,E)$ on updated $s$ and unchanged $t$
 4.  define $V$ by equation (8)
 5.  define $E$ by equation (9)
 6.  define conflict digraph $G = (V,E)$
    //IP solution of minimum feedback arc set problem
 7.  maximize $\sum_{v \in V} x_v$ where $x_v \in \{0,1\}$, subject to constraints (†) and (‡)
    //first constraint removes vertex from each simple cycle of $G$
    (†) $\sum_{v \in C} x_v < |C|$ for each simple directed cycle $C$ of $G$
    //ensure shift moves cannot be applied if they share same base pair from $s$ or $t$
    (‡) $x_v + x_{v'} \leq 1$, for all pairs of vertices $v = (x,y,z)$ and $v' = (x',y',z')$
      with $|\{x,y,z\} \cap \{x',y',z'\}| = 2$
 8.  $\overline{V} = \{v \in V : x_v = 1\}$
 9.  $\overline{E} = \{(v,v') : v,v' \in \overline{V} \wedge (v,v') \in E\}$
10.  let $\overline{G} = (\overline{V},\overline{E})$
     //remove all base pairs of $s$ not covered by $\overline{V}$
11.  let $\overline{\texttt{Cover}} = \{(x,y) \in s : (x,y)$ is not covered by $\overline{V}\}$
12.  for $(x,y) \in \overline{\texttt{Cover}}$
13.      remove $(x,y)$ from $s$; numMoves = numMoves+1
    //topological sort of IP solution $\overline{V}$
14.  topological sort of $\overline{G}$ to determine total ordering $\prec$ on $\overline{V}$
15.  for $v = (x,y,z) \in \overline{V}$ in topologically sorted order $\prec$
16.      shift $\{y,z\}$ to $\{x,y\}$ in $s$; numMoves = numMoves+1
        //add remaining base pairs from $t-s$
17.  for $(x,y) \in t-s$
18.      add $(x,y)$ to $s$; numMoves = numMoves+1
19.  return folding trajectory, numMoves
```

Toy example

Consider the following toy 48 nt example of initial structure s consisting of the six base pairs $(1,13)$, $(5,9)$, $(17,29)$, $(21,25)$, $(33,41)$, $(49,57)$ and the target structure t consisting of the four base pairs $(5,25)$, $(9,21)$, $(29,37)$, $(45,53)$ with dot-bracket structures given by

```
>s
123456789012345678901234567890123456789012345678901234567 8
GAAAGAAAUAAACAAAGAAAGAAACAAACAAAGAAAGAAACAAAGAAAGAAACAAACA
(...(...)...)...(...(...)...)...(.......).......(.......).
>t
1234567890123456789012345678901234567890123456789012345678
GAAAGAAAUAAACAAAGAAAGAAACAAACAAAGAAAGAAACAAAGAAAGAAACAAACA
....(...(...........)...)...(.......).......(.......).....
```

Figure 2a depicts the "rainbow" diagram of initial structure s shown below the line in red, and of target structure t shown above the line in blue. If $G = (V,E)$ denotes the corresponding conflict digraph of s,t, as depicted in Figure 2b, then the vertices $\mathbf{v} = (x,y,z)$

belonging to V are exactly those triples (x,y,z) such that blue arc $\mathbf{v}.t = (x,y)_<$ touches red arc $\mathbf{v}.s = (y,z)_<$. Moreover, for vertices $\mathbf{v}_1 = (x_1,y_1,z_1)$ and $\mathbf{v}_2 = (x_2,y_2,z_2)$, there is a directed edge $\mathbf{u} \to \mathbf{v}$ exactly when either (1) $z_1 = x_2$ or (2) $\mathbf{v}_1.s$ crosses $\mathbf{v}_2.t$. For the current example, it is straightforward for the user to derive the conflict digraph from the rainbow diagram, and to confirm that the conflict digraph $G = (V,E)$ consists of 5 vertices, 8 directed edges, and 2 directed cycles, as shown in Figure 2b.

A solution feedback vector set (FVS) problem is given by $\overline{V} = \{A,B,C\} = \{(37,29,17), (5,25,21), (9,21,25)\}$, since $\overline{V}$ is a maximum size subset of V such that the induced digraph $\overline{G} = (\overline{V},\overline{E})$ is acyclic, where edge set $\overline{E} = E \cap (\overline{V} \times \overline{V}) = \{a,c\}$ – in more intuitive terms, $\overline{G}$ is obtained by deleting the bottom two nodes $(21,9,5)$, $(25,5,9)$ of Figure 2b, and deleting all edges b,e,f,d,g,h incident to these two nodes. In contrast, a solution feedback arc set (FAS) problem is given by $\overline{E} = E - \{f,h\} = \{a,b,c,d,e,g\}$ obtained by deleting edges f,h from the conflict digraph $G = (V,E)$ of Figure 2b. The resulting digraph $\overline{G} = (V, E - \{f,g\})$ is acyclic.

We now trace Algorithm 1. The set BP_1 from equation (5) consists of the base pairs $(1,13)$, $(33,41)$, $(49,57)$ in s that do not touch any base pair of t, and hence cannot be removed by applying a shift move. Lines 1–3 of Algorithm 1 result in updating the value of the variable s by removing these three base pairs. Lines 4–6 construct the conflict digraph $G = (V,E)$ shown in Figure 2. Line 7 invokes the IP solver to maximize the number of vertices of V subject to constraints (†) and (‡). The maximum size subset of V that satisfies (†) is simply a solution of VAS. Constraint (‡) additionally requires that $|\mathit{flatten}(\mathbf{v}_1 \cap \mathit{flatten}(\mathbf{v}_2| \leq 1$ for all $\mathbf{v}_1, \mathbf{v}_2 \in \overline{V}$. Observe that from the original vertex set V in the conflict digraph of Figure 2b, $|\mathit{flatten}(B) \cap \mathit{flatten}(C)| = |\{21,25\}| = 2$, $|\mathit{flatten}(B) \cap \mathit{flatten}(D)| = |\{5,21\}| = 2$, $|\mathit{flatten}(B) \cap \mathit{flatten}(E)| = |\{5,25\}| = 2$, $|\mathit{flatten}(C) \cap \mathit{flatten}(D)| = |\{9,21\}| = 2$, $|\mathit{flatten}(C) \cap \mathit{flatten}(E)| = |\{9,25\}| = 2$. Thus, although $\{A,B,C\}$ is a solution of VAS, it is <u>not</u> a solution of both constraints (†) and (‡) – a maximum size set $\overline{V} \subseteq V$ that satisfies both (†) and (‡) is $\overline{V} = \{A,B\}$.

Line 11 of Algorithm 1 computes $\overline{\text{Cover}} = \{(5,9)\}$, i.e. base pair $(5,9)$ is the only <u>uncovered</u> base pair of s, which is removed from s by lines 12,13. Line 14 applies topological sorting to establish the following total ordering of vertices in $\overline{V}$: (1) vertex A or $(37,29,17)$, (2) vertex B or $(5,25,21)$. Lines 15–16 result in the shift $(17,29)$ to $(29,37)$, followed by the shift $(21,25)$ to $(5,25)$. Lines 17–18 add any remaining base pairs of $t - s$ to s, resulting in adding base pairs $(9,21)$ and $(45,53)$. This yields an 8-step MS_2 trajectory consisting of 4 base pair removals, 2 shifts and 2 base pair additions (not shown due to space constraints).

```
     GAAAGAAAUAAACAAAGAAAGAAACAAACAAAGAAAGAAACAAAGAAAGAAACAAACA
     1234567890123456789012345678901234567890123456789012345678

0.   (...(...)...)...(...(...)...)...(.......).......(.......).   initial structure
1.   ................(...(...)...)...(.......).......(.......).   remove (1,13)
2.   ................(...(...)...)...................(.......).   remove (33,41)
3.   ................(...(...)...).............................   remove (49,57)
4.   (...........)...(...(...)...)...(.......).......(.......).   remove (5,9)
5.   ....................(...)...(.......).....................   shift (17,29) -> (29,37)
6.   ....(...................)...(.......).....................   shift (21,25) -> (5,25)
7.   ....(...(...........)...)...(.......).....................   add (9,21)
8.   ....(...(...........)...)...(.......).......(.......).....   add (45,53)
```

Bistable switch

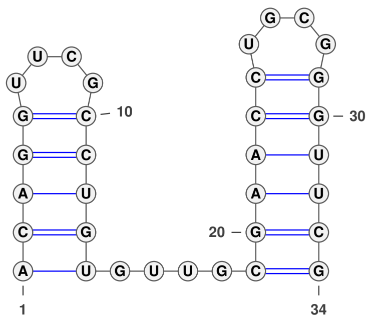
As nontrivial example, consider the 34 nt bistable switch with RNA sequence ACAGGUUCGC CUGUGUUGCG AACCUGCGGG UUCG taken from Figure 1(b).2 of [8], in which the authors performed structural probing by comparative imino proton NMR spectroscopy. Figures 3a, 3b, and 3c respectively depict the metastable secondary structure s having free energy -14.00 kcal/mol, the minimum free energy (MFE) secondary structure t having free energy of -14.70 kcal/mol, and the MFE conflict digraph. In the MFE conflict digraph $G = (V, E)$, vertices are triplet nodes (x, y, z), where (unordered) base pair $\{y, z\} \in s$ belongs to the metastable structure, and (unordered) base pair $\{x, y\} \in t$ belongs to the MFE structure. A direct edge $(x, y, z) \rightarrow (u, v, w)$ occurs if $\{y, z\} \in s$ touches or crosses $\{u, v\} \in t$. The conflict digraph $G = (V, E)$ for this bistable switch contains 11 vertices, 71 directed edges, and 92,114 directed cycles. The MS_2 distance is 13, consisting of 4 base pair removals, 7 shifts and 2 base pair additions. The the corresponding minimum length trajectory follows. This optimal MS_2 trajectory contains 4 base pair removals, 2 base pair additions, and 7 base pair shifts.

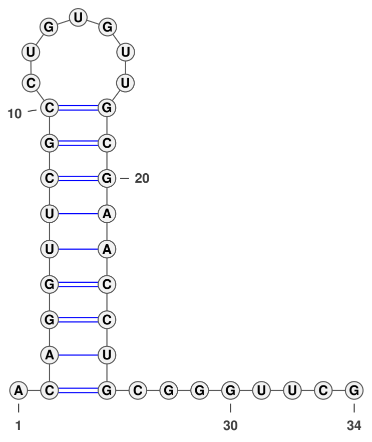
```
    ACAGGUUCGCCUGUGUUGCGAACCUGCGGGUUCG
    1234567890123456789012345678901234

 0.  (((((....)))))....((((((....))))))    initial structure
 1.  .((((....)))).....((((((....))))))    remove (1,14)
 2.  .((.(....).)).....((((((....))))))    remove (4,11)
 3.  ..(.(....).)......((((((....))))))    remove (2,13)
 4.  ....(....)........((((((....))))))    remove (3,12)
 5.  .........(.......)((((((....))))))    shift (5,10)  -> (10,18)
 6.  ........((.......))(((((....))))).    shift (19,34) -> (9,19)
 7.  .......(((.......)))((((....))))..    shift (20,33) -> (8,20)
 8.  ......((((.......))))(((....)))...    shift (21,32) -> (7,21)
 9.  .....(((((.......)))))((....))....    shift (22,31) -> (6,22)
10.  ....((((((.......))))))(....).....    shift (23,30) -> (5,23)
11.  ...(((((((.......)))))))..........    shift (24,29) -> (4,24)
12.  .(.(((((((.......))))))).)........    add (2,26)
13.  .(((((((((.......)))))))))........    add (3,25)
```

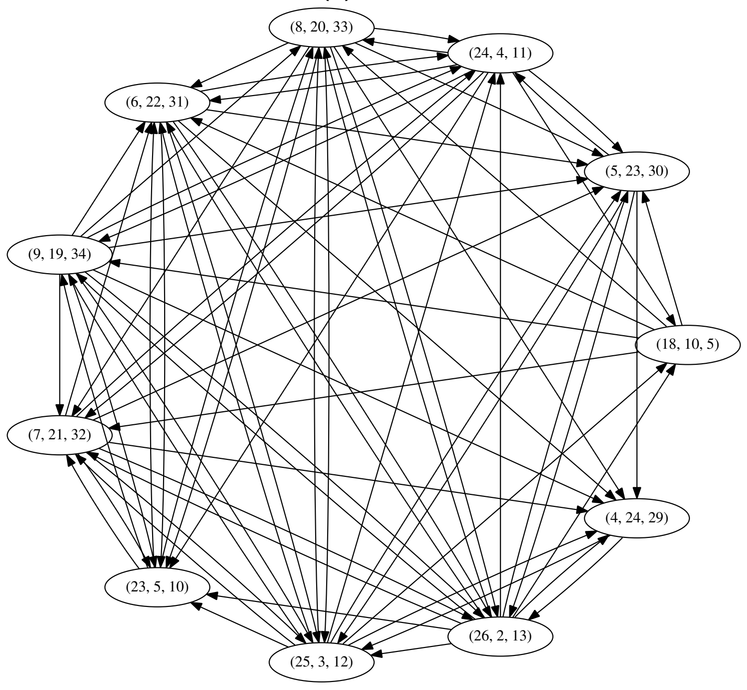
Details concerning our fast, near-optimal algorithm will be presented elsewhere; however, since Figures 4 and 5 compare the performance of the exact IP (optimal) algorithm with that of the near-optimal algorithm, we briefly sketch the idea behind the method.

▶ **Algorithm 2** (Near-optimal MS_2 distance from s to t).
INPUT: *Secondary structures s, t for RNA sequence $a_1, \ldots, a_n$*
OUTPUT: *Folding trajectory $s = s_0, s_1, \ldots, s_m = t$, where $s_0, \ldots, s_m$ are secondary structures, for each $i = 1, \ldots, m$, s_i is obtained from s_{i-1} by a single base pair addition, removal or shift, and m is an approximation to MS_2 distance between s and t.*

The idea is to generate all equivalence classes $[x]$ with respect to equivalence relation $\equiv$, defined to be the reflexive, transitive closure of $\sim$, where for $x, y \in [1, n]$, we say $x \sim y$ if $\{x, y\} \in s$ or $\{x, y\} \in t$. We consider a *coarse-grain* digraph, whose vertices are the equivalence classes $[x]$, and whose directed edges $[x] \rightarrow [y]$ are defined if a base pair from s that lies in $[x]$ crosses a base pair from t that lies in $[y]$. Solve the *feedback arc* problem (not feedback vertex problem) for this coarse-grained digraph using IP, where we note that the number of cycles is dramatically smaller than that for the exact IP algorithm. Apply topological sorting on the coarse-grained acyclic digraph after removal of feedback arcs. Subsequently process each equivalence class by using the exact IP algorithm. Due to space


(a) Metastable structure −14.00 kcal/mol


(b) MFE, −14.70 kcal/mol



(c) RNA conflict digraph, where s is metastable, t is MFE structure

Figure 3 Conflict digraph for toy example (a) and for the 34 nt bistable switch (b,c,d) with sequence ACAGGUUCGC CUGUGUUGCG AACCUGCGGG UUCG taken from Figure 1(b).2 of [8], in which the authors performed structural probing by comparative imino proton NMR spectroscopy. (a) Toy example used in a first example of Algorithm 1. (b) Metastable structure having next lowest free free energy (after that of minimum free energy structure) of −14.00 kcal/mol. (c) Minimum free energy (MFE) structure having −14.70 kcal/mol. (d) RNA conflict digraph $G = (V, E)$, having directed edges $(x, y, z) \to (u, v, w)$ if the (unordered) base pair $\{y, z\} \in s$ touches or crosses the (unordered) base pair $\{u, v\} \in t$. Here, s is in the metastable structure shown in (a) having −14.00 kcal/mol, while t is the MFE structure shown in (b) having −14.70 kcal/mol. The conflict digraph represents a necessary order of application of shift moves, in order to avoid the creation of base triples or pseudoknots in the optimal trajetory being constructed. The conflict digraph G has 11 vertices, 71 directed edges and 92,114 directed cycles.

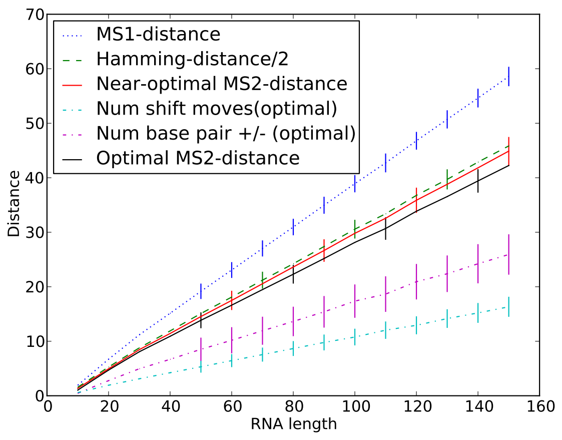


Figure 4 Benchmarking statistics for optimal and near-optimal algorithm to compute minimum length MS_2 folding trajectories between random secondary structures s, t of random RNA sequences of variable lengths. For each sequence length $n = 10, 15, 20, \cdots, 150$ nt, twenty random RNA sequences were generated of length n, with probability of $1/4$ for each nucleotide. For each RNA sequence, twenty secondary structures s, t were uniformly randomly generated so that 40% of the nucleotides are base-paired. It follows that the benchmarking dataset consisted of $20 \cdot \binom{20}{2} = 3800$ many triples $\mathbf{a}, s, t$, where $\mathbf{a} = a_1, \ldots, a_n$ denotes an RNA sequence of length n, and s, t are random secondary structures of $\mathbf{a}$. Using this dataset, consisting of $20 \cdot \binom{20}{2} = 3800$ many triples $\mathbf{a}, s, t$, where $\mathbf{a} = a_1, \ldots, a_n$ denotes an RNA sequence of length n, and s, t are random secondary structures of $\mathbf{a}$, the average MS_2 distance was computed for both the exact IP Algorithm 1 and the near-optimal algorithm, whose details cannot be described due to space constraints. In addition to MS_2 distance computed by the exact IP and the near-optimal algorithm, the figure displays MS_1 distance (also known as base pair distance), Hamming distance over 2, and provides a breakdown of the MS_1 distance in terms of the number of base pair addition/removal moves "num base pair +/- (optimal)" and the shift moves "num shift moves (optimal)".

constraints, we cannot provide additional details for the near-optimal algorithm, which will be described elsewhere.

3 Discussion and an application

Computational approaches to the problem of RNA secondary structure folding kinetics involve one of three approaches: (1) computation of energy-optimal folding pathways [13, 5, 4], (2) solution of the master equation [11] to determine the time necessary to reach equilibrium [19, 16], (3) repeated simulations using the Gillespie algorithm [6] as in the software `Kinfold` [5] and `KINEFOLD` [20].

An *energy-optimal* folding pathway is a sequence $s = s_0, s_1, \ldots, s_m = t$ of secondary structures from initial structure s to target structure t, such that each intermediate structure s_i is obtained from its predecessor s_{i-1} by application of a move from a specified move set, and such that the *maximum* energy difference $E(s,t) = \max_{i=1,\ldots,m} (E(s_i) - E(s_0))$ between an intermediate structure and the initial structure is the *minimum* possible value, when taken over all possible folding trajectories – this energy difference $E(s,t)$ is called the *barrier energy*. Intuitively, an energy-optimal folding trajectory is analogous to an alpine walk between two points A and B, for which the walker reaches the minimum possible intermediate altitude, and the barrier energy is analogous to the difference between the altitude at the

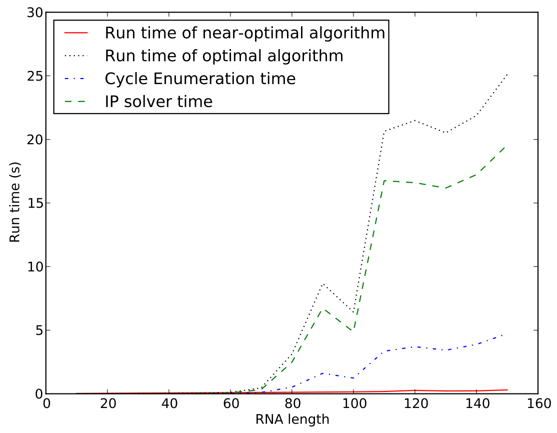


Figure 5 Run time for the exact IP (optimal) algorithm 1 and the near-optimal algorithm 2 to compute minimum length MS_2 folding trajectories for the same data set from previous Figure 4. Each data point represents the average $\mu \pm \sigma$ where error bars indicate one standard deviation, taken over 3800 sequence, structure pairs. Run time of the optimal algorithm depends almost entirely on the time to enumerate all directed cycles, using our C++ implementation of Johnson's algorithm [9] as well as time for the Gurobi IP solver.

mountain pass and that at ground camp A. The problem of computing the barrier energy is NP-complete, even for the trivial energy model of -1 per base pair [17]. After calling the program `RNAsubopt -e E` to generate all secondary structures, whose free energy is within E kcal/mol of the minimum free energy (MFE), the program `barriers` [5] uses a "flooding" procedure to determine an energy-optimal folding trajectory (run time of `RNAsubopt -e E` is exponential in the user-input energy parameter E). Note that `barriers` allows move sets MS_1 and MS_2, but that both the initial and target structure must be *locally optimal*, where a locally optimal structure has the property that no structure obained by applying one move from the move set yields a structure with strictly lower energy. The structures s, t for the previous toy example RNA are not locally optimal, in contrast to the structures s, t for the the previous bistable switch.

The program `RNAtabupath` [4] is a local search method using the *tabu* heuristic [7] which provides a very fast, near-optimal solution for the barrier energy and energy-optimal folding trajectory. Note that `RNAtabupath` does not require that initial and target structures be locally optimal, but at present only computes near-optimal MS_1 trajectories.de Another application of `RNAtabupath` is that the true MS_1 barrier energy is bounded above by the `RNAtabupath` barrier energy estimate, and hence can be used as energy parameter for `barriers`; i.e. ethe user need not use trial-and-error when entering an energy parameter that exceeds the barrier energy in order to generate an energy-optimal folding trajectory – a very time-consuming, manual procedure. Analogously, one can use Algorithm 1 to provide an energy parameter that exceeds the MS_2 barrier energy in order to generate an energy-optimal MS_2 folding trajectory using `barriers`.

Figure 6 shows the free energy profile of the shortest MS_2 folding trajectory returned by Algorithm 1 for the 34 nt bistable switch with sequence ACAGGUUCGC CUGUGUUGCG AACCUGCGGG UUCG, which sequence comes from Figure 1(b).2 of [8]. The barrier energy for the shortest MS_2 trajectory computed by Algorithm 1 is 10.8 kcal/mol with trajectory length 13. The figure also shows the nearly energy-optimal folding MS_2 trajectory

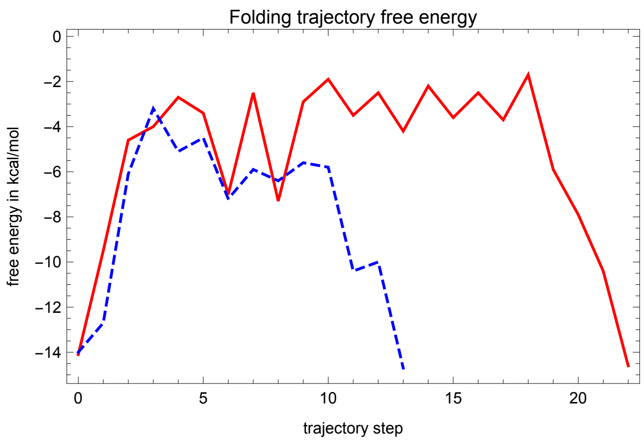


Figure 6 Free energy in kcal/mol of secondary structures appearing in RNA folding trajectories of the bistable switch, whose sequence is given in Figure 1(b).2 of [8]. The dashed blue line corresponds to the minimum length MS_2 trajectory computed by Algorithm 1; the solid red line corresponds to the lowest energy MS_1 folding trajectory found by the program `tabuPath` described in [4] in 100 folding attempts. The barrier energy for MS_1 trajectories estimated by the near-optimal software `tabuPath` (without shift) is 12.4 kcal/mol with trajectory length 23. The barrier energy for the shortest MS_2 trajectory computed by Algorithm 1 is 10.8 kcal/mol with trajectory length 13.

of `RNAtabupath`. Figure 7 shows the Arrhenius tree returned by `barriers` when run with energy parameter $10.8 + (14.7 - 14.0) = 11.5$ kcal/mol, where the value 10.8 is the energy barrier for the trajectory returned by Algorithm 1, -14.7 [resp. -14.0] is the free energy of initial [resp. target] structure s [resp. t]. Since t is the minimum free energy (MFE) structure, the program `RNAsubopt -e 11.5` will generate close to the smallest set of structures which guarantee that the program `barriers` can find an energy-optimal folding trajectory from s to t. Figures 6 and 7 consider the small 34 nt bistable switch sequence, for display purposes; clearly this approach becomes much more practical for long RNA sequences when using the near-optimal Algorithm 2.

4 Conclusion and discussion

In this paper, we have presented the first algorithms to compute the MS_2 distance between any two secondary structures s, t of a given RNA sequence. Despite the impressive speed and (approximate) accuracy of our near-optimal algorithm 2, we conjecture that the problem of computing a minimum-length MS_2 trajectory is NP-hard. This is due to several reasons: (1) the complexity of the exact IP algorithm, (2) the dramatic increase in the number of simple cycles in RNA conflict digraphs, as sequence length increases (not shown), (3) the dramatic increase in run time required by the Gurobi IP solver for sequences of increasing length (not shown), (4) the fact that FVS and FAS are NP-complete problems for several known families of digraphs. Initial investigations (omitted here) have shown the that family of RNA conflict digraphs is distinct from a host of graph families, for which the computational complexity of FVS/FAS is known; however, at present it is unclear whether there is a polynomial time algorithm to determine whether a given digraph is representable as an RNA conflict digraph.

The complexity of MS_2 distance suggests that simple simulation studies of RNA structural evolution and robustness [18] are unlikely to be extended to consider shift moves, despite the experimental evidence for particular shift moves such as helix zippering and defect

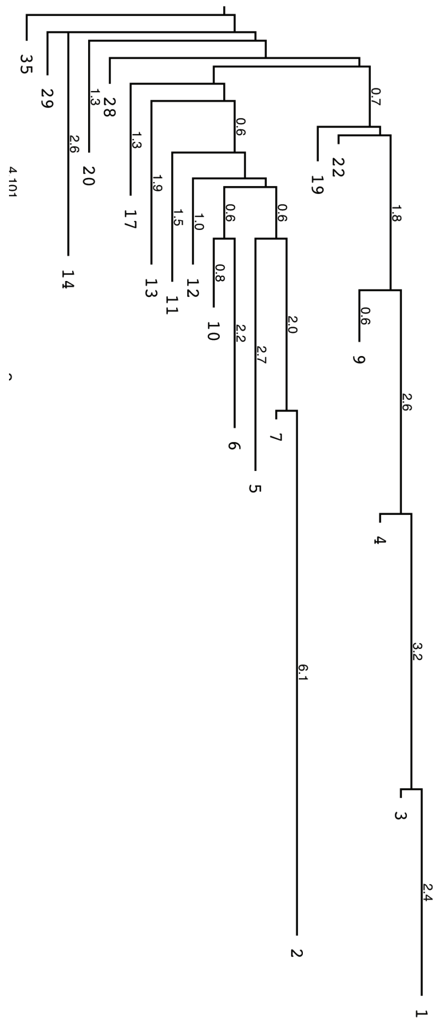


Figure 7 Arrhenius tree produced by running Vienna RNA Package programs `RNAsubopt -s -e 11.5` and `barriers` to obtain an optimal folding pathway. The energy bound of 11.5 kcal/mol was selected, because the free energy of initial structure s [resp. target structure t] is -14.0 [resp. -14.6] kcal/mol, and the barrier energy of the shortest MS_2 trajectory from the left panel of this figure is 10.8 kcal/mol. It follows that we know there exists a folding trajectory within $10.8 + (14.7 - 14.0) = 11.5$ kcal/mol of the MFE structure t. The advantage of first running Algorithm 1 is that knowledge of the energy barrier for the shortest MS_2 path allows an efficient computation of `RNAsubopt` and `barriers` – in the current case, `RNAsubopt` only needed to generate 1556 structures, and to find 28 saddle structures. The number of structures for this 34 nt bistable switch is 845,139,060,165 $\approx 8.45 \cdot 10^{11}$.

diffusion [14]. Moreover, studies of RNA structural evolution from [15] used Hamming distance as a simple approximation to MS_2 distance, which we now know from Figure 4 not to be particularly accurate. Finally, some very interesting, yet complex questions are raised concerning graph theory (which digraphs are representable as conflict digraphs), computational complexity (whether MS_2 distance is NP-hard), and potentially related group theoretic questions.

References

1. P. Clote. Expected degree for RNA secondary structure networks. *J. Comput. Chem.*, 0(O):O, November 2014.
2. P. Clote and A. Bayegan. Network Properties of the Ensemble of RNA Structures. *PLoS One*, 10(10):e0139476, 2015.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Algorithms.* McGraw-Hill, 1990. 1028 pages.
4. I. Dotu, W. A. Lorenz, P. VAN Hentenryck, and P. Clote. Computing folding pathways between RNA secondary structures. *Nucleic. Acids. Res.*, 38(5):1711–1722, 2010.
5. C. Flamm, I. L. Hofacker, P. F. Stadler, and M. Wolfinger. Barrier trees of degenerate landscapes. *Z. Phys. Chem.*, 216:155–173, 2002.
6. D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comp. Phys.*, 22(403):403–434, 1976.
7. F. W. Glover and M. Laguna. *Tabu Search.* Springer-Verlag, 1998. 408 p.
8. C. Hobartner and R. Micura. Bistable secondary structures of small RNAs and their structural probing by comparative imino proton NMR spectroscopy. *J. Mol. Biol.*, 325(3):421–431, January 2003.
9. D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4:77–84, 1975.
10. Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*, pages 85–103, 1972. URL: `http://www.cs.berkeley.edu/~luca/cs172/karp.pdf`.
11. A. Kolmogoroff. Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung. *Mathematische Annalen*, 104:415–458, 1931.
12. R. Lorenz, S. H. Bernhart, C. Höner zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker. Viennarna Package 2.0. *Algorithms. Mol. Biol.*, 6:26, 2011.
13. S. R. Morgan and P. G. Higgs. Barrier heights between ground states in a model of RNA secondary structure. *J. Phys. A: Math. Gen.*, 31:3153–3170, 1998.
14. D. Pörschke. Model calculations on the kinetics of oligonucleotide double-helix coil transitions: Evidence for a fast chain sliding reaction. *Biophys Chem*, 2(2):83–96, August 1974.
15. P. Schuster and P. F. Stadler. Modeling conformational flexibility and evolution of structure: RNA as an example. In U. Bastille, M. Roman, and M. Vendruscolo, editors, *Structural Approaches to Sequence-Evolution*, page 3–36. Springer, Heidelberg, 2007.
16. X. Tang, B. Kirkpatrick, S. Thomas, G. Song, and N. M. Amato. Using motion planning to study RNA folding kinetics. *J. Comput. Biol.*, 12(6):862–881, July/August 2005.
17. C. Thachuk, J. Maňuch, L. Stacho, and A. Condon. NP-completeness of the direct energy barrier height problem. *Natural Computing*, 10(1):391–405, 2011.
18. A. Wagner. Robustness and evolvability: a paradox resolved. *Proc. Biol Sci.*, 275(1630):91–100, January 2008.

19 Michael T. Wolfinger, W. Andreas Svrcek-Seiler, Christoph Flamm, Ivo L. Hofacker, and Peter F. Stadler. Efficient folding dynamics of RNA secondary structures. *J. Phys. A: Math. Gen.*, 37:4731–4741, 2004.

20 A. Xayaphoummine, T. Bucher, and H. Isambert. Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. *Nucleic. Acids. Res.*, 33(Web):W605–W610, July 2005.

Fast Spaced Seed Hashing*

Samuele Girotto[1], Matteo Comin[2], and Cinzia Pizzi[3]

1 **Department of Information Engineering, University of Padova, Padova, Italy**
2 **Department of Information Engineering, University of Padova, Padova, Italy**
`comin@dei.unipd.it`
3 **Department of Information Engineering, University of Padova, Padova, Italy**
`cinzia.pizzi@dei.unipd.it`

Abstract

Hashing k-mers is a common function across many bioinformatics applications and it is widely used for indexing, querying and rapid similarity search. Recently, spaced seeds, a special type of pattern that accounts for errors or mutations, are routinely used instead of k-mers. Spaced seeds allow to improve the sensitivity, with respect to k-mers, in many applications, however the hashing of spaced seeds increases substantially the computational time. Hence, the ability to speed up hashing operations of spaced seeds would have a major impact in the field, making spaced seed applications not only accurate, but also faster and more efficient.

In this paper we address the problem of efficient spaced seed hashing. The proposed algorithm exploits the similarity of adjacent spaced seed hash values in an input sequence in order to efficiently compute the next hash. We report a series of experiments on NGS reads hashing using several spaced seeds. In the experiments, our algorithm can compute the hashing values of spaced seeds with a speedup, with respect to the traditional approach, between 1.6x to 5.3x, depending on the structure of the spaced seed.

1998 ACM Subject Classification I.1.2 Algorithms

Keywords and phrases k-mers, spaced seeds, efficient hashing

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.7

1 Introduction

The most frequently used tools in bioinformatics are those searching for similarities, or local alignments, between biological sequences. k-mers, i.e. words of length k, are at the basis of many sequence comparison methods, among which the most widely used and notable example is BLAST [1].

BLAST uses the so-called "hit and extend" method, where a hit consists of a match of a 11-mers between two sequences. Then these matches are potential candidates to be extended and to form a local alignment. It can be easily noticed that not all local alignments include an identical stretch of length 11. As observed in [3] allowing for not consecutive matches increases the chances of finding alignments. The idea of optimizing the choice of the positions for the required matches, in order to design the so called *spaced seeds*, has been investigated in many studies, and it was used in PatternHunter [16], another popular similarity search software.

In general contiguous k-mers counts are a fundamental step in many bioinformatic applications [5, 6, 9, 20, 22, 21, 24]. However, spaced seeds are now routinely used, instead of

* This work was supported by the MIUR PRIN project n. 20122F87B2 "Compositional approaches for the characterization and mining of omics data".

contiguous k-mers, in many problems involving sequence comparison like: multiple sequence alignment [7], protein classification [18], read mapping [23] and for alignment-free phylogeny reconstruction [13]. More recently, it was shown that also metagenome reads clustering and classification can benefit from the use of spaced seeds [4, 8, 19].

A spaced seed of length k and weight $w < k$ is a string over the alphabet $\{1, 0\}$ that contains w '1' and $(k - w)$ '0' symbols. A spaced seed is a mask where the symbols '1' and '0' denote respectively match and don't care positions. The design of spaced seeds is a challenging problem itself, tackled by several studies in the literature [10, 11, 16]. Ideally, one would like to maximize the sensitivity of the spaced seeds, which is however an NP-hard problem [15].

The advantage of using spaced seeds, rather than contiguous k-mers, in biological sequence analysis, comes from the ability of such pattern model to account for mutations, allowing for some mismatches in predefined positions. Moreover, from the statistical point of view, the occurrences of spaced seeds at neighboring sequence positions are statistically less dependent than occurrences of contiguous k-mers [15]. Much work has been dedicated to spaced seeds over the years, we refer the reader to [2] for a survey on the earlier work.

Large-scale sequence analysis often relies on cataloging or counting consecutive k-mers in DNA sequences for indexing, querying and similarity searching. An efficient way of implementing such operations is through the use of hash based data structures, e.g. hash tables. In the case of contiguous k-mers this operation is fairly simple because the hashing value can be computed by extending the hash computed at the previous position, since they share $k - 1$ symbols [17]. For this reason, indexing all contiguous k-mers in a string can be a very efficient process.

However, when using spaced seeds these observations do not longer hold. As a consequence, the use of spaced seeds within a string comparison method generally produces a slow down with respect to the analogous computation performed using contiguous k-mers. Therefore, improving the performance of spaced seed hashing algorithms would have a great impact on a wide range of bioinformatics tools.

For example, from a recent experimental comparison among several metagenomic read classifiers [14], Clark [20] emerged as one of the best performing tools for such a task. Clark is based on discriminative contiguous k-mers, and it is capable of classifying about 3.5M reads per minute. When contiguous k-mers are replaced by spaced seeds, as in Clark-S [19], while the quality of the classification improves, the classification rate is reduced to just 200K reads per minute.

The authors of Clark-S attributed such a difference to the use of spaced seeds. In particular, the possible sources of slowdown are two: the hashing of spaced seeds, and the use of multiple spaced seeds. In fact, Clark-S uses three different spaced seeds simultaneously in its processing. However, while the number of spaced seeds used could explain a 3x slowdown, running Clark-S is 17x slower than the original k-mer based Clark. Thus, the main cause of loss of speed performances can be ascribe to the use of spaced seed instead of contiguous k-mers. A similar reduction in time performance when using spaced seeds is reported also in other studies [4, 18, 23]. We believe that the main cause is the fact that spaced seeds can not be efficiently hashed, as opposed to contiguous k-mers.

In this paper we address the problem of the computation of spaced seed hashing for all the positions in an given input sequence, and present an algorithm that is faster than the standard approach to solve this problem. Moreover, since using multiple spaced seeds simultaneously on the same input string can increase the sensitivity [13], we also developed a variant of our algorithm for simultaneous hashing of multiple spaced seeds.

In general, when computing a hash function there are also other properties of the resulting hash that might be of interest like: bit dependencies, hash distributions, collisions etc. However, the main focus of this paper is the fast computation of spaced seed hashing, using the most simple hash function. Note that our method can be extended to implement, for example, the cyclic polynomial hash used in [17] with no extra costs.

In the next section we briefly summarize the properties of spaced seeds and describe our algorithm, together with a variant for handling multiple seed hashing. Experimental results on NGS reads hashing for various spaced seeds are reported in Section 3. Conclusions are driven in Section 4.

2 Methods

A *spaced-seed* S (or just a seed) is a string over the alphabet $\{1, 0\}$ where the 1s correspond to matching positions. The *weight* of a seed corresponds to the number of 1s, while the overall *length*, or span, is the sum of the number of 0s and 1s.

Another way to denote a spaced seed is through the notation introduced in [12]. A spaced seed can be represented by its *shape* Q that is the set of non negative integers corresponding to the positions of the 1s in the seed. A seed can be described by its shape Q where its weight W is denoted as $|Q|$, and its span $s(Q)$ is equal to $\max Q + 1$. For any integer i and shape Q, the positioned shape $i + Q$ is defined as the set $\{i + k, k \in Q\}$. Let us consider the positioned shape $i + Q = \{i_0, i_1, \ldots, i_{W-1}\}$, where $i = i_0 < i_1 < \ldots < i_{W-1}$, and let $x = x_0x_1 \ldots x_{n-1}$ be a string over the alphabet $\mathcal{A}$. For any position i in the string x, with $0 \le i \le n-s(Q)$, the positioned spaced seed $i + Q$ identifies a string of length $|Q|$ that we call Q-gram. A Q-gram at position i in x is the string $x_{i_0}x_{i_1} \ldots x_{i_{W-1}}$ and it is denoted by $x[i + Q]$.

▶ **Example 1.** Let $Q = \{0, 2, 3, 4, 6, 7\}$, then Q is the seed 10111011, its weight is $|Q| = 6$ and its span is $s(Q) = 8$. Let us consider the string $x = ACTGACTGGA$, then the Q-gram $x[0 + Q] = ATGATG$ can be defined as:

x	A	C	T	G	A	C	T	G	G	A
Q	1	0	1	1	1	0	1	1		
$x[0+Q]$	A		T	G	A		T	G		

Similarly all other Q-grams are $x[1 + Q] = CGACGG$, and $x[2 + Q] = TACTGA$.

2.1 Spaced Seed Hashing

In order to hash any string, first we need to have a coding function from the alphabet $\mathcal{A}$ to a binary codeword. For example let us consider the function $encode : \mathcal{A} \to \{0, 1\}^{log_2|\mathcal{A}|}$, with the following values $encode(A) = 00, encode(C) = 01, encode(G) = 10, encode(T) = 11$. Based on this function we can compute the encodings of all symbols of the Q-gram $x[0 + Q]$ as follows:

$x[0+Q]$	A	T	G	A	T	G
encodings	00	11	10	00	11	10

There exist several hashing functions, in this paper we consider the Rabin-Karp rolling hash, defined as $h(x[0 + Q]) = encode(A) * |\mathcal{A}|^0 + encode(T) * |\mathcal{A}|^1 + encode(G) * |\mathcal{A}|^2 + encode(A) * |\mathcal{A}|^3 + encode(T) * |\mathcal{A}|^4 + encode(G) * |\mathcal{A}|^5$. In the original Rabin-Karp rolling hash all math is done in modulo n, here for simplicity we avoid that. In the case of DNA sequences $|\mathcal{A}| = 4$, that is a power of 2 and thus the multiplications can be implemented with a shift. In the above example, the hashing value associated to the Q-gram $ATGATG$ simply corresponds to the list of encoding in Little-endian: 101100101100.

To compute the hashing value of a Q-gram from its encodings one can define the function $h(x[i + Q])$, for any given position i of the string x, as:

$$h(x[i+Q]) = \bigvee_{k \in Q} (encode(x_{i+k}) \ll m(k) * log_2|\mathcal{A}|)\,, \tag{1}$$

where $m(k)$ is the number of shifts to be applied to the encoding of the k-th symbols. For a spaced seed Q the function m is defined as $m(k) = |\{i \in Q,\ \text{such that } i < k\}|$. In other words, given a position k in the seed, m stores the number of matching positions that appear to the left of k. The vector m is important for the computation of the hashing value of a Q-gram.

▶ **Example 2.** In the following we report an example of hashing value computation for the Q-gram $x[0 + Q]$.

x	A	C	T	G	A	C	T	G	G	A
Q	1	0	1	1	1	0	1	1		
m	0	1	1	2	3	4	4	5		
shifted encodings	00		11«2	10«4	00«6		11«8	10«10		
			1100							
				101100						
					00101100					
							1100101100			
hashing value								101100101100		

The hashing values for the others Q-grams can be determined through the function $h(x[i + Q])$ with a similar procedure. Following the above example the hashing values for the Q-grams $x[1 + Q] = CGACGG$ and $x[2 + Q] = TACTGA$ are respectively 101001001001 and 001011010011.

In this paper we decided to use the Rabin-Karp rolling hash, because it is very intuitive. There are other hashing functions, like the cyclic polynomial hash, that are usually more appropriate because of some desirable properties like uniform distribution in the output space, universality, higher-order independence [17]. In this paper we will focus on the efficient computation of the Rabin-Karp rolling hash. However, with the same paradigm proposed in the following sections, one can compute also the cyclic polynomial hash by replacing in Eq. (1): the function $encode(A)$ with a seed table where the letters of the DNA alphabet are assigned different random 64-bit integers, shifts with rotations, OR with XOR.

2.2 Efficient Spaced Seed Hashing

In many applications [4, 7, 13, 18, 19, 23] it is important to scan a given string x and to compute the hashing values over all positions. In this paper we want to address the following problem.

▶ **Problem 1.** *Let us consider a string $x = x_0x_1 \ldots x_i \ldots x_{n-1}$, of length n, a spaced seed Q and an hash function h that maps strings into a binary codeword. We want to compute the hashing values $\mathcal{H}(x, Q)$ for all the Q-grams of x, in the natural order starting from the first position 0 of x to the last $n - s(Q)$:*

$$\mathcal{H}(x, Q) = \langle h(x[0 + Q]), h(x[1 + Q]), \ldots h(x[n - s(Q)]) \rangle\,.$$

Clearly, in order to address Problem 1, it is possible to use Equation 1 for each position of x. Note that, in order to compute the hashing function $h(x[i+Q])$ for a given position, the number of symbols that have to be extracted from x and encoded into the hash is equal to the weight of the seed $|Q|$. Thus such an approach can be very time consuming, requiring the encoding of $|Q|(n-s(Q))$ symbols. In summary, loosely speaking, in the above process each symbol of x is read and encoded into the hash $|Q|$ times.

In this paper we present a solution for Problem 1 that is optimal in the number of encoded symbols. The scope of this study is to minimize the number of times that a symbol needs to be read and encoded for the computation of $\mathcal{H}(x,Q)$. Since the hashing values are computed in order, starting from the first position, the idea is to speed up the computation of the hash at a position i by reusing part of the hashes already computed at previous positions.

As mentioned above, using Equation (1) in each position of an input string x is a simple possible way to compute the hashing values $\mathcal{H}(x,Q)$. However, we can study how the hashing values are built in order to develop a better method. For example, let us consider the simple case of a contiguous k-mers. Given the hashing value at position i it is possible to compute the hashing for position $i+1$, with three operations: a rotation, the deletion of the encoding of the symbol at position i, and the insertion of the encoding of the symbol at position $i+k$, since the two hashes share $k-1$ symbols. In fact in [17] the authors showed that this simple observation can speed up the hashing of a string by recursively applying these operations. However, if we consider the case of a spaced seed Q, we can clearly see that this observation does not hold. In fact, in the above example, two consecutive Q-grams, like $x[0+Q] = ATGATG$ and $x[1+Q] = CGACGG$, do not necessarily have much in common.

In the case of spaced seeds the idea of reusing part of the previous hash to compute the next one needs to be further developed. More precisely, because of the shape of a spaced seed, we need to explore not only the hash at the previous position, but all the $s(Q)-1$ previous hashes.

Let us assume that we want to compute the hashing value at position i and that we already know the hashing value at position $i-j$, with $j < s(Q)$. We can introduce the following definition of $\mathcal{C}_j = \{k-j \in Q : k \in Q \wedge m(k-j) = m(k) - m(j)\}$ as the positions in Q that after j shifts are still in Q with the propriety of $m(k-j) = m(k) - m(j)$. In other words, if we are processing the position i of x and we want to reuse the hashing value already computed at position $i-j$, $\mathcal{C}_j$ represents the symbols of $h(x[i-j+Q])$ that we can keep while computing $h(x[i+Q])$. More precisely, we can keep the encoding of $|\mathcal{C}_j|$ symbols from that hash and insert the remaining $|Q|-|\mathcal{C}_j|$ symbols at positions $Q \setminus C_j$.

▶ **Example 3.** If we know the first hashing value $h(x[0+Q])$ and we want to compute the second hash $h(x[1+Q])$, the following example show how to construct C_1.

k		0	1	2	3	4	5	6	7
Q		1	0	1	1	1	0	1	1
Q«1	1	0	1	1	1	0	1	1	
m(k)		0	1	1	2	3	4	4	5
m(k)−m(1)	−1	0	0	1	2	3	3	4	
C_1				2	3			6	

The symbols at positions $C_1 = \{2,3,6\}$ of the hash $h(x[1+Q])$ have already been encoded in the hash $h(x[0+Q])$ and we can keep them. In order to complete $h(x[1+Q])$, the remaining $|Q| - |\mathcal{C}_1| = 3$ symbols need to be read from x at positions $i+k$, where $i=1$ and $k \in Q \backslash C_1 = \{0,4,7\}$.

x	A	C	T	G	A	C	T	G	G	A
$x[0+Q]$	A		T	G	A		T	G		
C_1				2	3			6		
$Q \backslash C_1$		0				4			7	
$x[1+Q]$		C		G	A	C		G	G	

Note that the definition of $|\mathcal{C}_j|$ is not equivalent to the overlap complexity of two spaced seeds, as defined in [11]. In some cases, like the one presented above, the overlap complexity coincides with $|\mathcal{C}_1| = 3$. However, there are other cases where $|\mathcal{C}_j|$ is smaller then the overlap complexity.

▶ **Example 4.** Let us consider the hash at position 2 $h(x[2+Q])$, and the hash at position 0 $h(x[0+Q])$. In this case we are interested in $\mathcal{C}_2$.

k			0	1	2	3	4	5	6	7
Q			1	0	1	1	1	0	1	1
Q«2	1	0	1	1	1	0	1	1		
m(k)			0	1	1	2	3	4	4	5
m(k)−m(2)	−1	0	0	1	2	3	3	4		
C_2			0				4			

The only symbols that can be preserved from $h(x[0+Q])$ in order to compute $h(x[2+Q])$ are those at positions 0 and 4, whereas the overlap complexity is 3.

For completeness we report all values of $\mathcal{C}_j$:

$$\begin{aligned}\mathcal{C} &= < \mathcal{C}_1, \ldots, \mathcal{C}_7 > \\ &= < \{2,3,6\}, \{0,4\}, \{0,3,4\}, \{0,2,3\}, \{2\}, \{0\}, \{0\} > \ .\end{aligned}$$

In order to address Problem 1, we need to find, for a given position i, the best previous hash that ensures to minimize the number of times that a symbol needs to be read and encoded, in order to compute $h(x[i+Q])$. We recall that $|\mathcal{C}_j|$ represents the number of symbols that we can keep from the previous hash at position $i-j$, and thus the number of symbols that needs to be read and encoded are $|Q \setminus C_j|$. To solve Problem 1 and to minimize the number of symbols that needs to be read, $|Q \setminus C_j|$, it is enough to search for the j that maximizes $|\mathcal{C}_j|$. The best previous hash can be detected with the following function:

$$ArgBH(s) = \arg\max_{j \in [1,s]} |\mathcal{C}_j| \, .$$

If we have already computed the previous j hashings, the best hashing value can be found at position $i - ArgBH(j)$, and will produce the maximum saving $|\mathcal{C}_{ArgBH(j)}|$ in terms of symbols that can be kept. Following the above observation we can compute all hashing values $\mathcal{H}(x,Q)$ incrementally, by using dynamic programming as described by the pseudocode of Algorithm 1.

The above dynamic programming algorithm scans the input string x and computes all hashing value according to the spaced seed Q. In order to better understand the amount of savings we evaluate the above algorithm by counting the number of symbols that are read and encoded. First, we can consider the input string to be long enough so that we can discard the transient of the first $s(Q)-1$ hashes. Let us continue to analyze the spaced seed 10111011. If we use the standard function $h(x[i+Q])$ to compute all hashes, each symbol of x is read $|Q| = 6$ times. With our algorithm, we have that $|\mathcal{C}_{ArgBH(7)}| = 3$ and thus

Algorithm 1 Fast Spaced Seed Hashing

```
1: for $i := 0$ to $|x| - s(Q)$ do
2:     if $(i == 0)$ then
3:         $h_0 :=$ compute $h(x[0 + Q])$;
4:     else if $(i < s(Q) - 1)$ then
5:         $h_i := h_{i-ArgBH(i)} \gg m(ArgBH(i)) * log_2|\mathcal{A}|$;
6:         for all $k \in \mathcal{Q} \backslash C_{ArgBH(i)}$ do
7:             insert $encode(x_{i+k})$ at position $m(k) * log_2|\mathcal{A}|$ of $h_i$;
8:         end for
9:     else
10:        $h_i := h_{i-ArgBH(s(Q)-1)} \gg m(ArgBH(s(Q) - 1)) * log_2|\mathcal{A}|$;
11:        for all $k \in \mathcal{Q} \backslash C_{ArgBH(s(Q)-1)}$ do
12:            insert $encode(x_{i+k})$ at position $m(k) * log_2|\mathcal{A}|$ of $h_i$;
13:        end for
14:    end if
15: end for
```

half of the symbols do need to be encoded again, overall each symbol is read 3 times. The amount of saving depends on the structure of the spaced seed. For example, the spaced seed 10101010101, with the same weight $|Q| = 6$, is the one that ensures the best savings ($|\mathcal{C}_{ArgBH(10)}| = 5$). In fact, with our algorithm, we can compute all hashing values while reading each symbol of the input string only once, as with contiguous k-mers. To summarize, if one needs to scan a string with a spaced seed and to compute all hashing values, the above algorithm guarantees to minimize the number of symbols to read.

2.3 Efficient Multiple Spaced Seed Hashing

Using multiple spaced seeds, instead of just one spaced seed, is reported to increase the sensitivity [13]. Therefore, applications that exploit such an observation (for example [4, 8, 19]) will benefit from further speedup that can be obtained from the information already computed from multiple spaced seeds.

Our algorithm can be extended to accommodate the need of hashing multiple spaced seeds simultaneously, without backtracking. Let us assume that we have a set $S = s_1, s_2, \ldots, s_{|S|}$ of spaced seeds, from which we can compute the corresponding vectors m_{s_i}. To this purpose, Algorithm 1 needs to be modified as follows. First of all, a new cycle (between steps 2 and 14) is needed to iterate the processing among the set of all spaced seeds. Next, $\mathcal{C}_j$ needs to be redefined so that it compares not only a given spaced seed with itself, but all spaced seeds vs all. In the new definition, $\mathcal{C}_j^{yz} = \{k - j \in s_y : k \in s_z \wedge m_{s_y}(k - j) = m_{s_z}(k) - m_{s_z}(j)\}$, evaluates the number of symbols in common between the seed s_y and the j-th shift of the seed s_z. The function $\mathcal{C}_j^{yz}$ allows to identify, while computing the hash of s_y, the number of symbols in common with the j-th shift of seed s_z. Similarly, we need to redefine $ArgBH(i)$ so that it detects not only the best previous hash, but also the best seed. We define $ArgBSH(y, s) = \arg\max_{z \in [1,|S|], j \in [1,s]} |\mathcal{C}_j^{yz}|$ that returns, for the seed s_y, the pair (s_z, j) representing the best seed s_z and best hash j. With these new definitions we can adjust our algorithm so that, while computing the hash of s_y for a given position i, it can start from the best previous hash identified by the pair $ArgBSH(y, s) = (s_z, j)$. The other steps for the insertion of the remaining symbols (steps 6–7 and 11-12) do not need to be modified.

Table 1 The nine spaced seeds used in the experiments grouped according to their type.

Spaced seeds maximizing the hit probability[19]	
Q1	11110111011100101110010110111111
Q2	11111010111001011011100110111111
Q3	11111010011101011011001110111111
Spaced seeds minimizing the overlap complexity[10]	
Q4	11110101110100110011101111110111
Q5	11101110111011110100101100111111
Q6	11111010010111001111101011011111
Spaced seeds maximizing the sensitivity[10]	
Q7	11110111100110101111101010110111
Q8	11101010111011001101001111111111
Q9	11111101011010111001110110011111

3 Results and discussion

In this section we will discuss the improvement in terms of time speedup of our approach ($T_{FastHash}$) with respect to the time T_{Eq1} needed for computing spaced seeds hashing repeatedly using Eq. (1): speedup $= \frac{T_{Eq1}}{T_{FastHash}}$.

3.1 Spaced seeds and datasets description

The spaced seeds we used have been proposed in literature as maximizing the hit probability [19], minimizing the overlap complexity [10] and maximizing the sensitivity [10]. We tested nine of such spaced seeds, three for each category. The spaced seeds are reported in Table 1 and labeled Q1, Q2, . . . , Q9. Besides these spaced seeds, we also tested Q0, which corresponds to an exact match with a 22mer (all 22 positions are set to 1), and Q10, a spaced seed with repeated '10' and a total of 22 symbols equal to '1'. All spaced seeds Q0–Q10 have the same weight $|Qi| = 22$. Furthermore, in order to compare seeds with different weights but similar density, we computed with rasbhari two sets of seeds with weights 11 and 32 and lengths respectively 16 and 45 (see Tables 3 and 4 in the Appendix).

The datasets we used were taken from previous scientific papers on metagenomic read binning and classification [9, 25]. We considered both simulated datasets (S,L,R), and synthetic datasets (MiSeq, HiSeq, MK_a1, MK_a2, and simBA5). The datasets S_x and L_x contain sets of paired-end reads of length approximately 80bp generated according to the Illumina error profile with an error rate of 1%, while the datasets R_x contain Roche 454 single-end long reads of length approximately 700bp, and a sequencing error of 1%. The synthetic datasets represent mock communities built from real shotgun reads of various species. Table 2 shows, for each dataset, the number of reads and their average length.

All the experiments where run on a laptop equipped with an Intel i74510U cpu at 2GHz, and 16 GB RAM.

3.2 Analysis of the time performances

Figure 1 plots, for each spaced seed, the speedup that is obtainable with our approach with respect to the standard hashing computation. As a reference, the baseline given by the standard approach is about 17 minutes to compute the hash for a given seed on all datasets.

Table 2 Number of reads and average lengths for each of the dataset used in our experiments.

Datasets	number of reads	avg. read length
S6	1426457	80
S7	3307100	80
S9	4468336	80
S10	9981172	80
L5	1016418	80
L6	1182178	80
HiSeq	9989713	91
simBA5	5439738	100
MixK1	9629886	101
MixK2	7149900	101
MiSeq	9933556	131
R7	290473	702
R8	374576	715
R9	588256	715

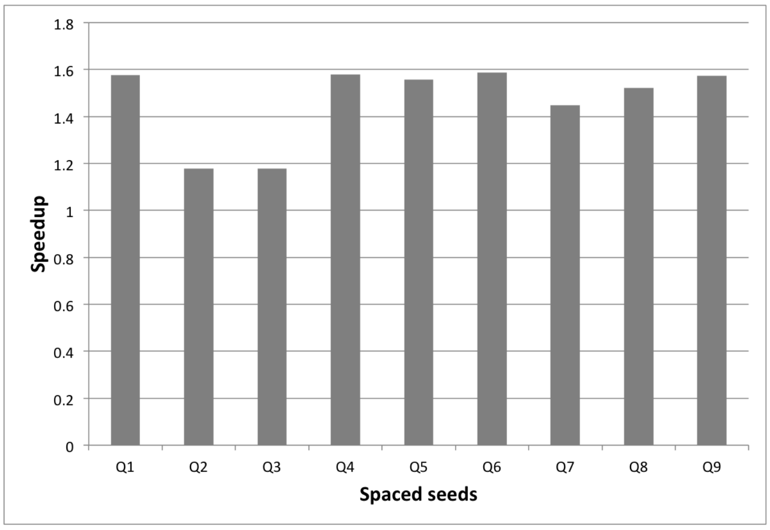


Figure 1 The speedup of our approach with respect to the standard hashing computation, as a function of the spaced seeds used in our experiments.

First of all it can be noticed that our approach improves over the standard algorithm for all of the considered spaced seeds. The smallest improvements are for the spaced seeds Q2 and Q3, both belonging to the class of spaced seeds maximizing the hit probability, for which the speedup is almost 1.2x, and the running time is about 15 minutes. For all the other spaced seeds the speedup is close to 1.6x, thus saving about 40% of the time required by the standard computation, and ending the computation in less than 11 minutes on average.

Figure 2 shows the performances of our approach with respect to the single datasets. In this experiment we considered the best performing spaced seed in each of the classes that we considered, namely Q1, Q6, and Q9, and the two additional special cases Q0 and Q10.

We notice that for the spaced seeds Q0 and Q10 the standard approach requires respectively, 12 and 10 minutes, to process all datasets. This is already an improvement of the standard method with respect to the 17 minutes required with the other seeds Q1–Q9.

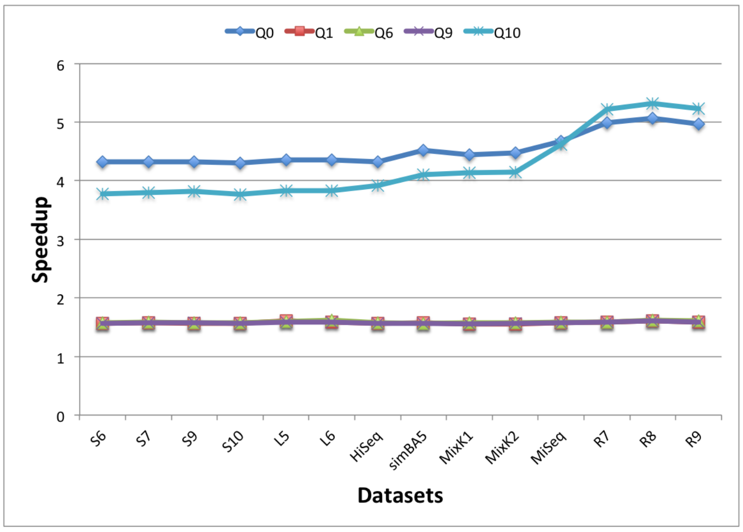


Figure 2 Details of the speedup on each of the considered datasets. Q0 is the solid 22mer, Q10 is the spaced seed with repeated 10. The other reported spaced seeds are the ones with the best performances for each class: Q1 (maximizing the hit probability), Q6 (minimizing the overlap complexity) and Q9 (maximizing the sensitivity).

Nevertheless, with our algorithm the hashing of all dataset can be completed in just 2.7 minutes for Q0 e 2.5 minutes for Q10, with a speedup of 4.5x and 4.2x.

We observe that while the speedup for the spaced seeds Q1, Q6, and Q9 is basically independent on the dataset and about 1.6x, the speedup for both the 22-mer Q0 and the 'alternate' spaced seed Q10 is higher, spanning from 4.3x to 5.3x, depending on the seed and on the dataset. In particular, the speedup increases with the length of the reads and it achieves the highest values for the long read datasets R_7, R_8 and R_9. This behavior is expected, as these datasets have longer reads with respect to the others, thus the effect of the initial transient is mitigated.

3.3 Multiple spaced seed hashing

When the analysis of biological data to perform requires the use of multiple spaced seeds, it is possible to compute the hash of all seeds simultaneously while reading the input string with the method described in Section 2.3.

In Figure 3 we report the comparison between the speedup we obtained when computing the hash for each spaced seed Q1,...,Q9 independently (light grey), and the speedup we obtained when using the multiple spaced seeds approach (dark grey).

In most cases, multiple spaced seed hashing allows for a further improvement of about 2–5%, depending on the dataset. In terms of absolute values, the standard computation to hash all datasets requires 159 minutes, the computation of all seeds independently with the approach described in Section 2.2 takes 109 minutes, while the simultaneous computation of multiple spaced seeds with our method (see Section 2.3) takes 107 minutes. When considering all datasets the average speedup increases from 1.45x (indipendent computation) to 1.49x (simultaneous computation). The small improvement can be justified by the fact that the spaced seeds considered are by construction with minimal overlap.

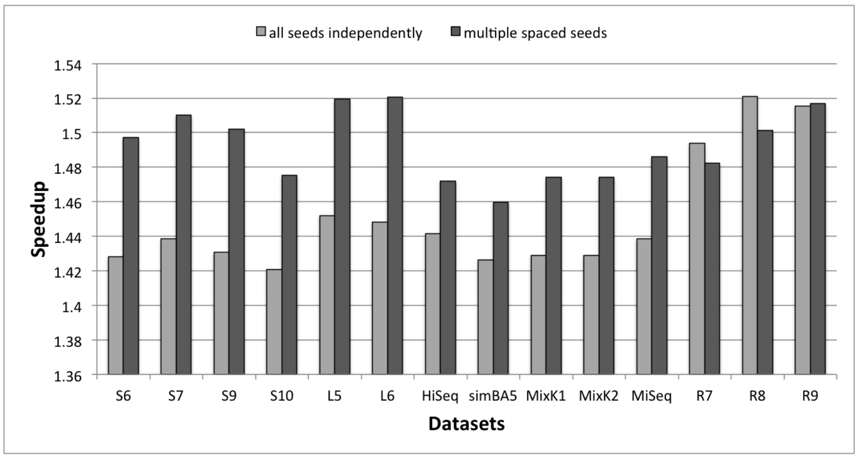


Figure 3 Details of the time speedup of our approach with the multiple spaced seeds hashing (dark grey) and of our approach with each spaced seed hashed independently (light grey).

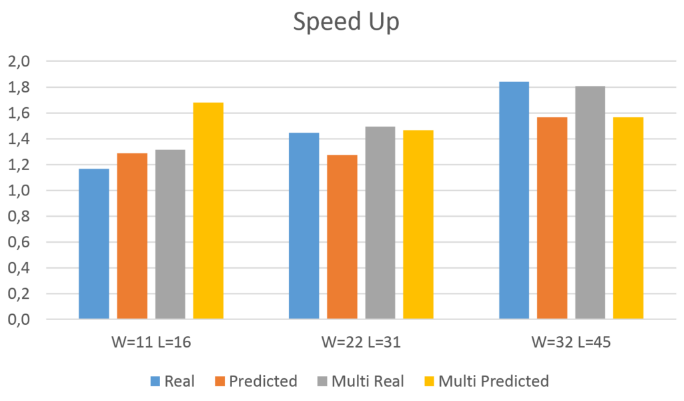


Figure 4 The theoretical and real speedup of our approach with respect to the standard hashing computation, as a function of the spaced seeds weight.

3.4 Spaced Seeds with Different Weights

In order to compare the performance of our method on spaced seeds with different weights we generated other two sets of nine spaced seeds with rasbhari, all with similar density (see Tables 3 and 4 in the Appendix). In Figure 4 are reported the average speedup (Real), over all datasets, for the three different groups of nine seeds. In the same Figure we include also the speedup when all nine seeds are used simultaneously (Multi) and the theoretical speedup predicted by our method (Predicted).

It can be observed that if the weight of the seeds grows then also the real speedup grows. This is expected, because if a seed has more 1s, then the chances to reuse part of the seed increase. As, for the theoretical predicted speedups, these are usually in line with the real speedups even if the absolute values are not necessarily close. We suspect that the model we use, where shifts and insertions have the same cost, is too simplistic. Probably, the real computational cost for the insertion of a symbol is greater than the cost for shifting, and also cache misses might play a role.

If the theoretical speedup for multiple seeds is greater than the theoretical speedup for independent seeds, this indicates that in principle, with multiple seeds, it is possible to improve with respect to the computation of seeds independently. It is interesting to note that the real results confirm these predictions. For example, in the multiple seeds with weights 32, it is impossible to improve both theoretically and in practice. In the other two cases, the computation of multiple seeds is faster in practice as correctly predicted by the theoretical speedup.

4 Conclusions

We presented a new approach for spaced seeds hashing that exploits the information available from previous matches in order to minimize the number of positions that need to be recomputed. The experiments we performed on several datasets showed that our method has a speedup of 1.6x with respect to the standard approach used to compute spaced seeds hashing, for several kind of spaced seeds defined in the literature. Furthermore, the gain greatly improved in special cases, where seeds show a high autocorrelation, and for which a speed up of about 4x to 5x can be achieved.

References

1 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. `doi:10.1016/S0022-2836(05)80360-2`.

2 Daniel G. Brown, Ming Li, and Bin Ma. A tutorial of recent developments in the seeding of local alignment. *Journal of Bioinformatics and Computational Biology*, 02(04):819–842, 2004. `doi:10.1142/S0219720004000983`.

3 Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419, 2001. `doi:10.1093/bioinformatics/17.5.419`.

4 Karel Břinda, Maciej Sykulski, and Gregory Kucherov. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics*, 31(22):3584, 2015. `doi:10.1093/bioinformatics/btv419`.

5 Matteo Comin and Morris Antonello. Fast entropic profiler: An information theoretic approach for the discovery of patterns in genomes. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 11(3):500–509, May 2014. `doi:10.1109/TCBB.2013.2297924`.

6 Matteo Comin, Andrea Leoni, and Michele Schimd. Clustering of reads with alignment-free measures and quality values. *Algorithms for Molecular Biology*, 10(1):4, 2015. `doi:10.1186/s13015-014-0029-x`.

7 Aaron E. Darling, Todd J. Treangen, Louxin Zhang, Carla Kuiken, Xavier Messeguer, and Nicole T. Perna. Procrastination leads to efficient filtration for local multiple alignment. In Philipp Bücher and Bernard M. E. Moret, editors, *Algorithms in Bioinformatics: 6th International Workshop, WABI 2006, Zurich, Switzerland, September 11-13, 2006. Proceedings*, pages 126–137. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. `doi:10.1007/11851561_12`.

8 Samuele Girotto, Matteo Comin, and Cinzia Pizzi. Binning metagenomic reads with probabilistic sequence signatures based on spaced seeds. *To Appear*, 2017.

9 Samuele Girotto, Cinzia Pizzi, and Matteo Comin. MetaProb: accurate metagenomic reads binning based on probabilistic sequence signatures. *Bioinformatics*, 32(17):i567–i575, September 2016. `doi:10.1093/bioinformatics/btw466`.

10 Lars Hahn, Chris-André Leimeister, Rachid Ounit, Stefano Lonardi, and Burkhard Morgenstern. Rasbhari: Optimizing spaced seeds for database searching, read mapping and alignment-free sequence comparison. *PLOS Computational Biology*, 12(10):1–18, 10 2016. `doi:10.1371/journal.pcbi.1005107`.

11 Lucian Ilie, Silvana Ilie, and Anahita Mansouri Bigvand. SpEED: fast computation of sensitive spaced seeds. *Bioinformatics*, 27(17):2433, 2011. `doi:10.1093/bioinformatics/btr368`.

12 Uri Keich, Ming Li, Bin Ma, and John Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253–263, 2004. `doi:10.1016/S0166-218X(03)00382-2`.

13 Chris-Andre Leimeister, Marcus Boden, Sebastian Horwege, Sebastian Lindner, and Burkhard Morgenstern. Fast alignment-free sequence comparison using spaced-word frequencies. *Bioinformatics*, 30(14):1991, 2014. `doi:10.1093/bioinformatics/btu177`.

14 Stinus Lindgreen, Karen L. Adair, and Paul Gardner. An evaluation of the accuracy and speed of metagenome analysis tools. *Scientific Reports*, 6, 2016. Article No. 19233. `doi:10.1038/srep19233`.

15 Bin Ma and Ming Li. On the complexity of the spaced seeds. *Journal of Computer and System Sciences*, 73(7):1024–1034, 2007. Bioinformatics {III}. `doi:10.1016/j.jcss.2007.03.008`.

16 Bin Ma, John Tromp, and Ming Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440, 2002. `doi:10.1093/bioinformatics/18.3.440`.

17 Hamid Mohamadi, Justin Chu, Benjamin P. Vandervalk, and Inanc Birol. ntHash: recursive nucleotide hashing. *Bioinformatics*, page btw397, July 2016. `doi:10.1093/bioinformatics/btw397`.

18 Taku Onodera and Tetsuo Shibuya. The gapped spectrum kernel for support vector machines. In *Proceedings of the 9th International Conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM'13, pages 1–15, Berlin, Heidelberg, 2013. Springer-Verlag. `doi:10.1007/978-3-642-39712-7_1`.

19 Rachid Ounit and Stefano Lonardi. Higher classification sensitivity of short metagenomic reads with CLARK-S. *Bioinformatics*, 32(24):3823, 2016. `doi:10.1093/bioinformatics/btw542`.

20 Rachid Ounit, Steve Wanamaker, Timothy J. Close, and Stefano Lonardi. CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):1–13, 2015. `doi:10.1186/s12864-015-1419-2`.

21 Laxmi Parida, Cinzia Pizzi, and Simona E. Rombo. Irredundant tandem motifs. *Theoretical Computer Science*, 525:89–102, 2014. Advances in Stringology. `doi:10.1016/j.tcs.2013.08.012`.

22 C. Pizzi, P. Rastas, and E. Ukkonen. Finding significant matches of position weight matrices in linear time. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(1):69–79, Jan 2011. `doi:10.1109/TCBB.2009.35`.

23 Stephen M. Rumble, Phil Lacroute, Adrian V. Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. Shrimp: Accurate mapping of short color-space reads. *PLOS Computational Biology*, 5(5):1–11, 05 2009. `doi:10.1371/journal.pcbi.1000386`.

24 Ariya Shajii, Deniz Yorukoglu, Yun William Yu, and Bonnie Berger. Fast genotyping of known snps through approximate k-mer matching. *Bioinformatics*, 32(17):i538, 2016. `doi:10.1093/bioinformatics/btw460`.

25 Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15:R46, 2014. `doi:10.1186/gb-2014-15-3-r46`.

A Appendix

Table 3 Nine spaced seeds with $W = 11$ and length 16 computed with rasbhari minimizing overlap complexity.

Q10	1011101100101110
Q11	1100111100011110
Q12	1101011100011110
Q13	1101101110111000
Q14	1110110010110110
Q15	1111001100101110
Q16	1111001101110010
Q17	1111100011010110
Q18	1111110001011100

Table 4 Nine spaced seeds with $W = 32$ and length 45 computed with rasbhari minimizing overlap complexity.

Q19	100111111111110010010111101111001110110110111
Q20	101001111001011111011110111111110001010111111
Q21	110100110101111100011111011011111111111110001
Q22	110101011001100111110101110011001111111111111
Q23	110111011111111101101111101010010000011111111
Q24	111011100111010001101111001111110011111110111
Q25	111100011011010010011111111011111111100011111
Q26	111101001101110011101110101011101110111011111
Q27	111101101111110001111110001011101011110111011

A General Framework for Gene Tree Correction Based on Duplication-Loss Reconciliation

Nadia El-Mabrouk[1] and Aïda Ouangraoua[2]

1 **Département d'informatique et de recherche opérationnelle, Université de Montréal, Montreal, QC, Canada**
mabrouk@iro.umontreal.ca

2 **Département d'informatique, Université de Sherbrooke, Sherbrooke, QC, Canada**
aida.ouangraoua@usherbrooke.ca

Abstract

Due to the key role played by gene trees and species phylogenies in biological studies, it is essential to have as much confidence as possible on the available trees. As phylogenetic tools are error prone, it is a common task to use a correction method for improving an initial tree. Various correction methods exist. In this paper we focus on those based on the Duplication-Loss reconciliation model. The polytomy resolution approach consists in contracting weakly supported branches and then refining the obtained non-binary tree in a way minimizing a reconciliation distance with the given species tree. On the other hand, the supertree approach takes as input a set of separated subtrees, either obtained for separared orthology groups or by removing the upper branches of an initial tree to a certain level, and amalgamating them in an optimal way preserving the topology of the initial trees. The two classes of problems have always been considered as two separate fields, based on apparently different models. In this paper we give a unifying view showing that these two classes of problems are in fact special cases of a more general problem that we call LabelGTC, whose input includes a 0-1 edge-labelled gene tree to be corrected. Considering a tree as a set of triplets, we also formulate the TripletGTC Problem whose input includes a set of gene triplets that should be preserved in the corrected tree. These two general models allow to unify, understand and compare the principles of the duplication-loss reconciliation-based tree correction approaches. We show that LabelGTC is a special case of TripletGTC. We then develop appropriate algorithms allowing to handle these two general correction problems.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases gene tree correction, supertree,- polytomy, reconciliation, phylogeny

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.8

1 Introduction

Studying the functional specificities of gene copies, such as their role in metabolic pathways of interest, usually requires a trusted gene tree. However, for various reasons related to the specificities of phylogenetic software, the considered evolutionary models or errors in the multiple alignments, constructed trees are usually not fully satisfactory. Consequently, most tree construction methods integrate measures of statistical support obtained by bootstrapping or jackknifing [3], reflecting the confidence we have on the prediction. A strong support on a branch reflects a strong support on the clade (in case of a rooted tree) or the bipartition (in case of an unrooted tree) represented by this branch. Results coming out from bioinformatics pipelines should then be analyzed in light of this uncertainty in the considered trees.

Alternatively, trees may be corrected before carrying on with the biological analysis. Due to the fact that standard gene tree databases, such as Ensembl [15], are known to be error prone, gene tree correction methods are frequently used upstream to obtain better trees for gene families (c.f. for example NOTUNG [2], TreeFix [16], ProfileNJ[10], MowgliNNI [9], ecceTERA [4], MRL [8]). Most are based on minimizing a reconciliation distance with a given species tree, some including horizontal gene transfer. In this paper, we restrict ourselves to the duplication-loss reconciliation distance that we simply call the reconciliation distance. Moreover, we focus on polytomy-based and supertree-based correction methods that cover a large set of correction methods, although not all (for example, TreeFix [16] relies on exploring a space surrounding an initial tree, and cannot be categorized as a polytomy or supertree-based correction method.)

Polytomy-based correction methods rely on contracting branches with weak support, leading to a non-binary tree, and then resolving multifurcated nodes (polytomies) in a binary way minimizing the reconciliation distance with the species tree [2, 5, 10, 12, 17]. The most efficient algorithm for resolving a non-binary tree is linear in the size of the tree [5, 17]. Such a polytomy resolution approach preserves the subtrees in terms of topology and gene content. In other words, the exhibited monophily of input gene clusters is not challenged by a polytomy resolution method.

Other methods rather stand on amalgamating "trusted" partial trees into a single one for the whole family [8, 14]. Such partial trees may be obtained by constructing them independently for partial sets of orthologs, or by removing weakly supported branches of an initial tree. In [6, 7], we have formalized this approach in terms of a supertree method for gene trees. The defined SuperGeneTree (SGT) problem consists in constructing, from a set of partial trees, a tree containing them all and minimizing the reconciliation distance. A simplest version considering the duplication distance has been shown NP-hard [6]. Conceptually, the supertree-based correction method is more general than the polytomy-based one, as only the topology of initial trees should be preserved in the former case, while the latter requires also to preserve the clades. Although it may be relevant to challenge the monophiletic nature of input partial gene trees, SGT may lead to a drastic reorganization giving rise to a tree grouping genes that are far apart in the original tree. To avoid this problem, we also introduced the Triplet Respecting Supergenetree (TRS) problem [7], asking for a supertree displaying all input subtrees, while preserving the topology of any triplet of genes taken from three different subtrees (clades).

The polytomy-based and supertree-based models of gene tree correction have been developed separately, considering separate assumptions and constraints. Some assumptions are actually questionable such as the one considering upper branches as the ones that should be removed in the case of a polytomy resolution, or alternatively kept in the case of TRS. In fact, support can be strong or weak on any branch of the tree. Moreover, in the absence of a unifying model, the conservative or permissive nature of each method with respect to another one can only be tested empirically. Here, we show that all these methods can in fact be considered in a unifying way, as special cases of a more general gene tree correction problem taking as input a 0-1 edge-labelling derived from edge statistical supports.

In Section 3, we introduce the Label Respecting GeneTreeCorrection (LabelGTC) Problem whose input includes a 0-1 edge-labelled gene tree to be corrected. We show that the polytomy related and supertree related correction problems are all special cases of this problem. In Section 4, we then define Triplet Respecting GeneTreeCorrection (TripletGTC) Problem, a more general problem whose input includes a set of gene triplets that should be preserved in the corrected tree. We show that LabelGTC is a special case of

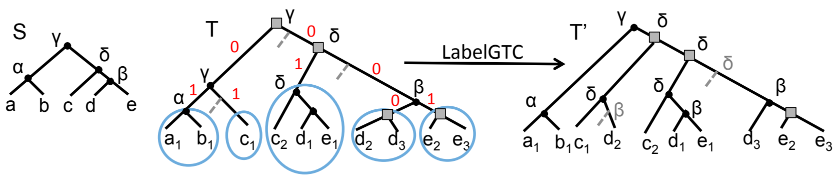


Figure 1 Left. A species tree S for $\Sigma = \{a, b, c, d, e\}$, a reconciled 0-1 edge-labelled gene tree T for $\Gamma = \{a_1, b_1, c_1, c_2, d_1, d_2, d_3, e_1, e_2, e_3\}$ where each leaf x_i denotes a gene belonging to species x, and a covering set $\mathcal{T}$ of subtrees for T indicated by circles around each subtree. Square nodes are duplications and circular nodes are speciations. Internal nodes are labelled according to corresponding ancestral species in S. Dotted lines are losses. **Right.** A supertree for $\mathcal{T}$ of minimum reconciliation cost (cost of 5) respecting the edge labelling of T.

TripletGTC. As these problems include the SGT problem as a sub-problem, they are both NP-hard for the duplication distance. In Section 5, we then exhibit a recursive algorithm for LabelGTC and show how it can be used to define a heuristic algorithm for TripletGTC. Finally, a variant of the LabelGTC Problem, allowing for an extended labelling, is presented in Section 6. Developed algorithms have the same exponential time-complexity as the one previously developed for the SGT problem [7], that is a particular case of the new problems studied here. All missing proofs are given in Appendix.

2 Preliminaries on gene tree correction methods

Notations on trees: All considered trees are rooted. A tree is *binary* if each internal node (all nodes except the leaves) has exactly two children, and *non-binary* otherwise. When not mentioned explicitly, all trees are considered binary (except for the polytomy resolution problem; we mention it explicitly in this case).

A tree T with leafset $\mathcal{L}(T) = X$ is called a tree for X. If X is a set of species, then T is a *species tree*. We denote by $V(T)$ the node set and by $E(T)$ the edge set of T. An edge of $E(T)$ is written as a pair (x, y) of two adjacent nodes, where x is the closest to the root. A node x is *an ancestor* of y if it is on the path from y to the root (excluding y). In this case, y is called *a descendant* of x. Similarly, an edge (x', y') is *an ancestor* of an edge (x, y) if it is on the path from (x, y) to the root. Given a node x, $T[x]$ is the subtree of T rooted at x and $\mathcal{L}(x)$ the leafset of $T[x]$. Two subtrees $T[x]$ and $T[y]$ are *separated* in T iff $x \neq y$ and none of x or y is an ancestor of the other. It follows that $\mathcal{L}(x) \cap \mathcal{L}(y) = \emptyset$.

The *lowest common ancestor* of $L' \subseteq \mathcal{L}(T)$, denoted $lca_T(L')$, is the ancestor common to all leaves in L' that is the most distant from the root. $T|_{L'}$ is the tree with leafset L' obtained from the subtree of T rooted at $lca_T(L')$ by removing all leaves that are not in L', and then all internal nodes of degree 2, except the root of this tree. Let T' be a tree such that $\mathcal{L}(T') = L' \subseteq \mathcal{L}(T)$. We say that T *displays* T' iff $T|_{L'}$ is label-isomorphic to T'. In this case, we also say that T *preserves* the topology of T'.

A set $\mathcal{T}$ of trees, with possibly overlapping leafsets, is *consistent* if there is a tree T' on the union of their leafsets displaying them all. Such a tree T' is called a *supertree* for $\mathcal{T}$. For example, in Figure 1, the tree T' is a supertree for the set of subtrees of T circled in blue color.

Gene tree and reconciliation: A *gene family* is a set of genes Γ accompanied with a *mapping function* $s : \Gamma \rightarrow \Sigma$ from each gene to its corresponding species in a set of species Σ. Consider a gene family Γ where each gene $x \in \Gamma$ belongs to a species $s(x)$ of Σ. The evolutionary

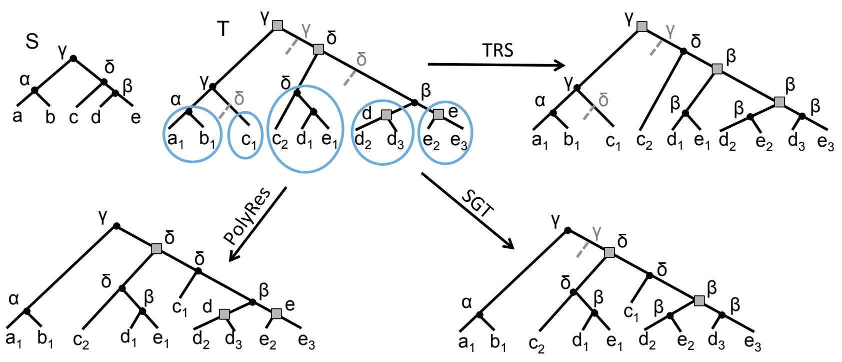


Figure 2 Top left. Species tree S for $\Sigma = \{a, b, c, d, e\}$, gene tree T for $\Gamma = \{a_1, b_1, c_1, c_2, d_1, d_2, d_3, e_1, e_2, e_3\}$ with a covering set $\mathcal{T}$ of subtrees for T as in Figure 1 (without the 0-1 labelling of edges). **Bottom left.** A polytomy resolution supertree for $\mathcal{T}$ of minimum reconciliation cost (cost of 3). **Bottom right.** A supertree for $\mathcal{T}$ of minimum reconciliation cost (cost of 3). **Top right.** A triplet respecting supertree for $\mathcal{T}$ of minimum reconciliation cost (cost of 5). Note that the three optimal supertrees for the TRS, SGT and PolyRes problems differ from the optimal supertree for the LabelGTC problem depicted in Figure 1, because the implicit 0-1 edge-labellings differ from the one in Figure 1.

history of Γ can be represented as a *gene tree* T for Γ. Each internal node of T refers to an ancestral gene at the moment of an event, either speciation (*Spec*) or duplication (*Dup*). Let S be a species tree for Σ. The mapping s is extended to be defined from $V(T)$ to $V(S)$ as follows: if x is an internal node of T, then $s(x) = lca_S(\{s(x') : x' \in \mathcal{L}(x)\})$.

When the type of event is known for each internal node, the gene tree T is said *labelled*. Formally, a *labelled gene tree* for Γ is a pair (T, ev_T), where T is a tree for $\mathcal{L}(T) = \Gamma$, and $ev_T : V(T) \setminus \mathcal{L}(T) \to \{Dup, Spec\}$ is a function labelling each internal node of T as a duplication or a speciation node.

The *lca-reconciliation* (or simply *reconciliation* for short) of a gene tree T with a species tree S is the labelled tree (T, ev_T) obtained by labelling each internal node x of T with children x_l and x_r as *Spec* iff $s(x_l)$ and $s(x_r)$ are separated in S, and as *Dup* otherwise. The mapping function s and the node labelling also induce gene loss events on branches of the gene tree (see Figures 1 and 2 for examples of lca-reconciliations of gene trees with species trees, and the induced loss events). The *reconciliation cost* of a labelled tree (T, ev_T) is its number of duplication nodes and induced loss events.

We end this section with a final definition. Let T and T' be two trees on Γ. If (x, y) is an edge of $E(T)$ and there is an edge (x', y') in $E(T')$ such that $\mathcal{L}(y) = \mathcal{L}(y')$, we say that T' *preserves* the edge (x, y) of T.

3 A unifying view on gene tree correction problems

In the remaining of the paper, S is a species tree on a species set Σ and T is a gene tree for a gene set Γ.

The correction problem asks for a "better tree" T' for Γ according to the reconciliation cost. The various versions of the problem differ on the flexibility we have on modifying T. Which parts of T should be preserved ? The most natural way to do is to preserve all well supported branches, according to a given statistical support, and be allowed to modify all weakly supported branches. Notice that the support on a branch (x, y) reflects the confidence we have on the fact that $\mathcal{L}(y)$ represents a separate clade in the gene family.

The underlying representation is a 0-1 edge labelling of T edges, where 0 indicates a low support and 1 a high support according to a certain threshold.

In addition, if the tree T contains a set of separated subtrees whose topologies are "trusted", they should be considered as an additional parameter. Such trusted topologies may, for instance, be those obtained separately for different orthology groups agreeing with the species tree, and used to build T.

Accordingly, the most general gene tree correction problem is formulated below, where a *covering set of subtrees* $\mathcal{T}$ for T is a set of separated subtrees of T, $\mathcal{T} = \{T[x_1], T[x_2], \dots, T[x_n]\}$ such that $\cup_{i=1}^{n}\mathcal{L}(x_i) = \mathcal{L}(T)$, and a 0-1 edge labelling for T is a function l defined from the set of edges $E(T)$ to $\{0,1\}$. In the following formulation, edge labels are ignored for the trees of $\mathcal{T}$ (see Figure 1 for an illustration of the LabelGTC Problem).

Label Respecting GeneTreeCorrection (LabelGTC) Problem:
Input: A species tree S, a gene tree T, a covering set of trees $\mathcal{T}$ for T and a 0-1 edge labelling l for T.
Output: A supertree T' for $\mathcal{T}$ of minimum reconciliation cost such that: if $(x, y) \in E(T)\backslash E(\mathcal{T})$ is such that $l(x, y) = 1$, then there is an edge (x', y') in $E(T')$ such that $\mathcal{L}(y) = \mathcal{L}(y')$.

Notice that if no information on "trusted" separated subtrees is available, then each tree of $\mathcal{T}$ is simply restricted to a leaf of T, in which case $\mathcal{T}$ simply refers to the leafset of T.

The above formulation is not the one actually considered in the literature. Some special cases of the LabelGTC problem where the input covering set of trees $\mathcal{T}$ is the leafset of T were considered in [2, 5, 10, 12] under the name of Polytomy Resolution problems, and in [6, 7, 8, 14] under the name of Supertree problems. In the following paragraphs, we recall these polytomy related and supertree related correction problems (see Figure 2 for an illustration of the problems). We will show later that they are all special cases of the general LabelGTC formulation.

The polytomy resolution problem

The general version of the problem consists in contracting all weakly supported internal branches of the input gene tree T, leading to a non-binary tree denoted by T^{nb}, and then finding a binary refinement of T^{nb} minimizing the reconciliation cost. Formally, given a non-binary gene tree T^{nb}, a binary tree T' is a *binary refinement* of T^{nb} if for any node x of T^{nb}, there exists a node x' in T' such that $\mathcal{L}(x) = \mathcal{L}(x')$.

The simplest form of a non-binary tree is a *polytomy* defined as a set of leaves $\mathcal{L}$, all being adjacent to the root. Given a non-binary gene tree T^{nb}, it has been shown that a refinement of minimum cost for T^{nb} can be obtained by a depth-first procedure iteratively solving each polytomy $T[x]$, for each internal non-binary node x of T^{nb}. This is the reason for the name given to the general problem formulated below. The restriction to a single polytomy is formulated afterwards. It is also required in the main Theorems 1 and 6 of the paper.

Mutiple Polytomy Resolution (M-PolyRes) Problem:
Input: A species tree S and a 0-1 edge-labelled gene tree T.
Output: A binary refinement of T^{nb} minimizing the reconciliation distance.

As stated above, the simplest form of the problem is a single polytomy. It consists in having a single non-binary node in T^{nb}, the root, such that the subtrees rooted at the children of the root are "trusted" partial trees that should remain rooted subtrees of the final tree (see the tree obtained from PolyRes in Figure 2).

POLYTOMY RESOLUTION (POLYRES) PROBLEM:
Input: A species tree S, a gene tree T and a covering set of trees $\mathcal{T}$ for T.
Output: A supertree T' for $\mathcal{T}$ of minimum reconciliation cost such that for any tree $T_i \in \mathcal{T}$, $T'_{|\mathcal{L}(T_i)} = T_i$.

The supertree resolution problem

The above formulation of the polytomy resolution problem is a special case of the more general supertree problem, where the constraint of preserving the monophily of input "trusted" partial trees is relaxed. In [6], the SuperGenetree correction problem is formulated as follows.

SUPERGENETREE (SGT) PROBLEM:
Input: A species tree S, a gene tree T and a covering set of trees $\mathcal{T}$ for T.
Output: A supertree T' for $\mathcal{T}$ of minimum reconciliation cost.

The triplet-respecting supertree problem

To avoid having a supertree grouping genes that are far apart in the original tree, we introduced, in [7], an alternative problem allowing to restrict the output space to supertrees preserving the topology of any triplet of genes taken from three different input subtrees of $\mathcal{T}$. The triplet-based constrained supertree problem is the following.

TRIPLET-RESPECTING SUPERGENETREE (TRS) PROBLEM:
Input: A species tree S, a gene tree T and a covering set of trees $\mathcal{T}$ for T.
Output: A supertree T' for $\mathcal{T}$ of minimum reconciliation cost respecting the following property: for any triplet (a, b, c) where a, b and c are genes of Γ being leaves of three different trees of $\mathcal{T}$, $T'_{|\{a,b,c\}} = T_{|\{a,b,c\}}$.

For example, the tree which is a solution of the SGT Problem in Figure 2 is not a solution of the TRS problem as the triplet (a_1, c_1, c_2), where each gene belongs to a separate subtree of the covering set $\mathcal{T}$ of T, has the topology $(a_1, (c_1, c_2))$ in this tree while it has the topology $((a_1, c_1), c_2)$ in T.

A unifying view

The following Theorem shows that the polytomy related and supertree related problems are in fact special cases of the general LABELGTC problem. Given a covering set of subtrees $\mathcal{T}$ for T, we call a *terminal edge* an edge of $E(T) \setminus E(\mathcal{T})$ which is adjacent to a tree of $\mathcal{T}$. All other edges of $E(T) \setminus E(\mathcal{T})$ are called *non-terminal edges* (see Figure 3 for an illustration).

▶ **Theorem 1.** *Let T be a 0-1 edge-labelled gene tree and $\mathcal{T}$ be a covering set for T. Then the* LABELGTC *Problem is reduced to:*
1. *the* M-POLYRES *Problem if $\mathcal{T} = \mathcal{L}(T)$; Otherwise:*
2. *the* POLYRES *Problem if all non-terminal edges are labelled* 0*, and all terminal edges are labelled* 1*;*
3. *the* SGT *Problem if all non-terminal and terminal edges are labelled* 0*;*
4. *the* TRS *Problem if all non-terminal edges are labelled* 1*, and all terminal edges are labelled* 0*.*

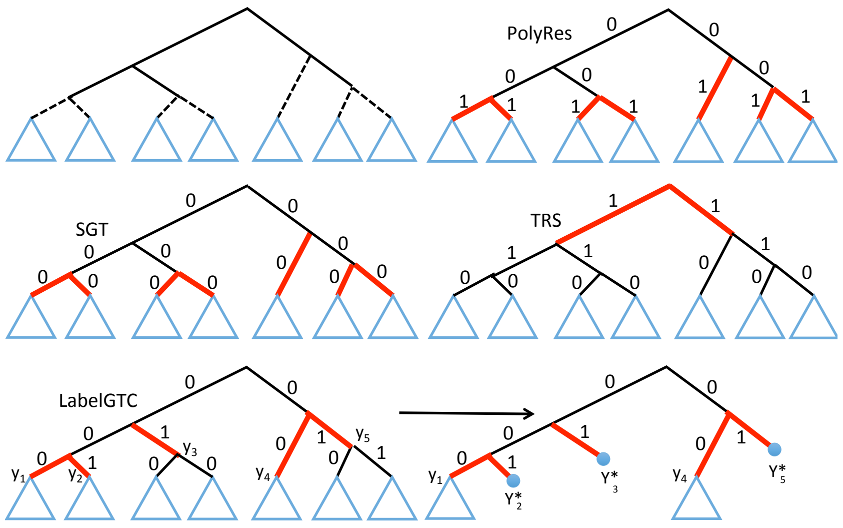


Figure 3 Top left. A gene tree T with a covering set $\mathcal{T}$ composed of 7 subtrees indicated as triangles. The set $E(T) \setminus E(\mathcal{T})$ contains 7 terminal edges (dotted lines) and 5 non-terminal edges (solid lines). Four 0-1 edge labelling corresponding to: **Top right.** the PolyRes problem ; **Middle left.** the SGT problem ; **Middle right.** the TRS problem ; **Bottom left.** a general input of the LabelGTC problem. In all four cases, the largest covering set of edges of $E(T)$ that have no ancestral edge labelled 1 are indicated with thicker red lines. **Bottom right.** The tree T^* built at step 2.b.) of the LabelGTC algorithm (Theorem 8) for the general input (Bottom left).

Proof. The problem LabelGTC on $\{T, \mathcal{L}(T)\}$ is reduced to:

1. M-PolyRes if $\mathcal{T} = \mathcal{L}(T)$, because there is a bijection between the edges of $E(T)$ labelled 1 and the set of nodes (excluding the root node) of the non-binary tree T^{nb} obtained from T by contracting all edges labelled 0 : an edge (x, y) of T labelled 1 corresponds to a node y' of T^{nb} such that $\mathcal{L}(y) = \mathcal{L}(y')$. So a tree T' preserves the edges of $E(T)$ labelled 1 iff it is a binary refinement of T^{nb}.
2. PolyRes if all non-terminal edges are labelled 0 and all terminal edges are labelled 1, because a supertree T' of $\mathcal{T}$ preserves the terminal edges of $E(T)$ iff for any tree $T_i \in \mathcal{T}$, $T'_{|\mathcal{L}(T_i)} = T_i$.
3. SGT if all non-terminal and terminal edges are labelled 0, because the set of edges to be conserved by LabelGTC is empty in this case.
4. TRS if all non-terminal edges are labelled 1 and all terminal edges are labelled 0, because any triplet (a, b, c) of genes belonging to three different trees of $\mathcal{T}$ can be associated to a non-terminal edge of $E(T)$ as follows: suppose that a and $y = lca(b, c)$ are separated w.l.o.g. and let x be the parent node of y, then the edge (x, y) of T is a non-terminal edge of $E(T)$ because b and c belong to two different trees of $\mathcal{T}$ and (a, b, c) belongs to $Trp(x, y)$ (Definition 2). Now, it suffices to notice that a supertree T' for $\mathcal{T}$ preserves a non-terminal edge (x, y) of T iff it preserves the topology of all triplets of $Trp(x, y)$. ◀

4 Relating gene tree correction problems to triplets

So far, the TRS Problem is the only one that was defined in terms of triplets. However, as a rooted tree is fully determined by the topology of its set of leaf triplets, TRS can be seen as a special case of a very natural general gene tree correction problem, that we formulate below.

We first need to introduce some definitions. Generalizing the notion used for the TRS problem, a *triplet of genes* is a triplet (a, b, c) of distinct genes of Γ. By convention and without loss of generality, we consider that a and $lca(b, c)$ are separated in T. Let T and T' be two trees for Γ and Trp be a set of triplets. We say that T' is *triplet respecting* for Trp as compared to T iff T' displays the same topology as T for each triplet of Trp. The general gene tree correction problem is formulated as follows.

TRIPLET-RESPECTING GENETREECORRECTION (TRIPLETGTC) PROBLEM:
Input: A species tree S, a gene tree T, a covering set of trees $\mathcal{T}$ for T and a set Trp of triplets;
Output: A supertree T' for $\mathcal{T}$ of minimum reconciliation cost respecting Trp.

The set Trp can be restricted to triplets with genes belonging to at least two different trees of $\mathcal{T}$, as the other triplets are necessarily displayed by a supertree for $\mathcal{T}$. We call TRIPLETGTC$_{leaves}$ the TRIPLETGTC problem in the special case where $\mathcal{T} = \mathcal{L}(T)$.

The following theorem makes the link between the LABELGTC problem and the TRIPLET-GTC problem. First, we formally define the set of triplets associated to an edge and that associated to a rooted subtree.

▶ **Definition 2.** Let (x, y) be an edge of $E(T)$. The set of triplets $Trp(x, y)$ contains all the triplets (a, b, c) such that a and y are separated and $lca(b, c) = y$.

▶ **Definition 3.** Let T_i be a subtree of T. The set of triplets $Trp(T_i)$ contains all the triplets (a, b, c) such that a, b and c are leaves of T_i.

For example, in the gene tree T depicted in Figure 1, if (x, y) is the non-terminal edge such that $\mathcal{L}(y) = \{d_2, d_3, e_2, e_3\}$, then $Trp(x, y) = \{(a, b, c) \,|\, a \in \Gamma \setminus \{d_2, d_3, e_2, e_3\} \, and \, (b, c) \in \{d_2, d_3\} \times \{e_2, e_3\}\}$, and $= Trp(T[y]) = \{(d_2, d_3, e_2), (d_2, d_3, e_3), (d_2, e_2, e_3), (d_3, e_2, e_3)\}$.

Note that a tree T' for $\mathcal{L}(T)$ preserves an edge (x, y) of $E(T) \setminus E(\mathcal{T})$ iff it preserves the topology of all triplets in $Trp(x, y)$. Similarly, it preserves the topology of a subtree T_i of T iff it preserves the topology of all triplets in $Trp(T_i)$.

▶ **Theorem 4.** *Let T be a 0-1 edge-labelled gene tree, $\mathcal{T}$ be a covering set for T and Trp be a set of triplets. Then, the* TRIPLETGTC *Problem is reduced to the* LABELGTC *Problem iff $Trp = \{Trp(x, y) \mid (x, y) \in E(T) \setminus E(\mathcal{T}) \, and \, l(x, y) = 1\}$.*

We now make the link between the various gene tree correction problems and the TRIPLETGTC$_{leaves}$ problem by analyzing the output of TRIPLETGTC$_{leaves}$ depending on the input set of triplets Trp. With respect to the set of initial subtrees $\mathcal{T}$, a triplet (a, b, c) of genes can either be included in a single subtree or distributed among two or three subtrees. We formally define these three possibilities of triplet-respecting trees in the next definition.

▶ **Definition 5.** Let T be a gene tree and $\mathcal{T}$ be a covering set for T. Each of the following sets of triplets contains all the triplets (a, b, c) where a, b and c are disjoint genes satisfying the corresponding property:

- $Trp1$: a, b, c all belong to the same tree of $\mathcal{T}$;
- $Trp2$: a, b, c belong to two different trees of $\mathcal{T}$;
- $Trp3$: a, b, c all belong to different trees of $\mathcal{T}$.

The following theorem extends the result of Theorem 4.

▶ **Theorem 6.** *Let T be a tree, $\mathcal{T}$ be a covering set of subtrees for T and Trp be a set of triplets. Then the* TRIPLETGTC$_{leaves}$ *Problem is reduced to:*

1. *the Identity if $Trp = Trp1 \cup Trp2 \cup Trp3$ (no modification of the input tree); Otherwise:*
2. *if $Trp = Trp1 \cup Trp2$, the* PolyRes *Problem;*
3. *if $Trp = Trp1 \cup Trp3$, the* TRS *Problem;*
4. *if $Trp = Trp2 \cup Trp3$, the* M-PolyRes *Problem with:*
   - **(a)** *all non-terminal and terminal edges labelled* 1 *and,*
   - **(b)** *all other edges (in $E(\mathcal{T})$) labelled* 0*;*
5. *if $Trp = Trp1$, the* SGT *Problem;*
6. *if $Trp = Trp2$, the* M-PolyRes *Problem with:*
   - **(a)** *all terminal edges labelled* 1*;*
   - **(b)** *all other edges (non-terminal and in $E(\mathcal{T})$) labelled* 0.
7. *if $Trp = Trp3$, the* M-PolyRes *Problem with:*
   - **(a)** *all non-terminal edges labelled* 1 *and,*
   - **(b)** *all other edges (terminal and in $E(\mathcal{T})$) labelled* 0.

5 Algorithm for the LabelGTC Problem

In this section, we describe an algorithm for the LabelGTC problem in the general case of a 0-1 edge labelling that does not correspond to a pre-defined gene tree correction method, as pointed out by Theorem 1. We will show later that it leads to a heuristic algorithm for the more general TripletGTC problem.

The idea behind the algorithm for reconstructing the new tree T' from the input tree T is the following. For any edge (x, y) in $E(T) \setminus E(\mathcal{T})$ such that $l(x, y) = 1$, by definition of the LabelGTC Problem, there exists a node y' of T' such that $\mathcal{L}(y') = \mathcal{L}(y)$. So the subtree $T'[y']$ of T' for the subset $\mathcal{L}(y)$ can first be constructed independently from the remaining of the tree, and then grafted at the appropriate location in a way minimizing the reconciliation cost. This leads to a recursive algorithm reconstructing and amalgamating iteratively, in a bottom-up order, the subtrees of T' for subsets $\mathcal{L}(y)$ corresponding to the edges (x, y) in $E(T) \setminus E(\mathcal{T})$ verifying $l(x, y) = 1$. The root $r(T)$ of T is associated to a dummy edge (s, r) such that $l(s, r) = 1$.

A *covering set of edges* for T is a set of separated edges $\mathcal{E} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ such that $\mathcal{E} \subseteq E(T) \setminus E(\mathcal{T})$ and $\cup_{i=1}^{n} \mathcal{L}(y_i) = \mathcal{L}(T)$.

Lemma 7 describes a property of covering sets that will be useful for a formal description of the algorithm solving LabelGTC.

▶ **Lemma 7.** *Let (x, y) be an edge in $E(T) \setminus E(\mathcal{T})$ such that $l(x, y) = 1$, and $\mathcal{E}$ be the largest covering set of edges of $E(T[y])$ that have no ancestral edge in $E(T[y])$ labelled* 1*. Then, any edge of $\mathcal{E}$ labelled* 0 *is a terminal edge.*

Given an edge (x, y) in $E(T) \setminus E(\mathcal{T})$ such that $l(x, y) = 1$, to compute the subtree $T'[y']$ of T' for $\mathcal{L}(y)$, first we look for the largest covering set of edges $\mathcal{E}$ for $T[y]$ such that any edge in $\mathcal{E}$ has no ancestral edge in $E(T[y])$ labelled 1 (see Figure 3 for illustration). Next, we distinguish two possible cases. If all edges in $\mathcal{E}$ are labelled 0, then $T'[y']$ can be obtained by applying the SGT algorithm [7] (designed as SGT in Algorithm 1) on the set of subtrees of $\mathcal{T}$ belonging to $T[y]$. Otherwise, $\mathcal{E}$ contains edges that are labelled 1. In this case, we compute each of the subtrees of $T'[y']$ corresponding to these edges labelled 1, yielding a set of subtrees $\mathbb{T}'$ and then we build $T'[y']$ using the SGT algorithm again with the constraint that the trees in $\mathbb{T}'$ should remain unmodified.

▶ **Theorem 8.** *Algorithm 1 solves the* LabelGTC *problem on an instance $\{S, T, \mathcal{T}, l\}$ in time $O(4^k.(n + 1)^k.k)$ where $n = |\Gamma|$ and $k = |\mathcal{T}|$.*

Algorithm 1 *LabelGTC*$(S, T, \mathcal{T}, l)$

```
$\mathcal{E} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ is the largest covering set of edges for $T$ in $E(T) \setminus E(\mathcal{T})$
that have no ancestral edge labelled 1.
(Stop condition)
if all edges of $\mathcal{E}$ are labelled 0 then
  return ($SGT(S, T, \mathcal{T})$)
end if
(Iterative step)
for $(x_i, y_i) \in \mathcal{E}$ such that $l(x_i, y_i) = 1$ do
  $T'[y'_i] = LabelGTC(S, T[y_i], \mathcal{T}_{|\mathcal{L}(y_i)}, l_{|E(T[y_i])})$;
end for
$T^*$ is obtained from $T$ by contracting each subtree $T[y_i]$ such that $(x_i, y_i) \in \mathcal{E}$ and
$l(x_i, y_i) = 1$ to a single leaf node $y^*_i$;
$\mathcal{T}^*$ is the set of separated subtrees $\{T[y_i] : (x_i, y_i) \in \mathcal{E}$ and $l(x_i, y_i) = 0\} \cup \{y^*_i : (x_i, y_i) \in$
$\mathcal{E}$ and $l(x_i, y_i) = 1\}$ of $T^*$;
$T'^* = SGT(S, T^*, \mathcal{T}^*)$;
return (the tree obtained from $T'^*$ by replacing each leaf node $y^*_i$ by $T'[y'_i]$);
```

Heuristic algorithm for TripletGTC

A natural heuristic algorithm for the TripletGTC problem on a gene tree T with a covering set $\mathcal{T}$ and a set of triplet Trp consists in first giving the label 1 to any edge (x, y) of $E(T) \setminus E(\mathcal{T})$ such that there exists a triplet (a, b, c) in Trp belonging to the set $Trp(x, y)$. Next the LabelGTC algorithm is applied to the obtained edge labelled tree. The corrected tree resulting from this algorithm will preserve all triplets of Trp, but more largely all triplets of the set $\{Trp(x, y) \mid (x, y) \in E(T) \setminus E(\mathcal{T})$ *and* $l(x, y) = 1\}$, which includes Trp.

6 Accounting for the 0-1 edge labelling in partial subtrees

In the formulation of the LabelGTC problem, the 0-1 edge labels are ignored for the trees of $\mathcal{T}$, and only edges in $E(T) \setminus E(\mathcal{T})$ labelled 1 have to be preserved. A natural extension would be to preserve also the edges of $\mathcal{T}$ labelled 1.

This can be done by mean of Algorithm 1, but replacing the call to the SGT algorithm by an algorithm solving the following problem.

Label SuperGenetree (LabelSGT) Problem:
Input: A species tree S, a gene tree T, a covering set of trees $\mathcal{T}$ for T and a 0-1 edge labelling l for T.
Output: A supertree T' for $\mathcal{T}$ of minimum reconciliation cost such that: if $(x, y) \in E(\mathcal{T})$ is such that $l(x, y) = 1$, then there is an edge (x', y') in $E(T')$ such that $\mathcal{L}(y) = \mathcal{L}(y')$.

Before describing an algorithm solving the above LabelSGT problem, we first recall some useful definitions and the algorithm described in [7] for the SGT problem.

Given a gene tree T and a species tree S, *cost*(T) denotes the reconciliation cost of T with S. If the root x of T has two children node, one of the subtree of T rooted at a child of x is arbitrarily denoted by T_l and the other one by T_r. Given a set of gene subtrees $\{t_1, \ldots, t_k\}$ and a bipartition (L_l, L_r) of $\bigcup_{i=1}^{k} \mathcal{L}(t_i)$, if T'_l and T'_r are two trees for L_l and L_r respectively, then (T_l, T_r) denotes the tree T such that $T_l = T'_l$ and $T_r = T'_r$. In this case, *cost*(L_l, L_r)

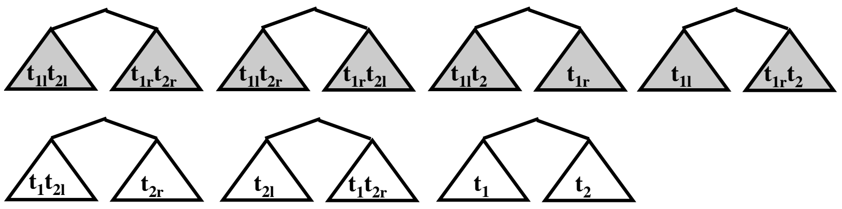


Figure 4 The 7 bipartitions of a set $\mathcal{B}(t_1, t_2)$. The first 4 bipartitions are those in which $\mathcal{L}(t_{1_l})$ and $\mathcal{L}(t_{1_r})$ are separated.

denotes the local reconciliation cost at the root x of the tree (T_l, T_r) counting the number of losses on the two edges linking x to T_l and T_l, plus 1 if x is a duplication node.

▶ **Definition 9** (Reformulation of Property 1 from [7]). Given a set of gene subtrees $\{t_1, \ldots, t_k\}$, $\mathcal{B}(t_1, \ldots, t_k)$ denotes the set of bipartitions (L_l, L_r) of $\bigcup_{i=1}^{k} \mathcal{L}(t_i)$ such that each subtree $t_i, 1 \leq i \leq k$, satisfies either: 1) $\mathcal{L}(t_i) \subseteq L_l$; or 2) $\mathcal{L}(t_i) \subseteq L_r$; or 3) $\mathcal{L}(t_{i_l}) \subseteq L_l$ and $\mathcal{L}(t_{i_r}) \subseteq L_r$; or 4) $\mathcal{L}(t_{i_l}) \subseteq L_r$ and $\mathcal{L}(t_{i_r}) \subseteq L_l$.

$\mathcal{B}(t_1, \ldots, t_k)$ contains exactly $\frac{4^k}{2} - 1$ bipartitions. For example, for $k = 2$ subtrees, the 7 bipartitions are depicted in Figure 4.

The following is the recurrence formulae of the dynamic programming algorithm described in [7] for the SGT problem.

▶ **Lemma 10** (Reformulation of Lemma 3 from [7]). *The following algorithm solves the* SGT *problem on an instance $\{S, T, \mathcal{T}\}$ such that $\mathcal{T} = \{t_1, \ldots, t_k\}$ in time $O(4^k.(n+1)^k.k)$ where $n = |\bigcup_{i=1}^{k} \mathcal{L}(t_i)|$.*

1. (Stop condition) *If $|\bigcup_{i=1}^{k} \mathcal{L}(t_i)| = 1$, then $SGT(t_1, \ldots, t_k)$ is the gene tree composed of the corresponding single node;*
2. *Otherwise, $SGT(t_1, \ldots, t_k) = (T'_l, T'_r)$ where $T'_l = SGT(t_{1|L_l}, \ldots, t_{k|L_l})$ and $T'_r = SGT(t_{1|L_r}, \ldots, t_{k|L_r})$ such that:*

$$(L_l, L_r) = \underset{(L_l,L_r)\in\mathcal{B}(t_1,\ldots,t_k)}{\text{argmin}} \{cost(L_l, L_r) + cost(T'_l) + cost(T'_r)\,.\}$$

Let T be a 0-1 edge-labelled gene tree, and $\mathcal{T}$ be a covering set of subtrees for T. In order to preserve the edges in $\mathcal{T}$ labelled 1, it suffices to consider, at each step of the above algorithm, only the bipartitions that do not separate $\mathcal{L}(t_{i_l})$ and $\mathcal{L}(t_{i_r})$ for any subtree $t_i, 1 \leq i \leq k$ such that $t_i = T_j[y]$ and (x, y) is an edge of $E(T_j)$ labelled 1, unless $k = 1$. For example in Figure 4, if t_1 is such a subtree, then the four first bipartitions that separate $\mathcal{L}(T_{1_l})$ and $\mathcal{L}(T_{1_r})$ are discarded.

Given a set of gene subtrees $\{t_1, \ldots, t_k\}$ of $\mathcal{T}$, we define $\mathcal{B}_{Label}(t_1, \ldots, t_k)$ as the subset of $\mathcal{B}(t_1, \ldots, t_k)$ containing all bipartitions that do not separate any subtree $t_i, 1 \leq i \leq k$ such that $t_i = T_j[y]$, T_j is a tree of T and (x, y) is an edge of $E(T_j)$ labelled 1.

▶ **Theorem 11.** *The algorithm described in Lemma 10 in which any set $\mathcal{B}(t_1, \ldots, t_k)$ is replaced by the set $\mathcal{B}_{Label}(t_1, \ldots, t_k)$ solves the* LabelSGT *problem on $\mathcal{T}$ with the same time complexity as the initial algorithm, i.e. in time $O(4^k.(n+1)^k.k)$.*

7 Conclusion

This paper provides a unifying view allowing to reconcile apparently heterogeneous gene tree correction methods. The general LabelGTC and TripletGTC approaches have the advantage of not being dependent upon particular assumptions on trees. We present the first general algorithm allowing to correct a tree according to an arbitrary 0-1 edge-label, or more generally to an arbitrary set of triplets whose topology should be preserved. These algorithms have the same exponential time-complexity as the one previously developed for the SGT problem [7], which has been proved NP-hard for the duplication distance. Although no proof of complexity for the more general reconciliation distance exists, it is unlikely that adding losses makes the problems more tractable. We conjecture the SGT problem, and more generally the LabelGTC and TripletGTC problems, remain NP-hard in this case.

The 0-1 edge-labelling considered in this paper can be seen as a first step towards integrating knowledge on edge statistical support in a gene tree correction algorithm. Generalizing the edge-labelling function l to an arbitrary domain would require a complete reformulation of the problems. It may be more intuitive in this case to use a heuristic algorithm exploring a tree space around the input tree, and among statistically equivalent trees, take the one minimizing a combination of values accounting for both sequence alignment cost and reconciliation cost. Several such methods using species tree information in addition to sequence information, have been developed (e.g. TreeBeST [13], TreeFix [16], PhylDog [1], SPIMAP [11], ProfilNJ [10]). However, a formal conceptual framework, as the one developed in this paper, remains to be developed for the gene tree correction problem with a non-binary edge-labelling function.

References

1 B. Boussau, G.J. Szöllősi, L. Duret, M. Gouy, E. Tannier., and V. Daubin. Genome-scale coestimation of species and gene trees. *Genome Research*, 23:323-330, 2013.

2 K. Chen, D. Durand, and M. Farach-Colton. Notung: Dating gene duplications using gene family trees. *Journal of Computational Biology*, 7:429–447, 2000.

3 J. Felsenstein. Phylogenies from molecular sequences: Inference and reliability. *Ann. Review Genet.*, 22:521–565, 1988.

4 E. Jacox, C. Chauve, G.J. Szollosi, Y. Ponty, and C. Scornavacca. ecceTERA: comprehensive gene tree-species tree reconciliation using parsimony. *Bioinformatics*, 32(13):2056-2058, 2016.

5 M. Lafond, E. Noutahi, and N. El-Mabrouk. Efficient non-binary gene tree resolution with weighted reconciliation cost. In *Combinatorial Pattern Matching, LIPIcs-Leibniz International Proceedings in Informatics*, volume 54. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

6 M. Lafond, A. Ouangraoua, and N. El-Mabrouk. Reconstructing a supergenetree minimizing reconciliation. *BMC-Genomics*, 16:S4, 2015. Special issue of RECOMB-CG 2015.

7 N. El-Mabrouk M. Lafond, C. Chauve and A. Ouangraoua. Gene tree construction and correction using supertree and reconciliation. In *Asia Pacific Bioinformatics Conference*, 2017. soon in IEEE/ACM TCBB.

8 N. Nguyen, S. Mirarab, and T. Warnow. MRL and SuperFine+MRL: new supertree methods. *J. Algo. for Mol. Biol.*, 7(3), 2012.

9 Thi Hau Nguyen, Vincent Ranwez, Stéphanie Pointet, Anne-Muriel Arigon Chifolleau, Jean-Philippe Doyon, and Vincent Berry. Reconciliation and local gene tree rearrangement can be of mutual profit. *Algorithms Mol Biol*, 8(1):12, 2013. `doi:10.1186/1748-7188-8-12`.

10 E. Noutahi, M. Semeria, M. Lafond, J. Seguin, L. Gueguen, N. El-Mabrouk, and E. Tannier. Efficient gene tree correction guided by genome evolution. *Plos.One*, 11(8), 2016.

11 M. D. Rasmussen and M. Kellis. A bayesian approach for fast and accurate gene tree reconstruction. *Molecular Biology and Evolution*, 28(1):273- 290, 2011.

12 Jamal S. M. Sabir, Robert K. Jansen, Dhivya Arasappan, Virginie Calderon, Emmanuel Noutahi, Chunfang Zheng, Seongjun Park, Meshaal J. Sabir, Mohammed N. Baeshen, Nahid H. Hajrah, Mohammad A. Khiyami, Nabih A. Baeshen, Abdullah Y. Obaid, Abdulrahman L. Al-Malki, David Sankoff, Nadia El-Mabrouk, and Tracey A. Ruhlman. The nuclear genome of Rhazya stricta and the evolution of alkaloid diversity in a medically relevant clade of apocynaceae. *Nature Scientific Reports*, 6(33782), 2016.

13 F. Schreiber, M. Patricio, M. Muffato, M. Pignatelli, and A. Bateman. Treefam v9: a new website, more species and orthology-on-the-fly. *Nucleic Acids Research*, 2013. doi:10.1093/nar/gkt1055.

14 C. Scornavacca, L. van Iersel, S. Kelk, and D. Bryant. The agreement problem for unrooted phylogenetic trees is FPT. *Journal of Graph Algorithms and Applications*, 18(3):385 - 392, 2014.

15 A. J. Vilella, J. Severin, A. Ureta-Vidal, L. Heng, R. Durbin, and E. Birney. EnsemblCompara gene trees: Complete, duplication-aware phylogenetic trees in vertebrates. *Genome Research*, 19:327-335, 2009.

16 Y. C. Wu, M. D. Rasmussen, M. S. Bansal, and M. Kellis. TreeFix: Statistically informed gene tree error correction using species trees. *Systematic Biology*, 62(1):110- 120, 2013.

17 Y. Zheng and L. Zhang. Reconciliation with non-binary gene trees revisited. In *Lecture Notes in Computer Science*, volume 8394, pages 418-432, 2014. Proceedings of RECOMB.

A Proofs

Proof of Theorem 4. Let T be a 0-1 edge-labelled gene tree, $\mathcal{T}$ be a covering set of subtrees for T and Trp be a set of triplets. If $Trp = \bigcup\{Trp(x,y) | (x,y) \in E(T) \setminus E(\mathcal{T}) and l(x,y) = 1\}$, then the TripletGTC problem on $\{T, \mathcal{T}, Trp\}$ is reduced to LabelGTC because a supertree T' of $\mathcal{T}$ preserves an edge (x,y) of $E(T) \setminus E(\mathcal{T})$ iff it preserves the topology of all triplets in $Trp(x,y)$. ◀

Proof of Theorem 6. The proof follows directly from the following three equalities:

1. $Trp1 = \bigcup\{Trp(T_i) \mid T_i \in \mathcal{T}\}$;
2. $Trp2 = \{Trp(x,y) \mid (x,y)\ is\ a\ terminal\ edge\ of\ T\}$;
3. $Trp3 = \{Trp(x,y) \mid (x,y)\ is\ a\ non\ terminal\ edge\ of\ T\}$.

So a tree T' for $\mathcal{L}(T)$ preserves the topology of all trees $T_i \in \mathcal{T}$ iff it preserves $Trp1$; It preserves all terminal edges of $E(T)$ iff it preserves $Trp2$ and it preserves all non-terminal edges of $E(T)$ iff $Trp3$. The combination of the inclusion or exclusion of $Trp1$, $Trp2$ and $Trp3$ in Trp (except the case where $Trp = \emptyset$) results in the 7 cases of the theorem. ◀

Proof of Lemma 7. Let (x,y) be an edge in $E(T) \setminus E(\mathcal{T})$ such that $l(x,y) = 1$, and $\mathcal{E}$ be the largest covering set of edges of $E(T[y])$ that have no ancestral edges in $E(T[y])$ labelled 1. Suppose that $\mathcal{E}$ contains an edge (s,t) labelled 0 that is not a terminal edge and denote by t_l and t_r the two children of the node t. Then the edges (t,t_l) and (t,t_r) belong to $E(T) \setminus E(\mathcal{T})$ and satisfy the condition that they have no ancestral edges in $E(T[y])$ labelled 1. So $(\mathcal{E} \cup \{(t,t_l),(t,t_r)\}) \setminus \{(s,t)\}$ is also a covering set of edges of $E(T[y])$ that have no ancestral edges in $E(T[y])$ labelled 1, which contradicts the assumption that $\mathcal{E}$ is the largest such covering set of edges. ◀

Proof of Theorem 8. Let T be a 0-1 edge-labelled gene tree, $\mathcal{T}$ be a covering set of subtrees for T and $\mathcal{E} = \{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$ be the largest covering set of edges for T that have no ancestral edge labelled 1.

1. If all edges of $\mathcal{E}$ are labelled 0, then by Lemma 7, all the edges of $\mathcal{E}$ are terminal edges. Since $\mathcal{E}$ covers T, then all edges of $E(T) \setminus E(\mathcal{T})$ are labelled 0. So, by Theorem 1, the problem is reduced to SGT.
2. Otherwise, let $T' = LabelGTC(S, T, \mathcal{T}, l)$, and for any edge (x_i, y_i) of $\mathcal{E}$ labelled 1 and preserved in T', let y_i' be the node of T' such that $\mathcal{L}(y_i') = \mathcal{L}(y_i)$. Since $T_{|\mathcal{L}(y_i)}$ is a complete subtree of T, then $T'[y_i']$ is also a complete subtree of T' that can be constructed independently of the remaining of the tree T' as $T'[y_i'] = LabelGTC(S, T[y_i], \mathcal{T}_{|\mathcal{L}(y_i)}, l_{|E(T[y_i])})$ (computed at Step 2.a. of the algorithm). Next, the obtained subtrees must be adequately grafted on branches of a supergenetree of the remaining trees $\{T[y_i] \in \mathcal{T} \mid (x_i, y_i) \in \mathcal{E}\ and\ l(x_i, y_i) = 0\}$. In this case, the problem is also reduced to the SGT problem on $\mathcal{T}^* = \{T[y_i] : (x_i, y_i) \in \mathcal{E}\ and\ l(x_i, y_i) = 0\} \cup \{y_i^* : (x_i, y_i) \in \mathcal{E}\ and\ l(x_i, y_i) = 0\}$ where each tree $T'[y_i']$ computed at Step 2.a. is contracted into a single leaf node y_i^* associated to the leafset $\mathcal{L}(T'[y_i'])$ (computed at Steps 2.b. to 2.e. of the algorithm).

The worst case of the algorithm is the stop case where it is directly reduced to the SGT algorithm whose time complexity is in $O(4^k.(n+1)^k.k)$. ◀

Proof of Theorem 11. The algorithm described in Lemma 10 solves the SGT problem by exploring the set of all possible supergenetrees for $\mathcal{T}$. The set of supergenetrees is explored by considering all possible bipartitions (L_l, L_r) for each set $\bigcup_{i=1}^{k} \mathcal{L}(t_i)$ where each t_i is a subtree of a tree of $\mathcal{T}$. So, replacing any set $\mathcal{B}(t_1, \ldots, t_k)$ by the set $\mathcal{B}_{Label}(t_1, \ldots, t_k)$ will only discard the bipartitions (L_l, L_r) that prevent the preservation of an edge of $E(\mathcal{T})$ labelled 1. So the modification of the algorithm will return a supergenetree for $\mathcal{T}$ of minimum reconciliation cost preserving all edges of $E(\mathcal{T})$ labelled 1. The time complexity of the modified algorithm remains the same as that of the initial algorithm. ◀

Towards Distance-Based Phylogenetic Inference in Average-Case Linear-Time[*]

Maxime Crochemore[1], Alexandre P. Francisco[2], Solon P. Pissis[3], and Cátia Vaz[4]

1 **Department of Informatics, King's College London, London, UK**
2 **INESC-ID and Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal**
3 **Department of Informatics, King's College London, London, UK**
4 **INESC-ID and Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Lisbon, Portugal**

Abstract

Computing genetic evolution distances among a set of taxa dominates the running time of many phylogenetic inference methods. Most of genetic evolution distance definitions rely, even if indirectly, on computing the pairwise Hamming distance among sequences or profiles. We propose here an average-case linear-time algorithm to compute pairwise Hamming distances among a set of taxa under a given Hamming distance threshold. This article includes both a theoretical analysis and extensive experimental results concerning the proposed algorithm. We further show how this algorithm can be successfully integrated into a well known phylogenetic inference method.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases computational biology, phylogenetic inference, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.9

1 Introduction

The evolutionary relationships between different species or *taxa* are usually inferred through known phylogenetic analysis techniques. Some of these techniques rely on the inference of phylogenetic trees, which can be computed from molecular sequences or from profiles built by sequencing specific regions, *e.g.*, housekeeping genes for a given species. Phylogenetic trees are also used in other contexts, such as to understand the evolutionary history of gene families, to allow phylogenetic foot-printing, to trace the origin and transmission of infectious diseases, or to study the co-evolution of hosts and parasites [11, 23].

In most cases, the process of phylogenetic inference starts with a multiple alignment of the sequences under study; and then tree-building methods are used. These methods rely on some distance-based analysis of sequences or profiles [24].

Distance-based methods for phylogenetic analysis rely on a measure of genetic evolution distance, which is often defined directly or indirectly from the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches. A first step of these methods is to compute this distance between all pairs of sequences. The simplest approach

* This work was partly supported by the Royal Society International Exchanges Scheme, and by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects BacGenTrack (TUBITACK/0004/2014), PRECISE (SAICTPAC/0021/2015) and UID/CEC/500021/2013.

is to use the Hamming distance, also known as observed p-distance, defined as the number of positions at which two aligned sequences differ. Note that the Hamming distance between two sequences underestimates their true evolutionary distance and, thus, a correction formula based on some model of evolution is often used [11, 24]. Although distance-based methods not always produce the best tree for the data, usually they also incorporate an optimality criterion into the distance model for getting more plausible phylogenetic reconstructions, such as the minimum evolution criterion [5], the least squares criterion [22] or the clonal complexes expansion and diversification [7].

Most of the distance-based methods are agglomerative methods. They start with each sequence being a singleton cluster and, at each step, they join two clusters. The iterative process stops when all sequences are part of a single cluster. A phylogenetic tree is obtained within this process. At each step the candidate pair is selected taking into account the distance among clusters as well as the optimality criterion chosen to adjust it.

The computation of a distance matrix (2D array containing the pairwise distances between the elements of a set) is a common first step for distance-based methods, such as eBURST [8], goeBURST [9], Neighbor Joining [25] and UPGMA [26]. This particular step dominates the running time of most methods, taking $\Theta(md^2)$ time in general, d being the number of sequences or profiles and m the length of each sequence or profile. For large-scale datasets this running time may be quite problematic.

However, depending on the underlying model of evolution and on the optimality criterion, it may not be strictly necessary to be aware of the complete distance matrix. There are methods that continue to provide optimal solutions without a complete matrix. For such methods, one may still consider a truncated distance matrix and several heuristics, combined with final local searches through topology rearrangements, to improve the running time [22]. The goeBURST, our use case in this article, is an example of a method that can work with truncated distance matrices by construction, *i.e.*, one needs only to know which pairs are at Hamming distance at most k.

Our results. We propose here an average-case $\mathcal{O}(md)$-time and $\mathcal{O}(md)$-space algorithm to compute the pairs of sequences, among d sequences of length m, that are at distance at most k, when $k < \frac{(m-k-1)\cdot\log\sigma}{\log md}$, where σ is the size of the sequences alphabet. We support our result with both a theoretical analysis and an experimental evaluation on synthetic and real datasets of different data types (MLST, cgMLST, wgMLST and SNP). We further show that our method improves goeBURST.

Structure of the article. We describe and analyze the proposed algorithm in Section 2. The goeBURST use case is presented in Section 3. The experimental evaluation using both synthetic and real datasets is presented in Section 4.

2 Closest pairs in linear time

Let P be the set of profiles (or sequences) each of length m, defined over an integer alphabet Σ, (*i.e.*, $\Sigma = \{1, \ldots, m^{O(1)}\}$), with $d = |P|$ and $\sigma = |\Sigma|$. Let also $H : P \times P \to \{0, \ldots, m\}$ be the function such that $H(u, v)$ is the Hamming distance between profiles $u, v \in P$. Given an integer threshold $0 < k < m$, the problem is to compute all pairs $u, v \in P$ such that $H(u, v) \leq k$, and the corresponding $H(u, v)$ value, faster than the $\Theta(md^2)$ time required to compute naïvely the complete distance matrix for the d profiles of length m.

Table 1 Data structures used in our approach for each step.

Profile indexing	Candidate profile pairs enumeration	Pairs verification
Suffix array	Binary search	Naïve
	LCP based clusters	RMQ_{LCP}

We address this problem by indexing all profiles P using the suffix array (denoted by SA) and the longest common prefix (denoted by LCP) array [16]. We rely also on a range minimum queries (RMQ) data structure [1, 2] over the LCP array (denoted by RMQ_{LCP}). The problem is then solved in three main steps:

1. Index all profiles using the SA data structure.
2. Enumerate all candidate profile pairs given the maximum Hamming distance k.
3. Verify each candidate profile pair by checking if the associated Hamming distance is no more than k.

Table 1 summarizes the data structures and strategies followed in each step. Profiles are concatenated and indexed using SA. Depending on the strategy to be used, we further process the SA and build the LCP array and pre-process it for fast RMQ. This allows for enumerating candidate profile pairs and computing distances faster.

In what follows, we detail the above steps and show how the data structures are used to improve the overall running time.

2.1 Step 1: Profile indexing

Profiles are concatenated and indexed in an SA in $\mathcal{O}(md)$ time and space [12, 14]. Let us denote this string by s. Since we only need to compute the distances between profiles that are at Hamming distance at most k, we can conceptually split each profile into k non-overlapping *blocks* of length $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$ each. It is then folklore knowledge that if two profiles are within distance k, they must share at least one such block of length $\mathcal{L}$. Our approach is based on using the SA of s to efficiently identify matching blocks among profile pairs. This lets us quickly filter in candidate profile pairs and filter out the ones that can never be part of the output.

2.2 Step 2: Candidate profile pairs enumeration

The candidate profile pairs enumeration step provides the pairs of profiles that do not differ in more than k positions, but it may include spurious pairs. Since SA is an ordered structure, a simple solution is to use a binary search approach. For each block of each profile, we can obtain in $\mathcal{O}(\mathcal{L}\log\ n)$ time, where $n = md$, all the suffixes that have that block as a prefix. If a given match is not aligned with the initial block, *i.e.* it does not occur at the same position in the respective profile, then it should be discarded. Otherwise, a candidate profile pair is reported. This searching procedure is done in $\mathcal{O}(dk\mathcal{L}\log n) = \mathcal{O}(n \log n)$ time.

Another solution relies on computing the LCP array: the longest common prefix between each pair of consecutive elements within the SA. This information can also be computed in $\mathcal{O}(n)$ time and space [13]. Since SA is an ordered structure, for the contiguous suffixes s_i, s_{i+1}, s_{i+2} of s, with $0 \leq i < n-2$, we have that the common prefix between s_i and s_{i+1} is at least as long as the common prefix of s_i and s_{i+2}. By construction, it is possible to get the position of each suffix in the corresponding profile in constant time. Then, we cluster the corresponding profiles of contiguous pairs if they have an LCP value greater than or equal to $\mathcal{L}$ *and* they are also aligned. This clustering procedure can be done in $\mathcal{O}(kd^2)$ time.

2.3 Step 3: Pairs verification

After getting the set of candidate profile pairs, a naïve solution would be to compute the distance for each pair of profiles by comparing them in linear time, *i.e.*, $\mathcal{O}(m)$ time. However, if we compute the LCP array of s, we can then perform a sequence of $\mathcal{O}(k)$ RMQ over the LCP array for checking if a pair of profiles is at distance at most k. These RMQ over the LCP array correspond to longest common prefix queries between a pair of suffixes of s. Since after a linear-time pre-processing over the LCP array, RMQ can be answered in constant time per query [1], we obtain a faster approach for computing the distances. This alternative approach takes $\mathcal{O}(k)$ time to verify each candidate profile pair instead of $\mathcal{O}(m)$ time.

2.4 Average-case analysis

Algorithm 1 below details the solution based on LCP clusters; and Theorem 1 shows that this algorithm runs in linear time on average using linear space. We rely here on well-known results concerning the linear-time construction of the SA [12, 14] and the LCP array [13], as well as the linear-time pre-processing for the RMQ data structure [2].

In what follows, LCP$[i]$, $i > 0$, stores the length of the longest common prefix of suffixes s_{i-1} and s_i of s, and $\text{RMQ}_{\text{LCP}}(i, j)$ returns the index of the smallest element in the subarray LCP$[i \dots j]$ in constant time [2]. We rely also on some auxiliary subroutines; let $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$:

Aligned(i). Let $\ell = i \mod m$, *i.e.*, the starting position of the suffix s_i within a profile. Then this subroutine returns $\ell/\mathcal{L}$ if ℓ is multiple of $\mathcal{L}$, and -1 otherwise.

HD(p_i, p_j, ℓ). Given two profiles p_i and p_j which share a substring of length $\mathcal{L}$, starting at index $\ell\mathcal{L}$, this subroutine computes the minimum of k and the Hamming distance between p_i and p_j. This subroutine relies on RMQ_{LCP} to find matches between p_i and p_j and, hence, it runs in $\mathcal{O}(k)$ time since it can terminate after k mismatches.

▶ **Theorem 1.** *Given d profiles of length m each over an integer alphabet Σ of size $\sigma > 1$ with the letters of the profiles being independent and identically distributed random variables uniformly distributed over Σ, and the maximum Hamming distance $0 < k < m$, Algorithm 1 runs in $\mathcal{O}(md)$ average-case time and space if*

$$k < \frac{(m-k-1) \cdot \log \sigma}{\log md}.$$

Proof. Let us denote by s the string of length md obtained after concatenating the d profiles. The time and space required for constructing the SA and the LCP arrays for s and the RMQ data structure over the LCP array is $\mathcal{O}(md)$.

Let us denote by $\mathcal{B}$ the total number of blocks over s and by $\mathcal{L}$ the block length. We set $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$ and thus we have that $\mathcal{B} = d\lfloor \frac{m}{\mathcal{L}} \rfloor$. Let us also denote by C a maximal set of indices over x satisfying the following:

1. the length of the longest common prefix between any two suffixes of s starting at these indices is at least $\mathcal{L}$;
2. both of these suffixes start at the starting position of a block;
3. and both indices correspond to the starting position of the ith block in their profiles.

This can be done in $\mathcal{O}(md)$ time using the LCP array (lines 7–17). Processing all such sets C (lines 21–27) requires total time

$$\mathsf{PROC}_{i,j} \times \mathit{Pairs}$$

Algorithm 1: Algorithm using LCP clusters.

```
1  Input: A set $P$ of $d$ profiles of length $m$ each; an integer threshold $0 < k < m$.
2  Output: The set $X$ of distinct pairs of profiles that are at Hamming distance at
     most $k$, i.e., $X = \{(u,v) \in P \times P \mid u < v$ and $H(u,v) \leq k\}$.
3  Initialization: Let $s = s[0 \dots n-1]$ be the string of length $n = md$ obtained after
     concatenating the $d$ profiles, and $\mathcal{L} = \lfloor \frac{m}{k+1} \rfloor$. Construct the SA $\mathcal{S}$ for $s$, the LCP
     array for $s$ and $\mathrm{RMQ}_{\mathrm{LCP}}$. Initialize a hash table $H$ to track verified pairs.
4  Candidate pairs enumeration:
5  $X := \emptyset$; $\ell_p := -1$; $C_t := \emptyset$, for $0 \leq t \leq k$
6  foreach $1 \leq i < n$ do
7      $\ell := \mathrm{LCP}[i]$
8      if $\ell \geq \mathcal{L}$ then
9          $p_i := \lfloor \mathcal{S}[i]/m \rfloor$
10         $x :=$ Aligned($i$)
11         if $x \neq -1$ then
12             $C_x := C_x \cup \{p_i\}$
13         if $\ell_p = -1$ then
14             $p_{i-1} := \lfloor \mathcal{S}[i-1]/m \rfloor$
15             $x :=$ Aligned($i-1$)
16             if $x \neq -1$ then
17                 $C_x := C_x \cup \{p_{i-1}\}$
18         $\ell_p := \ell$
19     else if $\ell_p \neq -1$ then
20         Pairs enumeration:
21         foreach $C_t$, with $0 \leq t \leq k$ do
22             foreach $(p,q) \in C_t \times C_t : p < q$ do
23                 if $(p,q) \notin H$ then
24                     $H := H \cup \{(p,q)\}$
25                     $\delta :=$ HD($p$, $q$, $t$)
26                     if $\delta \leq k$ then
27                         $X := X \cup \{(p,q)\}$
28         $\ell_p := -1$; $C_t := \emptyset$, for $0 \leq t \leq k$
29 Finalize: Return the set $X$.
```

where $\mathsf{PROC}_{i,j}$ is the time required to process a pair i,j of elements of a set C, and *Pairs* is the sum of $|C|^2$ over all such sets C. We have that $\mathsf{PROC}_{i,j} = \mathcal{O}(k)$ by using RMQ over the LCP array. Additionally, by the stated assumption on the d profiles, the expected value for *Pairs* is no more than $\frac{\mathcal{B}d}{\sigma^{\mathcal{L}}}$: we have $\mathcal{B}$ blocks in total and each block can only match at most d other blocks by the conditions above. Hence, the algorithm requires on average the following running time

$$\mathcal{O}(md + k \cdot \frac{\mathcal{B}d}{\sigma^{\mathcal{L}}}).$$

Let us analyze this further to obtain the relevant condition on k. We have the following:

$$k \cdot \frac{\mathcal{B}d}{\sigma^{\mathcal{L}}} = \frac{k \cdot \lfloor \frac{m}{\lfloor m/(k+1) \rfloor} \rfloor \cdot d^2}{\sigma^{\lfloor \frac{m}{k+1} \rfloor}} \leq \frac{k \cdot (\frac{m}{\lfloor m/(k+1) \rfloor}) \cdot d^2}{\sigma^{\frac{m}{k+1}-1}}.$$

Since $0 < k < m$ by hypothesis, we have the following:

$$\frac{k \cdot (\frac{m}{\lfloor m/(k+1) \rfloor}) \cdot d^2}{\sigma^{\frac{m}{k+1}-1}} \leq \frac{(md)^2}{\sigma^{\frac{m}{k+1}-1}}.$$

By some simple rearrangements we have that:

$$\frac{(md)^2}{\sigma^{\frac{m}{k+1}-1}} = \frac{(md)^2}{(md)^{\frac{\log \sigma}{\log md}(\frac{m}{k+1}-1)}} = (md)^{2-\frac{(m-k-1)\log \sigma}{(k+1)\log md}}.$$

Consequently, in the case when

$$k < \frac{(m-k-1) \cdot \log \sigma}{\log md}$$

the algorithm requires $\mathcal{O}(md)$ time on average. The extra space usage is clearly $\mathcal{O}(md)$. ◀

3 Use case: goeBURST algorithm

The distance matrix computation is a main step in distance-based methods for phylogenetic inference. This step dominates the running time of most methods, taking $\Theta(md^2)$ time, for d sequences of length m, since it must compute the distance among all sequence pairs. But for some methods, or when we are only interested in local phylogenies for sequences or profiles of interest, one does not need to know all pairwise distances for reconstructing a phylogenetic tree. The problem addressed in this article was motivated by the goeBURST algorithm [9], our use case. goeBURST is one of such methods for which one must know only the pairs of sequences that are at Hamming distance at most k. The solution proposed here can however be extended to other distance-based phylogenetic inference methods, that rely directly or indirectly on Hamming distance computations. Note that most methods either consider the Hamming distance or its correction accordingly to some formula based on some model of evolution [11, 24]. In both cases we must start by computing the Hamming distance among sequences, but not necessarily all of them [22].

The underlying model of goeBURST is as follows: a given genotype increases in frequency in the population as a consequence of a fitness advantage or of random genetic drift, becoming a founder clone in the population; and this increase is accompanied by a gradual diversification of that genotype, by mutation and recombination, forming a cluster of phylogenetic closely-related strains. This diversification of the "founding" genotype is reflected in the appearance of genetic profiles differing only in one housekeeping gene sequence from this genotype – single locus variants (SLVs). Further diversification of those SLVs will result in the appearance of variations of the original genotype with more than one difference in the allelic profile, *e.g.*, double and triple locus variants (DLVs and TLVs).

The problem solved by goeBURST can be stated as a graphic matroid optimization problem and, hence, it follows a classic greedy approach [21]. Given the maximum Hamming distance k, we can define a graph $G = (V, E)$, where $V = P$ (set of profiles) and $E = \{(u, v) \in V^2 \mid H(u, v) \leq k\}$. The main goal of goeBURST is then to compute a minimum spanning forest for G taking into account the distance H and a total order on links. It starts with a forest of singleton trees (each sequence/profile is a tree). Then it constructs the optimal forest by adding links connecting profiles in different trees in increasing order accordingly to the total order, similarly to what is done in the Kruskal's algorithm [15]. In the current implementation, a total order for links is implicitly defined based on the distance between sequences, on the number of SLVs, DLVs, TLVs, on the occurrence frequency of sequences, and on the assigned sequence identifier. With this total order, the construction of the tree consists of building a minimum spanning forest in a graph [15], where each sequence is a node and the link weights are defined by the total order. By construction, the pairs at distance δ will be joined before the pairs at distance $\delta + 1$.

4 Experimental evaluation

We evaluated the proposed approach using both real and synthetic datasets. We used real datasets obtained through different typing schemas, namely wide-genome multi-locus sequence typing (wgMLST) data, core-genome multi-locus sequence typing (cgMLST) data, and single-nucleotide polymorphism (SNP) data. Table 2 summarizes the real datasets used. We should note that wgMLST and cgMLST datasets contain sequences of integers, where each column corresponds to a locus and different values in the same column denote different alleles. Synthetic datasets comprise sets of binary sequences of variable length, uniformly sampled, allowing us to validate our theoretical findings.

We implemented both versions described above in the C programming language: one based on binary search over the SA; and another one based on finding clusters in the LCP array. Since allelic profiles can be either string of letters or sequences of integers, we relied on `https://github.com/y-256/libdivsufsort` and `http://www.larsson.dogma.net/qsufsort.c` libraries, respectively. For RMQ over the LCP array, we implemented a fast well-known solution that uses constant time per query and linearithmic space for pre-processing [1].

All tests were conducted on a machine running Linux, with an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz (8 cores, cache 32KB/4096KB) and with 32GB of RAM. All binaries where produced using GCC 5.3 with full optimization enabled.

4.1 Synthetic datasets

We first present results with synthetic data for different values of d, m and k. All synthetic sequences are binary sequences uniformly sampled. Results presented in this section were averaged over ten runs and for five different sets of synthetic data.

The bound proved in Theorem 1 was verified in practice. For k satisfying the conditions in Theorem 1, the running time of our implementation grows almost linearly with n, the size of the input. We can observe in Fig. 1 a growth slightly above linear. Since we included the time for constructing the SA, the LCP array and the RMQ data structure, with the last one in linearithmic time, that was expected.

We also tested our method for values of k exceeding the bound shown in Theorem 1. For $d = m = 4096$ and a binary alphabet, the bound for k given in Theorem 1 is no more than $\lfloor m/(2 \log m) \rfloor = 170$. For k above this bound we expect that proposed approaches are no longer competitive with the naïve approach. As shown in Fig. 2, for $k > 250$ and $k > 270$ respectively, both limits above the predicted bound, the running time for both computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, becomes slower than the running time of the naïve approach.

In Fig. 3 we have the running time as a function of the number d of profiles, for different values of m and for k satisfying the bound given in Theorem 1. The running time for the naïve approach grows quadratically with d, while it grows linearly for both computing

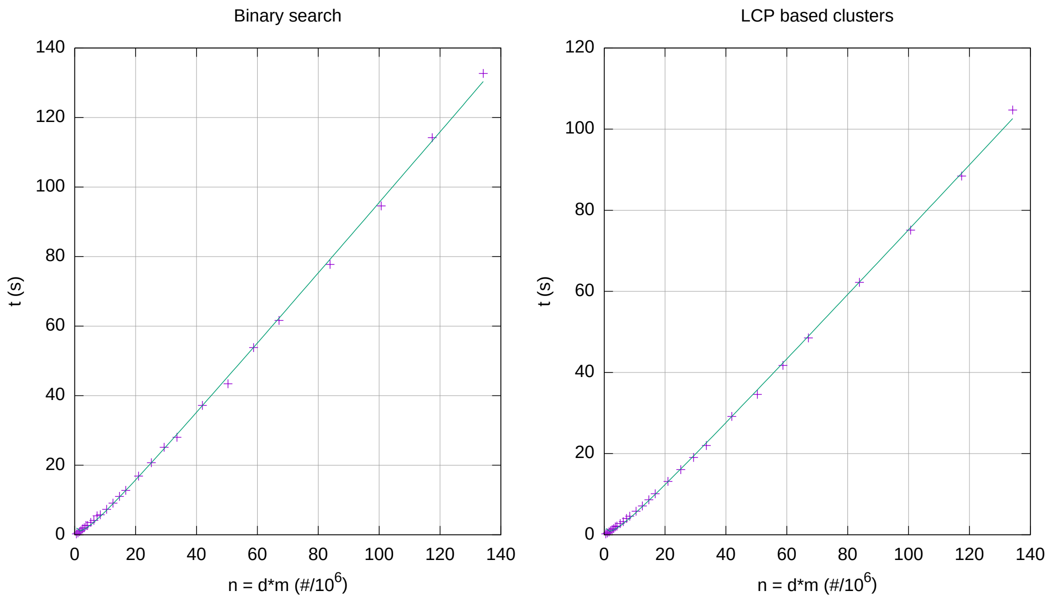


Figure 1 Synthetic datasets, with $\sigma = 2$ and $k = \lfloor m/(2 \log m) \rfloor$ according to Theorem 1. Running time for computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, as function of the input size $n = dm$.

Table 2 Real datasets used in the experimental evaluation. (*)Dataset provided by the Molecular Microbiology and Infection Unit, IMM.

Dataset	Typing method	Profile length	Number of distinct elements	Reference
Campylobacter jejuni	wgMLST	5446	5669	(*)
Salmonella enterica	wgMLST	3002	6861	[6]
Salmonella typhi	SNP	22143	1534	[20]
Streptococcus pneumoniae	cgMLST	235	1968	[4, 3, 19]

pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters. Hence, for synthetic data, as described by Theorem 1, the result holds.

4.2 Real datasets

For each dataset in Table 2, we ranged the threshold k accordingly and compared the approaches discussed in Section 2 with the naïve approach that computes the distance for all sequence pairs. Results are provided in Table 3.

In most cases, the approach based on the LCP clusters is the fastest up to two orders of magnitude compared to the naïve approach. As expected, in the case when data are not uniformly random, our method works reasonably well for smaller values of k than the ones implied by the bound in Theorem 1. As an example, the upper bound on k for *C. jejuni* would be around 200, but the running time for the naïve approach is already better for $k = 64$. We should note however that the number of candidate profile pairs at Hamming distance at most k is much higher than the expected number when data are uniformly random. This tells us that we can design a simple hybrid scheme that chooses a strategy (naïve or the

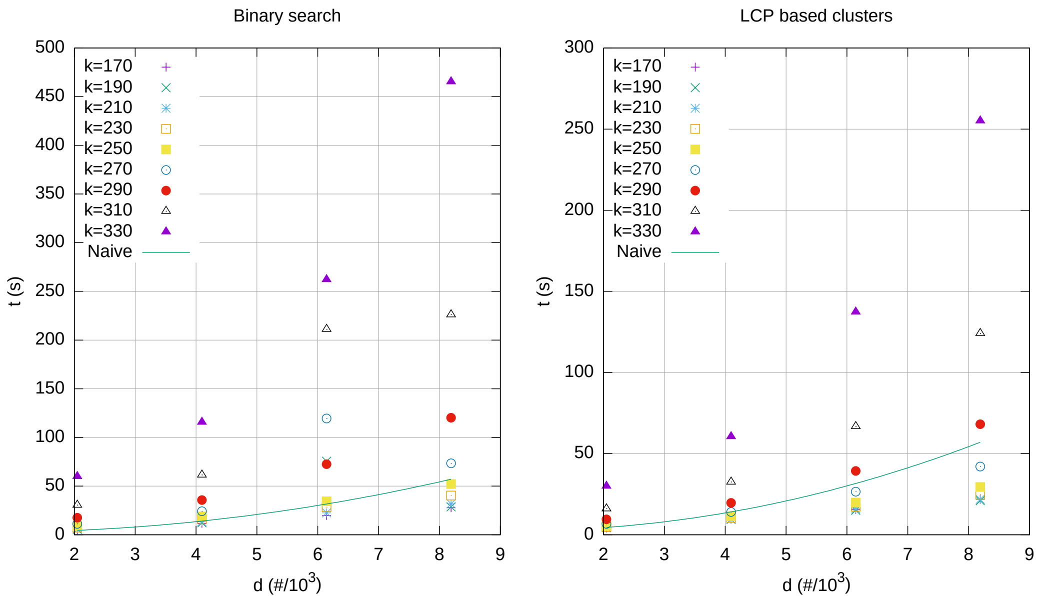


Figure 2 Synthetic datasets, with $\sigma = 2$ and $m = 4096$. Running time for computing pairwise distances by finding lower and upper bounds in the SA, and by processing LCP based clusters, as function of the number d of profiles and for different values of k.

proposed method) depending on the nature of the input data. It seems also to point out clustering effects on profile dissimilarities, which we may exploit to improve our results. We leave both tasks as future work for the full version of this article.

We incorporated the approach based on finding lower and upper bounds in the SA in the implementation of goeBURST algorithm, discussed in Section 3. We did not incorporate the approach based on the LCP clusters as the running time did not improve much as observed above. Since running times are similar to those reported in Table 3, we discuss only the running time for *C. jejuni*. We need only to index the input once. We can then use the index in the different stages of the algorithm and for different values of k. In the particular case of goeBURST, we use the index twice: once for computing the number of neighbors at a given distance, used for untying links according to the total order discussed in Section 3, and a second time for enumerating pairs at distance below a given threshold. Note that the goeBURST algorithm does not aim to link all nodes, but to identify clonal complexes (or connected components) for a given threshold on the distance among profiles [9]. In the case of *C. jejuni* dataset, and for $k = 52$, the running time is around 36 seconds, while the naïve approach takes around 115 seconds, yielding a three-fold speedup.

In this case we get several connected components, *i.e.*, several trees, connecting the most similar profiles. We provide the tree for the largest component in Fig. 4, where each node represents a profile. The nodes are colored according to one of the loci for which profiles in this cluster differ. Note that this tree is optimal with respect to the criterion used by the goeBURST algorithm, not being affected by the threshold on the distance. In fact, since this problem is a graphic matroid, the trees found for a given threshold will be always subtrees of the trees found for larger thresholds [21]. Comparing this tree with other inference methods is beyond the scope of this article; the focus here was on the faster computation of an optimal tree under this model.

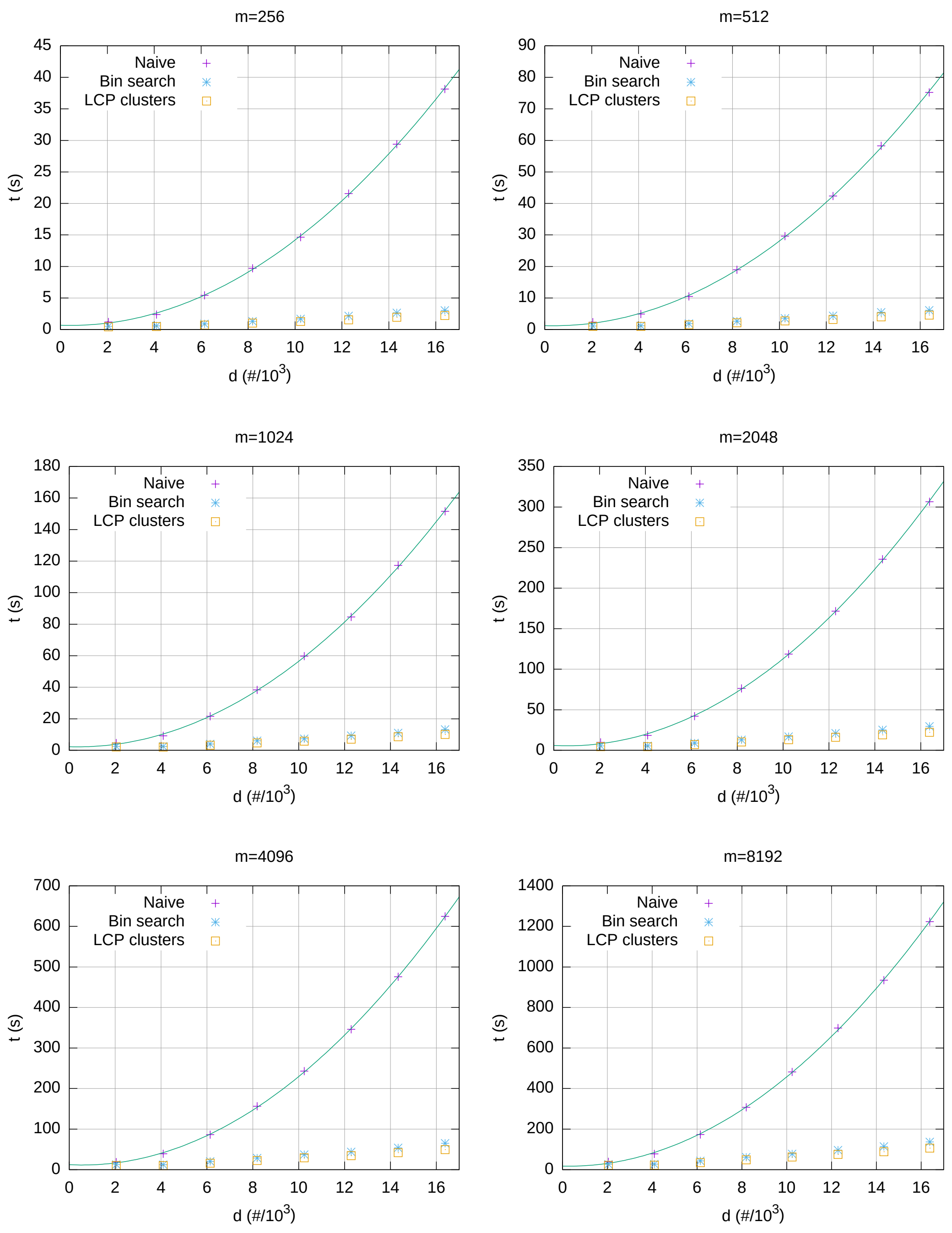


Figure 3 Synthetic datasets, with $\sigma = 2$ and $k = \lfloor m/(2\log m)\rfloor$ according to Theorem 1. Running time for computing pairwise distances naïvely, by finding lower and upper bounds in the SA, and by processing LCP based clusters, as a function of the number d of profiles.

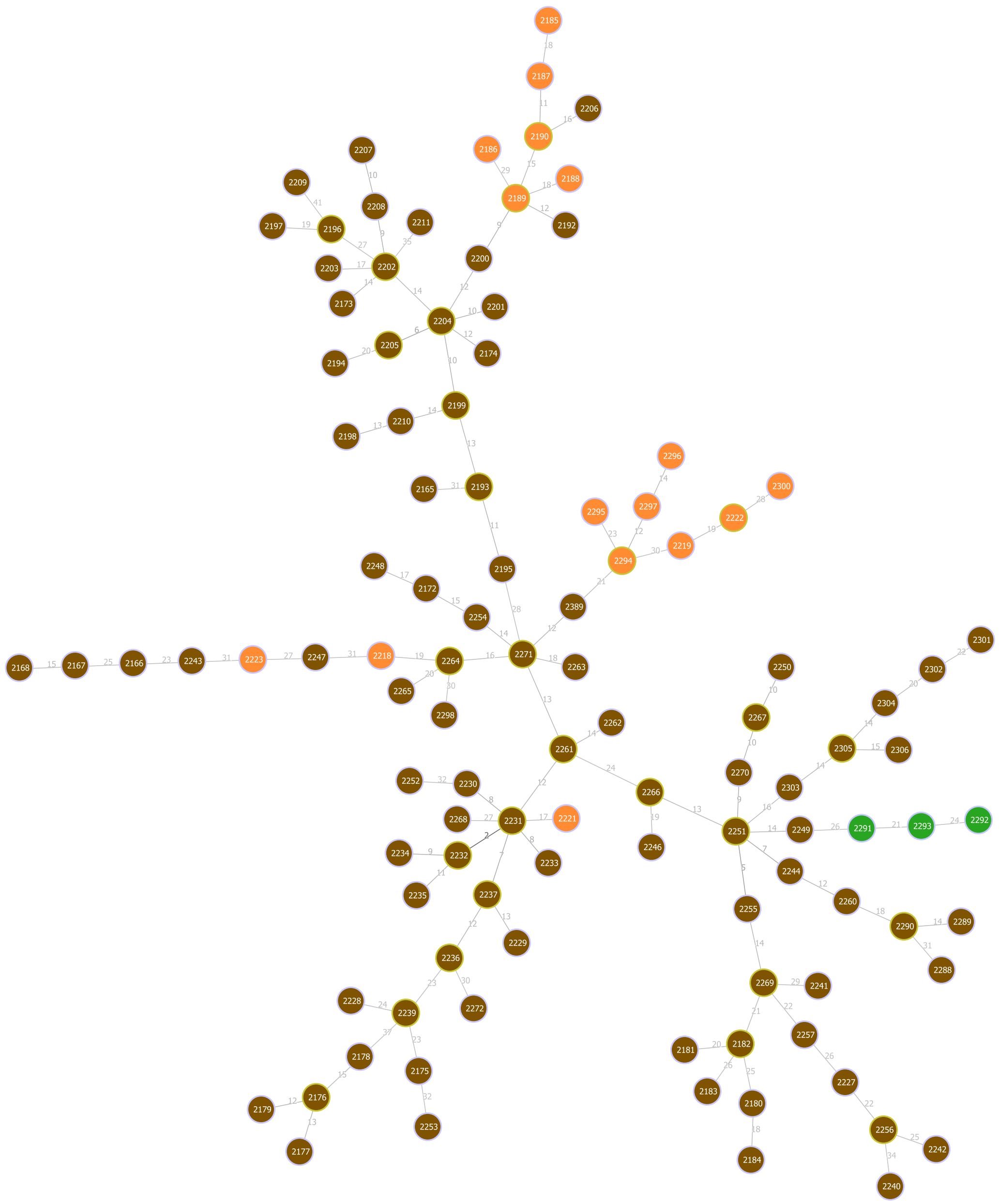


Figure 4 The tree inferred for the largest connected component found with $k = 52$ for the *C. jejuni* dataset. Image produced by PHYLOViZ [18].

Table 3 Time and percentage of pairs processed for each method and dataset.

Dataset	k	Naïve		Binary search		LCP clusters	
		t (s)	pairs (%)	t (s)	pairs (%)	t (s)	pairs (%)
C. jejuni	8	108.59	100	0.22	0.06	**0.17**	0.06
	16	109.30	100	0.48	0.32	**0.34**	0.32
	32	108.60	100	3.52	5.45	**2.67**	5.45
	64	**108.60**	100	231.05	99.98	162.36	99.98
S. enterica	8	89.85	100	1.04	2.37	**0.95**	2.37
	16	87.26	100	7.16	12.69	**6.73**	12.69
	32	85.36	100	36.29	33.22	**30.76**	33.22
	64	**84.63**	100	254.45	82.44	187.15	82.44
S. typhi	89	28.83	100	16.63	91.48	**12.02**	91.48
	178	**28.32**	100	46.98	99.91	32.03	99.91
	890	**30.04**	100	113.57	100	129.14	100
S. pneumoniae	8	0.56	100	0.02	0.93	**0.02**	0.93
	16	0.57	100	0.05	1.71	**0.04**	1.71
	32	0.56	100	0.20	4.42	**0.15**	4.42
	64	**0.58**	100	5.63	73.36	5.01	73.36

In many studies, the computation of trees based on pairwise distances below a given threshold, usually small compared with the total number of loci, combined with ancillary data, such as antibiotic resistance and host information, allows microbiologists to uncover evolution patterns and study the mechanisms underlying the transmission of infectious diseases [10].

5 Concluding remarks

Most distance-based phylogenetic inference methods rely directly or indirectly on Hamming distance computations. The computation of a distance matrix is a common first step for such methods, taking $\Theta(md^2)$ time in general, with d being the number of sequences or profiles and m the length of each sequence or profile. For large-scale datasets this running time may be problematic; however, for some methods, we can avoid to compute all-pairs distances [22].

We addressed this problem when only a truncated distance matrix is needed, *i.e.*, one needs to know only which pairs are at Hamming distance at most k. This problem was motivated by the goeBURST algorithm [9], which relies on a truncated distance matrix by construction. We proposed here an average-case linear-time and linear-space algorithm to compute the pairs of sequences or profiles that are at Hamming distance at most k, when $k < \frac{(m-k-1)\cdot\log\sigma}{\log md}$, where σ is the size of the alphabet. We integrated our solution in goeBURST demonstrating its effectiveness using both real and synthetic datasets.

We must note however that our analysis holds for uniformly random sequences and, hence, as observed with real data, the presented bound may be optimistic. It is thus interesting to investigate how to address this problem taking into account local conserved regions within sequences. Moreover, it might be interesting to consider in the analysis null models such as those used to evaluate the accuracy of distance-based phylogenetic inference methods [24].

The proposed approach is particularly useful when one is interested in local phylogenies, *i.e.*, local patterns of evolution, such as searching for similar sequences or profiles in large typing databases. In this case we do not need to construct full phylogenetic trees, with tens of thousands of taxa. We can focus our search on the more similar sequences or profiles, within a given threshold k. There are however some issues to be solved in this scenario, namely, dynamic updating of the data structures used in our algorithm. Note that after querying a database, if new sequences or profiles are identified, then we should be able to add them while keeping our data structures updated. Although more complex and dynamic data structures are known, a technique proposed recently for adding dynamism to otherwise static data structures can be useful to address this issue [17]. This and other challenges raised above are left as future work.

References

1 Michael A. Bender and Martín Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics: 4th Latin American Symposium*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

2 Michael A Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005. doi:10.1016/j.jalgor.2005.08.001.

3 Claire Chewapreecha, Simon R. Harris, Nicholas J. Croucher, Claudia Turner, Pekka Marttinen, Lu Cheng, Alberto Pessia, David M. Aanensen, Alison E. Mather, Andrew J. Page, Susannah J. Salter, David Harris, Francois Nosten, David Goldblatt, Jukka Corander, Julian Parkhill, Paul Turner, and Stephen D. Bentley. Dense genomic sampling identifies highways of pneumococcal recombination. *Nature Genetics*, 46(3):305–309, 2014. doi:10.1038/ng.2895.

4 Nicholas J Croucher, Jonathan A Finkelstein, Stephen I Pelton, Patrick K Mitchell, Grace M Lee, Julian Parkhill, Stephen D Bentley, William P Hanage, and Marc Lipsitch. Population genomics of post-vaccine changes in pneumococcal epidemiology. *Nature Genetics*, 45(6):656–663, 2013. doi:10.1038/ng.2625.

5 Richard Desper and Olivier Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computational Biology*, 9(5):687–705, 2002. doi:10.1089/106652702761034136.

6 EnteroBase. Enterobase.warwick.ac.uk. URL: http://enterobase.warwick.ac.uk.

7 Edward J. Feil, Edward C. Holmes, Debra E. Bessen, Man-Suen Chan, Nicholas P. J. Day, Mark C. Enright, Richard Goldstein, Derek W. Hood, Awdhesh Kalia, Catrin E. Moore, et al. Recombination within natural populations of pathogenic bacteria: short-term empirical estimates and long-term phylogenetic consequences. *Proceedings of the National Academy of Sciences*, 98(1):182–187, 2001. doi:10.1073/pnas.98.1.182.

8 Edward J. Feil, Bao C. Li, David M. Aanensen, William P. Hanage, and Brian G. Spratt. eBURST: inferring patterns of evolutionary descent among clusters of related bacterial genotypes from multilocus sequence typing data. *Journal of Bacteriology*, 186(5):1518–1530, 2004. doi:10.1128/JB.186.5.1518-1530.2004.

9 Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João Carriço. Global optimal eBURST analysis of multilocus typing data using a graphic matroid approach. *BMC Bioinformatics*, 10(1), 2009. doi:10.1186/1471-2105-10-152.

10 Alexandre P. Francisco, Cátia Vaz, Pedro T. Monteiro, José Melo-Cristino, Mário Ramirez, and Joao A. Carriço. PHYLOViZ: phylogenetic inference and data visualization for sequence based typing methods. *BMC Bioinformatics*, 13(1):87, 2012. doi:10.1186/1471-2105-13-87.

11 Daniel H. Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications.* Cambridge University Press, 2010. doi:10.1017/CBO9780511974076.

12 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

13 Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Annual Symposium on Combinatorial Pattern Matching*, pages 181–192. Springer, 2001. doi:10.1007/3-540-48194-X.

14 Pang Ko and Srinivas Aluru. Space efficient linear time construction of suffix arrays. In *Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2003. doi:10.1016/j.jda.2004.08.002.

15 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. doi:10.2307/2033241.

16 Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

17 J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Dynamic data structures for document collections and graphs. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, pages 277–289. ACM, 2015. doi:10.1145/2745754.2745778.

18 Marta Nascimento, Adriano Sousa, Mário Ramirez, Alexandre P. Francisco, João A. Carriço, and Cátia Vaz. PHYLOViZ 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods. *Bioinformatics*, 33(1):128–129, 2017. doi:10.1093/bioinformatics/btw582.

19 National Center for Biotechnology Information. GeneBank. URL: ftp://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/.

20 Andrew J. Page, Ben Taylor, Aidan J. Delaney, Jorge Soares, Torsten Seemann, Jacqueline A. Keane, and Simon R. Harris. SNP-sites: rapid efficient extraction of SNPs from multi-FASTA alignments. *Microbial Genomics*, 2(4), 2016. doi:10.1099/mgen.0.000056.

21 Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

22 Fabio Pardi and Olivier Gascuel. Distance-based methods in phylogenetics. In *Encyclopedia of Evolutionary Biology*, pages 458–465. Elsevier, 2016. doi:10.1016/B978-0-12-800049-6.00206-7.

23 D. Ashley Robinson, Edward J. Feil, and Daniel Falush. *Bacterial population genetics in infectious disease*. John Wiley & Sons, 2010. doi:10.1002/9780470600122.

24 Naruya Saitou. *Introduction to evolutionary genomics*. Springer, 2013. doi:10.1007/978-1-4471-5304-7.

25 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. doi:10.1093/oxfordjournals.molbev.a040454.

26 Robert R. Sokal. A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38:1409–1438, 1958.

Yanagi: Transcript Segment Library Construction for RNA-Seq Quantification*

Mohamed K. Gunady[1], Steffen Cornwell[2], Stephen M. Mount[3], and Héctor Corrada Bravo[4]

1 **Department of Computer Science, University of Maryland, College Park, MD, USA; and**
Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA
`mgunady@cs.umd.edu`

2 **School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA, USA**

3 **Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA; and**
Department of Cell Biology and Molecular Genetics, University of Maryland, College Park, MD, USA
`smount@umd.edu`

4 **Department of Computer Science, University of Maryland, College Park, MD, USA; and**
Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA
`hcorrada@umiacs.umd.edu`

Abstract

Analysis of differential alternative splicing from RNA-seq data is complicated by the fact that many RNA-seq reads map to multiple transcripts, and that annotated transcripts from a given gene are often a small subset of many possible complete transcripts for that gene. Here we describe Yanagi, a tool which segments a transcriptome into disjoint regions to create a segments library from a complete transcriptome annotation that preserves all of its consecutive regions of a given length L while distinguishing annotated alternative splicing events in the transcriptome. In this paper, we formalize this concept of transcriptome segmentation and propose an efficient algorithm for generating segment libraries based on a length parameter dependent on specific RNA-Seq library construction. The resulting segment sequences can be used with pseudo-alignment tools to quantify expression at the segment level. We characterize the segment libraries for the reference transcriptomes of Drosophila melanogaster and Homo sapiens. Finally, we demonstrate the utility of quantification using a segment library based on an analysis of differential exon skipping in Drosophila melanogaster and Homo sapiens. The notion of transcript segmentation as introduced here and implemented in Yanagi will open the door for the application of lightweight, ultra-fast pseudo-alignment algorithms in a wide variety of analyses of transcription variation.

1998 ACM Subject Classification I.1.2 Algorithms

Keywords and phrases RNA-Seq, Genome Sequencing, Kmer alignment, Transcriptome Quantification, Differential Alternative Splicing

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.10

* This work was partially supported by NSF grant ABI 1564785 to SMM and MKG, and NIH grants HG005220 and GM114267 to HCB, MKG and SC.

1 Introduction

Messenger RNA transcript abundance estimation from RNA-Seq data is a crucial task in studies that seek to describe the effect of genetic or environmental changes on gene expression. Differential expression analysis over either genes or transcripts is used to find the set of genes or transcripts with different expression levels between conditions. Although there are many tools that can provide satisfying results at the gene level, transcript level analysis still faces major challenges that make the problem of transcript expression quantification harder.

Over the years, various approaches have addressed the joint problems of (gene level) transcript expression quantification and differential alternative RNA processing. Much effort in the area has been dedicated to the problem of efficient alignment of reads to a genome or a transcriptome, since this is typically a bottleneck in the analytical processes that start with RNA-Seq reads and yield gene-level expression or differentially expressed transcripts. Among these approaches are alignment techniques such as bowtie [6], Tophat [16, 5], and Cufflinks [17], and newer techniques such as sailfish [10], RapMap [13], Kallisto [2] and Salmon [9], which provide efficient strategies that are much faster, but maintain comparable, or superior, accuracy.

In order to achieve faster alignment and quantification, the newer methods introduced novel approaches, such as alignment-free k-mer based quantification [Sailfish], quasi-mapping [RapMap], pseudo-alignment [Kallisto], or lightweight alignment [Salmon]. These methods all simplified the expected outcome of the alignment step, finding only sufficient read-alignment information required by the quantification step, utilizing k-mer counting to be the sufficient statistic built from the alignment step. In other words, given a transcriptome reference, an index of kmers is created and used to find a mapping between reads and the list of compatible transcripts based on each approach's definition of compatibility. The next step would be to resolve the ambiguity in reads that were mapped to multiple transcripts. Multi-mapping reads are common even assuming error free reads, due to shared regions produced by alternative splicing. The ambiguity in mapping reads is resolved using probabilistic models, such as the EM algorithm, to produce the abundance estimate of each transcript [8].

The presence of sequence repeats and paralogous genes in many organisms also creates ambiguity in the placement of reads. Moreover, the fact that alternatively spliced transcripts share most of their genomic region, greatly increases the portion of reads coming from these shared regions and consequently reads being multi-mapped becomes more frequent when aligning to enumerated transcripts. In fact, local splicing variations can be joined combinatorially to create a very large number of possible transcripts from many genes. An extreme case is the Drosophila gene Dscam, which can produce over 38,000 transcripts by joining less than 50 exons [19]. More generally, long-read sequencing indicates that although there are correlations between distant splicing choices [15], a large number of possible combinations is typical. Thus, standard annotations, which enumerate only a minimal subset of transcripts from a gene (e.g. [3]) are inadequate descriptions. Furthermore, short read sequencing, which is likely to remain the norm for some time, does not provide information for long-range correlations between splicing events.

In this paper, we propose a novel strategy that aims at constructing a set of segments that can be used in the read-alignment-quantification steps instead of the whole transcriptome without loss of information. Such a set of segments (a segment library) can fully describe individual events (primarily local splicing variation, but also editing sites or sequence variants) independently, leaving the estimation of transcript abundances as a separate problem. Here we introduce and formalize the idea of transcriptome segmentation, propose and analyze an algorithm for transcriptome segmentation, and present a tool called Yanagi, which implements this segmentation algorithm to build a segment library from a reference transcriptome based on the possible splicing variations. We show results from the application of Yanagi to reference transcriptomes of Drosophila melanogaster and Homo sapiens that characterize the resulting segment libraries. Since the segment libraries are amenable for usage with lightweight pseudo-alignment methods for segment quantification, we illustrate the utility of the segmentation approach using the differential analysis of exon skipping events across samples from two conditions of interest. We use simulation studies in Drosophila melanogaster and Homo sapiens and show that this is a promising approach for this type of analysis.

2 Methodology

Exonic regions of a messenger RNA precursor can be combined differently through alternative splicing (AS) to form distinct isoforms. Alternative transcripts can be generated by AS proper (including exon skipping, mutual exon exclusion, intron retention, and alternative splice site use), alternative transcription start sites, and alternative 3' termini (sites of cleavage and polyadenylation). Combinations of these allow more complex events. A comprehensive treatment of the different types of splicing events can be found in [18]. Over 95% of human genes with multiple exons undergo AS [18]; consequently a majority of the coding genomic region is spliced into more than one isoform.

The goal of our approach is to segment the transcriptome into a set of disjoint regions (where disjointness is parameterized by a specific read length) without losing any possible transcriptome sub-sequence that may be sequenced in a given RNA-Seq experiment. Afterwards, we can pseudo-align reads into the set of segments and quantify abundance at the segment level for use in further downstream analysis. Consequently, our quantification pipeline can use available kmer-based pseudo-mapping or pseudo-alignment techniques over the set of segments generated by Yanagi from the transcriptome reference and generate counts for segments. The rest of this section describes Yanagi's algorithm for generating the segment library. We later discuss how it can be used for quantification purposes using differential analysis of exon skipping events as an illustrative use case.

2.1 Transcriptome Segments Properties

▶ **Definition 1** (Segment). A segment $seg(Exs, loc, w)$ is a genomic region of width w beginning at genomic location loc and spanning the sequence of consecutive exonic regions Exs. Exonic regions are considered consecutive if they are consecutively joined into at least one possible isoform.

▶ **Definition 2** (L-disjoint property). The set of segments S is *L-disjoint* if and only if

$$width[overlap(seg_i, seg_j)] < L; \forall seg_i, seg_j \in S, i \neq j$$

That restricts any pair of *L-disjoint* segments to have an overlap region shorter than parameter L, corresponding to the read length of a specific RNA-Seq experiement. In other words no read of length at least L can be mapped to both segments of an *L-disjoint* segment pair, assuming error-free reads.

Given a reference transcriptome, a naive approach to generating such L-disjoint segments would be to use the set of exonic regions and junctions defined in the transcriptome and generate segments spanning each exonic region and junction. Specifically, a junction segment would be formed by spanning $L-1$ positions from both sides of the junction and exon

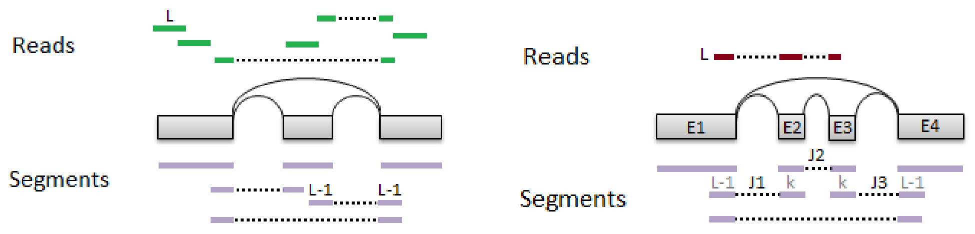


Figure 1 An example of naive segments based on exons and junctions. Two cases are shown, each is represented using a splicing graph of two transcripts, along with a set of possible RNA-seq reads and the generated segments following the naive approach. The first case (left) shows a simple case where the naive approach successfully generates segments spanning all possible reads. The second case (right) shows a case of two short exons (E2, E3 of width $k < L$) where the naive approach fails to span the given read.

segments would simply include the genomic sequence of the exon that does not overlap any of the junction segments. Figure 1 (left) shows a simple exon skipping event using splicing graph representation [4] and the corresponding generated segments following that naive approach. This approach would successfully generate segments capturing all possible sequences required to map any read to the transcriptome. However this naive approach faces a few challenges.

First, exons that are shorter than parameter L are problematic. For instance, around 30% of the exons in the UCSC hg38 genome are shorter than 100bp (Illumina's common paired-end read length). These short exons will make junction segments miss reads that span more than two of such short exons. Consider the example in Figure 1 (right) where the two exons E2, E3 of width k are both shorter than L, no segments will capture a read that span E1, E2, and E3 for instance.

Another related challenge is that the annotated exons are not strictly disjoint in the reference itself. Some annotated exons overlap due the use of alternative transcription start and end sites. Such challenges indicate that more careful choice of segments is necessary to guarantee the L-disjointness property, so we formalized an additional segment property.

First, denote $Txs(exs)$ as the set of annotated transcripts splicing exons $exs \in Exs$, and $Txs(seg)$ as the union of $Txs(exs)$ for exons exs included in segment seg. We can define a subsumption relationship between segments as $seg_1(Exs, loc, w) \succ seg_2(Exs, loc, w)$ if $Txs(seg_1) = Txs(seg_2)$ and $width(seg_1) > width(seg_2)$. With this relationship we can define the following property of a segment library.

▶ **Definition 3** (Max-spanning property).

$$seg_1(Exs, loc, w) \succ seg_2(Exs, loc, w) \Rightarrow seg_2(Exs, loc, w) \notin S, \forall seg_1(Exs, loc, w) \in S\,.$$

Thus a segment is the longest common sequence of genomic regions starting at loc, such that these regions are spliced similarly, i.e. the entire sequence belongs to the same set of transcripts. That means the three junctions J1, J2, J3 shown in 1 (right) will be concatenated into one segment which captures any read of length L spanning any of these junctions.

2.2 Segmentation Algorithm Overview

Given the transcriptome annotation (GTF format file) and the transcript sequences (FASTA format files) as input, Yanagi generates the set of segments and its sequences (as a FASTA file) as the output of the segmentation process. Figure 2 illustrates an example of how Yanagi perform transcriptome segmentation given the splicing graph of a complex AS event studied in [18]. Recall, that in splicing graphs, nodes represent genomic regions and edges represent how the regions are spliced, while paths represent possible transcripts.

The transcriptome segmentation process can be summarized into three steps: (1) Preprocessing the transcriptome annotation in order to obtain disjoint transcriptome regions, (2) Constructing a Segments Graph (SgG), and finally (3) Generating the segment library. Each transaction in Figure 2 represents one of these three steps.

2.3 Preprocessing

In our algorithm, exons and junctions serve as initial candidates for segment generation. We apply a preprocessing step to eliminate exon overlaps present in the transcriptome reference from events involving alternative 3'/5' splice sites, or transcription start/end sites. This step ensures that any splicing event is occurring either at the beginning or the end of a genomic segment, which makes the process of generating L-disjoint and max covering segments easier. The preprocessing step is independent from the parameter L, so it can be done only once per transcriptome reference. We implemented the preprocessing step based on the GenomicRanges package in R, specifically the *disjoin* function, which takes less than a few seconds to run on the human genome.

2.4 Segments Graph

Currently Yanagi builds a separate segment graph for each gene, since there are no alternative splicing events between transcripts of different genes. However, future work may use segment graphs that connect different genes sharing regions of identical sequence length L or greater, but we have yet to address this.

▶ **Definition 4** (Segment Graph)**.** A segment graph G is an acyclic directed graph defined by the pair (N, E), where N is a set of nodes representing segments, and E is the set of directed edges between the nodes. An edge $e : (n_i, n_j) \in E$ is created if the segment corresponding to node n_i directly precedes the segment corresponding to node n_j in some transcript.

▶ **Definition 5** (Segment Node)**.** A segment node n is a node in segment Graph G that represents an *L-disjoint* and *max-spanning* segment $seg_n(Exs_n, loc_n, w_n)$, such that $w_n \geq L$.

While full details of the algorithm are given in Appendix A, here we present a high-level description. For a given gene, the algorithm iterates over the set of annotated transcripts in that gene. A cursor loc starting at the beginning of a transcript slides over the sequence of genomic regions forming that transcript. Given the current cursor location loc_n, a seed of the node n is initiated. Then a refinement step, explained further in the next paragraph, is used to handle cases involving exons shorter than L. The segment node is then added to the graph with the key pair (exs_n, loc_n) as the node identifier, and the cursor loc is advanced to the new location. It should be noted that the out-degree of each segment node corresponds to the number of upcoming alternative splices. The final step is generating the actual segments. Any segment with an out-degree greater than one is a candidate start of a segment. Each possible path beginning at a start-segment node till the following start-segment node (or a leaf node) produces an output segment. See Figure 2 and Appendix A for the full details of the segmentation algorithm. It is worth mentioning that a segment graph may look like a de Bruijn graph (DBG) that is commonly used in assembly problems. However, a path of nodes in DBG represents a sequence of k-mer components while a path of nodes in SgG represents a sequence of genomic regions spliced into an isoform. As a result, the DBG built from the list of transcripts and the DBG built from the list of segments should be identical, since both graphs represent the same sequences of nucleotides.

Back to the issue of short regions which raises the possibilities of generating segment nodes of length L spanning more than two exonic regions. Once the key pair (exs_n, loc_n) is determined, the node refinement step determines the extent of that node and how the cursor loc should be advanced in order to preserve the L-disjointness constraint. Figure 3 shows a diagram of how a segment node is refined. The logic behind the refinement step is aggregating the sequence of nodes expected to be created spanning the same set of regions exs_n; since it is guaranteed that there are no splicing events occurring between the start and end of region Ex_n, the next necessary segment node would be the node spanning part of Ex_{n+1}. Consequently, the new location of the cursor loc would be the location where the first segment spanning Ex_{n+1} starts. That aggregation improves the time and space complexity of the algorithm as it reduces the number of generated nodes, and avoids shredding the genome in dense areas of short exons which may impose a problem in the quantification step as discussed in next subsection. In fact, the refinement step ensures that for every distinct value of exs, there is a maximum of two segment nodes generated and that reduces the algorithm complexity by factor of L.

As an attempt to analyze the complexity of the algorithm, we can estimate a loose upper bound of the number of segment nodes N in G. Consider a gene with T_g transcripts and E_g disjoint genomic regions (obtained by the preprocessing step), where the maximum width of such a region is w_{max}. Recalling the property mentioned in the previous paragraph, that a maximum of two nodes can be generated for the same set of regions, a segmentation iteration for a transcript that spans $E_t \leq E_g$ regions can generate $1 < o(E_t - \left\lceil \frac{L}{w_{max}} \right\rceil) < o(E_t)$ segment nodes. That gives the upper bound of $o[\sum_{T_g}(E_t - \left\lceil \frac{L}{w_{max}} \right\rceil)]$ or $o[T_g.(E_g - \left\lceil \frac{L}{w_{max}} \right\rceil)]$ for N. That means the time and space complexity of the graph construction increases when using lower values of parameter L, or with organisms of longer and more complex transcriptome structure. Table 1 shows time and memory analysis for constructing the segments library for two organisms used in our later analysis. The results shows that running Yanagi is an efficient and fast process that does not add a burden in terms of time and space requirements.

2.5 Quantification Analysis

After the transcriptome segmentation stage, Yanagi provides the set of generated segments in FASTA format. The segment sequences are accompanied by headers specifying metadata of how each segment was formed, including: gene ID, the set of exonic regions exs included in the segment, start and end locations in the first and last spanned regions, and the set of transcripts corresponding to the segment. The quantification stage starts afterwards by supplying the segment sequences to the preferred kmer-based pseudo-alignment tool, e.g. kallisto, sailfish or RapMap. For single end reads we can obtain segment counts from these tools. In the paired end case, we obtain pseudo alignments from either kallisto or RapMap as a BAM file for each read in the read pair independently. The two generated BAM files are then processed together to obtain segment-pair read counts. A read r_1 is counted toward a pair of segments $< seg_1, seg_2 >$ if the first end is mapped to seg_1 and the second end to seg_2 while both seg_1 and seg_2 have at least one transcript in common. This latter condition ensures that the counts reflects only annotated transcripts, although this condition can be relaxed in principle in favor of identifying unannotated transcripts.

After the segment quantification stage, a count tables for each sample is prepared to be used in the downstream analysis. In this paper we illustrate a workflow based on Yanagi

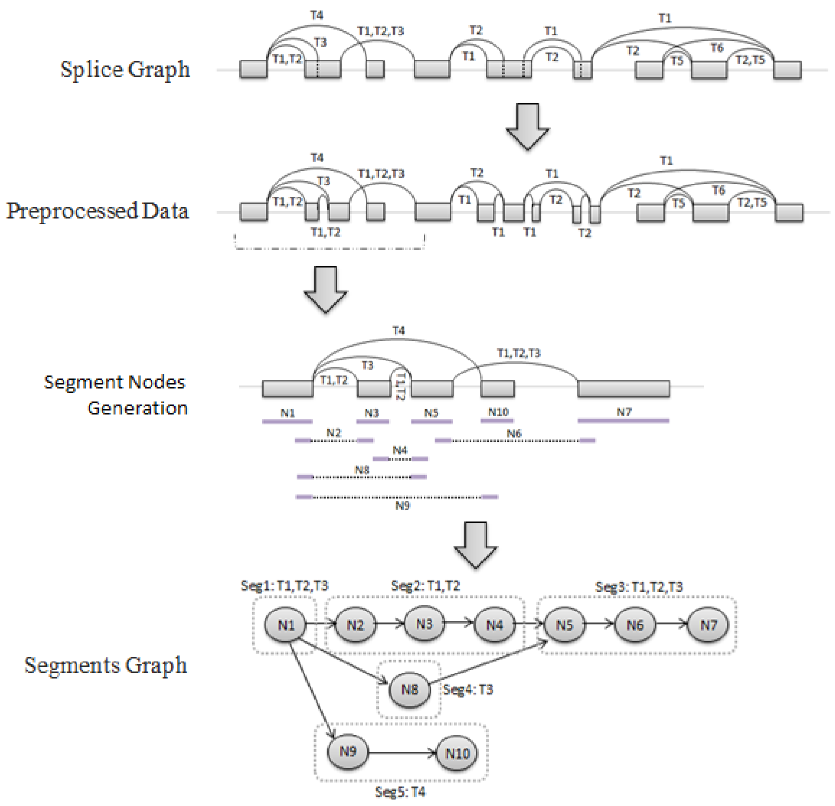


Figure 2 The process of generating segments using the splice graph for an example of a complex splicing event. Each transition represents one of the three main steps of the transcriptome segmentation process. Assuming no short exons for simplicity. Step two and three are cropped to include only the beginning portion of the graph for brevity.

using the problem of differential analysis of exon skipping events across samples from two conditions of interest. By providing the list of annotated splicing events, Yanagi maps each exon skipping event with its corresponding set of segments and sums their counts. For example, an exon skipping event is defined by three exons as in Figure 1 (left). Two segment-level counts are calculated: one from segments spanning the inclusion junction and another from the segments spanning the skipping junction. Note that although segments are L-disjoint, if the skipped exon is shorter than L we can have more than one segment spanning the inclusion junctions. Figure 4 illustrates a full workflow based on Yanagi, assuming paired-end reads and targeting AS quantification.

3 Experiments

3.1 Segment Analysis

To analyze the outcome of the segmentation stage, we used Yanagi to build segment libraries for the fruit fly and human genomes: Drosophila melanogaster (UCSC dm6) and Homo sapiens (UCSC hg38) genome assemblies and annotations respectively. These organisms show different genome characteristics, *e.g.* the fruit fly genome has longer exons and transcripts than the human genome, while the number of transcripts per gene is much higher for human genome than the fruit fly. A summary of the properties of each genome is found in [12].

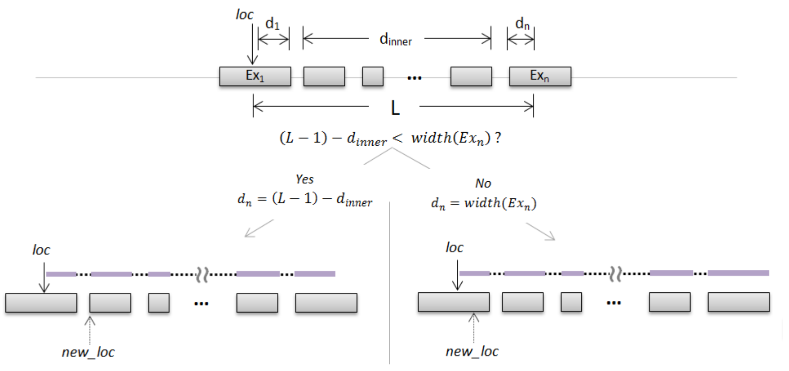


Figure 3 Diagram illustrates the node refinement step, for a node spanning n genomic regions. The step determines the extent of the node and how the cursor *loc* should advance in each of the two candidate cases.

Since L is the only parameter value required by the segmentation algorithm, we tried different values of L to understand the impact of that choice on the generated segments library. Recall that the choice of L is based on the expected read length of the sequencing experiment. For this analysis we chose the set $L = (40, 108, 1000, 10000)$.

Figure 5 shows the histogram of the lengths of the generated segments compared to the the full lengths of the transcripts, for each value of L, for both fruit fly (left) and human (right) genomes. It should be noted that the generated segments should be of at least length L. However, there are the exceptions of segments hitting the end of the transcript where the remaining portion of transcript is shorter than L. The figure shows the expected behavior when increasing the value of L; using small values of L tends to shred the transcriptome more (higher frequencies for small sequence lengths), especially with genomes of complex splicing structure like the human genome. While with high values of L, such as $L = 10,000$, the minimum segment length required tends to be higher than the length of most transcripts, ending up generating segments such that each segment represents a whole transcript.

Figure 6 shows how the number of generated segments in a gene is compared to the number of the transcripts in that gene, for each value of L, for both fruit fly (left) and human (right) genomes. A similar behavior is observed while increasing the value L, as with the segments length distribution. The fitted line included in each scatter plot provides indication of how the number of target sequences grows compared to the original transcriptome. For example when using $L = 108$, which is a suitable value with Illumina reads, the number of target sequences per gene, which will be the target of the subsequent pseudo-alignment steps, almost doubles. It is clear from both figures the effect of the third step in the segmentation stage. It is important not to shred the transcriptome so much that the target sequences become very short leading to resulting complications in the pseudo-alignment and quantification steps, and not to increase the number of target sequences leading to increasing the processing complexity of these steps.

3.2 Use Case: Differential Exon Skipping

We use the analysis of differential exon skipping events across samples from two conditions of interest as a use case of how to apply segment-level quantification in downstream analysis.

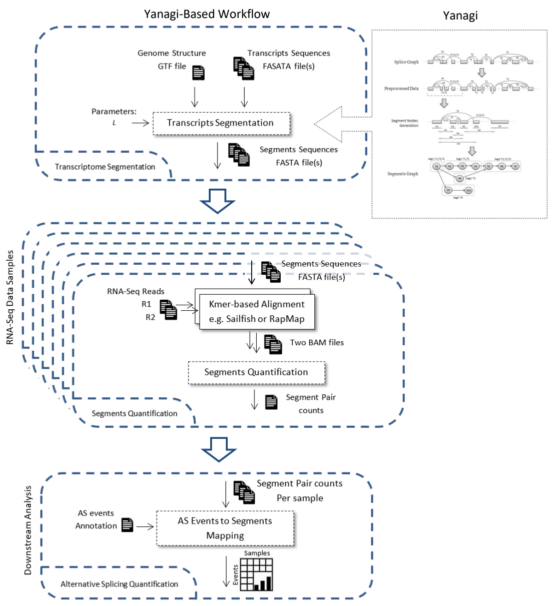

Yanagi-Based Workflow
Yanagi
Genome Structure GTF file
Transcripts Sequences FASATA file(s)
Parameters: L
Transcripts Segmentation
Segments Sequences FASTA file(s)
Transcriptome Segmentation
RNA-Seq Data Samples
RNA-Seq Reads R1 R2
Kmer-based Alignment e.g. Sailfish or RapMap
Two BAM files
Segments Quantification
Segment Pair counts
Downstream Analysis
Segment Pair counts Per sample
AS events Annotation
AS Events to Segments Mapping
Samples
Events
Alternative Splicing Quantification

Figure 4 Yanagi-based workflow for alternative splicing analysis, based on paired-end RNA-Seq reads. Dotted blocks are components introduced to assist Yanagi.

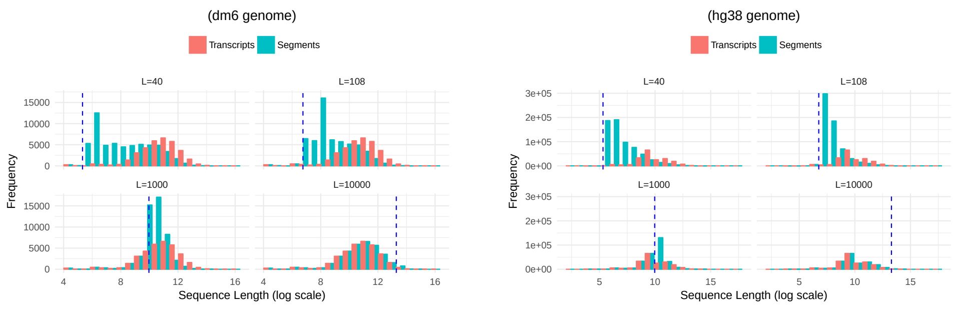


Figure 5 Histogram of transcripts lengths vs. segments lengths for both fruit fly (left) and human (right) genomes, with different values of L (40, 108, 1000, 10,000). Dotted vertical line represents the used value of L during the transcriptome segmentation.

Datasets. The experiments are based on the simulation data provided by [12] for both fruit fly and human organisms (dm3 and hg37 assembly versions, respectively). Each dataset consists of samples from two conditions. Each condition has three replicates. The reads for the replicates are simulated from real RNA-seq samples, to get realistic expression values, after incorporating a variance model and the change required between conditions. The simulation is restricted to only protein-coding genes in the primary genome assembly. The difference in transcripts usage across conditions was simulated in 1000 genes randomly selected from genes with at least two transcripts and high enough expression levels. For each of these 1000 genes, the expression levels of the two most abundant transcripts is switched across conditions. Refer to [12] for full details of the preparation procedure of the dataset.

Differential Splicing Model. Recall that the outcome of the alternative splicing quantification workflow (as in figure 4) is two counts per exon skipping event for each sample: the inclusion count and the exclusion count. The count matrix is then used in a linear model for

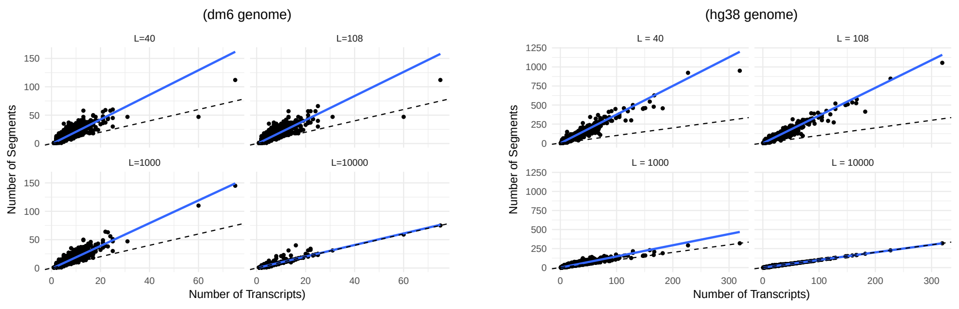


Figure 6 Number of transcripts vs. number of segments, per gene, for both fruit fly (left) and human (right) genomes, with different values of L (40, 108, 1000, 10,000). The figure shows how a fitted line (solid blue) compares to the identity line (dotted black).

differential splicing detection for count y_{ij} of segment i, sample j:

$$\log_2 y_{ij} = b_0 + \delta_c(j) \times (2\delta_t(i) - 1) \times b_1 + \delta_t(i) \times b_2$$

with

$$\delta_c(j) = \begin{cases} 1, & \text{if } j \text{ is } \textit{case} \\ 0, & \text{o.w.} \end{cases}$$

$$\delta_t(i) = \begin{cases} 1, & \text{if } i \text{ is } \textit{inclusion} \\ 0, & \text{o.w.} \end{cases}$$

Parameter b_0 models the expected log counts for exclusion segment in control samples, and parameter b_2 models expected log counts for the inclusion segment in control samples. With this, parameter b_1 corresponds to the log-odds ratio for the Percent Spliced In (PSI) statistic frequently used for alternative splicing analysis. Denoting Ψ_0 and Ψ_1 as the PSI values for case and control respectively, $b_1 = 1/2 \log_2 (\frac{\Psi_1}{1-\Psi_1} / \frac{\Psi_0}{1-\Psi_0})$. This model is similar to the DEXSeq model [1], modified to accomodate inclusion/exclusion counts at the single event level.

Preliminary Results. We tested this model over the synthetically generated data for both genomes (using the Limma-voom R Package [11, 7]) using the segments library generated by Yanagi with different values of the parameter L. The preliminary results shown here considers exon skipping events that involve only one inclusion transcript and one exclusion transcript, just to obtain a reasonable level of stability. In addition, only transcripts of high enough expression levels are considered, i.e. the two transcripts have a combined TPM value of 1. Similar constraints is advised in previous analysis [12, 14].

Figure 7 shows ROC plots for sensitivity and specificity measures, for each genome. As a reference, we applied the same linear model on the transcripts' true TPMs provided while preparing the dataset. In principle, the prediction based on the true transcript expression levels (TPMs) would give an upper bound performance, given the particular setting of this synthetic dataset and the provided linear model. Figure 7 shows that using our segments library with the proposed workflow, and using the suitable value of L, gives promising results in detecting differential splicing events (Table 2 shows the AUC values). A suitable value of L would be a value corresponds to the read length of the data, as discussed earlier in Definition 2. While using lower value of L intensely shreds the reference into segments shorter

Table 1 Running time (seconds) and memory usage (gigabytes) by Yanagi to generate segment library for fruit fly (Dm6) and human (Hg38) genomes, for both the preprocessing and segmentation steps. Time for the preprocessing step does not include the time to load the FASTA and GTF files. Most of the memory usage is from loading the input data in both steps. Running on a 6-core 2.1 GHz AMD processor, using single-threaded processes. The lower half shows the time and memory usage for running Rapmap's quasi-mapping using the segments library and the the full transcriptome, to quantify samples of 40M paired-end reads, each of length 101bp.

	Dm6		Hg38	
	time(s)	memory(GB)	time(s)	memory(GB)
Preprocessing	13	0.9	112	1.5
Segmentation				
L=40	20	0.4	248	1.3
L=108	20	0.4	250	1.3
L=1000	20	0.4	228	1.3
L=10000	8.5	0.4	77	1.3
Rapmap Indexing (4 Threads)				
L=108	103	0.8	420	2.6
Txs	121	1.1	480	3.7
Rapmap Quantification (8 Threads)				
L=108	236	0.7	220	2.1
Txs	292	1.2	416	3.1

Table 2 AUC values for the ROC curves in Figure 7 for both fruit fly and human genomes. Including the AUC value when using the transcripts' true TPM, besides AUC values from applying the linear model to the generated segments counts using different values of L.

	Tx.TPM	Seg.Counts			
		$L = 40$	$L = 108$	$L = 1000$	$L = 10000$
Dm3	0.912	0.762	0.889	0.824	0.760
Hg37	0.916	0.706	0.881	0.754	0.727

than the reads which hurts the quantification step, using higher values rather increases the portion of unnecessary overlap between the generated segments, leading to higher rates of multi-mapped reads.

The results also shows that the segmentation approach gives slightly better performance in the fruit fly case. That matches the fact that the fly transcriptome structure is much simpler than the human transcriptome. Besides, the sequences in the fruit fly are more unique with no allele repeats as the case with the human genome. That makes the counts in the quantification step more stable for the differential analysis.

4 Discussion and Conclusion

In this paper we introduce Yanagi, an efficient tool that creates disjoint segments of reference transcriptomes amenable for quantification of RNA-seq reads using pseudo-alignment techniques. We have formalized the notion of transcriptome segmentation, and proposed an efficient algorithm for constructing L-disjoint, max-spanning segments. We report on the

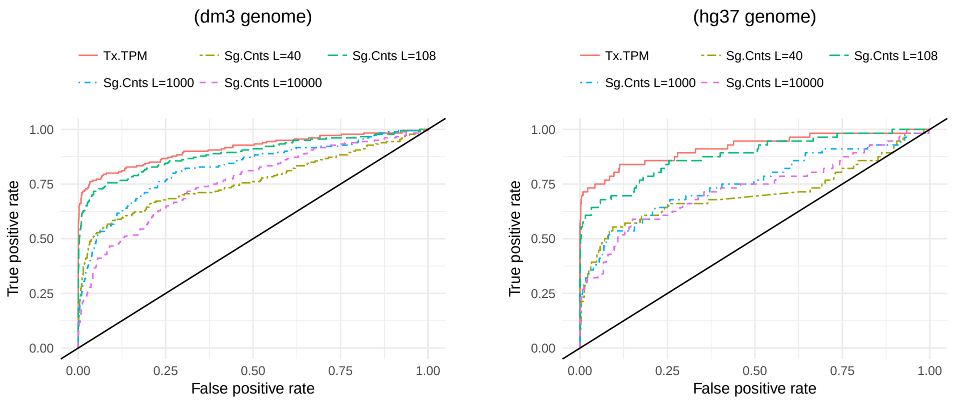


Figure 7 ROC plots for differential alternative splicing (exon skipping events), for both fruit fly (left) and human (right) genomes. ROC curves are included for: transcript-level quantification, besides segment-level quantification using different values of the parameter L.

characteristics of segment libraries in Drosophila melanogaster and Homo sapiens, and use the resulting segments in a use case of differential analysis of exon skipping events across samples in two conditions of interest.

Although it may appear that the discussed Yanagi-based workflow can perform quantification only for the annotated transcripts, the workflow can be extended to discover unannotated transcripts. An unannotated junction can be detected during the segment quantification stage by relaxing the restriction of accepting alignments of a pair of reads only if the pair of segments belong to at least one annotated transcript. When this restriction is relaxed, an unannotated junction can be detected when reads show enough evidence of that junction. I.e. when a segment pair that has no transcripts in common has high enough count.

Finally, the issues of paralogs and intersecting genes are not tackled in the scope of this paper. However, it is clear that there is no extra alignment complexity added due to these issues over the transcriptme-based alignment. Consequently, the occurrence of multi-mapping resulting from such cases also remains the same as the transcriptome-based quantification. So a warranted extension to the current approach of Yanagi is to consider distinct genes that share identical exonic regions of length greater than L altogether.

The concept of transcriptome segmentation, and a tool that can build a segments library, opens the door for more extended analysis than just the use case mentioned in this paper. For instance, segment counts can serve as statistics into algorithms for differential isoform usage analysis, for which existing pseudo-alignment methods are commonly used. Moreover, segment level quantification can provide much more flexible opportunities for analysis including quantification of RNA editing or other non splicing variations. Currently we are exploring the possibility of utilizing the concept of segmentation into the problem of variant calling.

Acknowledgements. We want to thank Julien Buchbinder for preliminary work in transcript segmentation.

References

1 Simon Anders, Alejandro Reyes, and Wolfgang Huber. Detecting differential usage of exons from RNA-seq data. *Genome research*, 22(10):2008–2017, 2012.

2 Nicolas L. Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature biotechnology*, 34(5):525–527, 2016.

3 Brian J. Haas, Arthur L. Delcher, Stephen M. Mount, Jennifer R. Wortman, Roger K. Smith Jr., Linda I. Hannick, Rama Maiti, Catherine M. Ronning, Douglas B. Rusch, Christopher D. Town, et al. Improving the arabidopsis genome annotation using maximal transcript alignment assemblies. *Nucleic acids research*, 31(19):5654–5666, 2003.

4 Steffen Heber, Max Alekseyev, Sing-Hoi Sze, Haixu Tang, and Pavel A. Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics*, 18(suppl_1):S181, 2002.

5 Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L. Salzberg. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome biology*, 14(4):R36, 2013.

6 Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

7 Charity W. Law, Yunshun Chen, Wei Shi, and Gordon K. Smyth. Voom: precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome biology*, 15(2):R29, 2014.

8 Bo Li and Colin N. Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC bioinformatics*, 12(1):323, 2011.

9 Rob Patro, Geet Duggal, Michael I. Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 2017.

10 Rob Patro, Stephen M. Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature biotechnology*, 32(5):462–464, May 2014.

11 Gordon K. Smyth et al. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Stat Appl Genet Mol Biol*, 3(1):3, 2004.

12 Charlotte Soneson, Katarina L. Matthes, Malgorzata Nowicka, Charity W. Law, and Mark D. Robinson. Isoform prefiltering improves performance of count-based methods for analysis of differential transcript usage. *Genome biology*, 17(1):12, 2016.

13 Avi Srivastava, Hirak Sarkar, Nitish Gupta, and Rob Patro. RapMap: a rapid, sensitive and accurate tool for mapping RNA-seq reads to transcriptomes. *Bioinformatics*, 32(12):i192, 2016.

14 Mingxiang Teng, Michael I. Love, Carrie A. Davis, Sarah Djebali, Alexander Dobin, Brenton R. Graveley, Sheng Li, Christopher E. Mason, Sara Olson, Dmitri Pervouchine, et al. A benchmark for RNA-seq quantification pipelines. *Genome biology*, 17(1):74, 2016.

15 Hagen Tilgner, Fereshteh Jahanbani, Tim Blauwkamp, Ali Moshrefi, Erich Jaeger, Feng Chen, Itamar Harel, Carlos D. Bustamante, Morten Rasmussen, and Michael P. Snyder. Comprehensive transcriptome analysis using synthetic long-read sequencing reveals molecular co-association of distant splicing events. *Nature biotechnology*, 33(7):736–742, 2015.

16 Cole Trapnell, Lior Pachter, and Steven L. Salzberg. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, 2009.

17 Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J. Van Baren, Steven L. Salzberg, Barbara J. Wold, and Lior Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, 28(5):511–515, 2010.

18 Jorge Vaquero-Garcia, Alejandro Barrera, Matthew R. Gazzara, Juan Gonzalez-Vallinas, Nicholas F. Lahens, John B. Hogenesch, Kristen W. Lynch, Yoseph Barash, and Juan Valcárcel. A new view of transcriptome complexity and regulation through the lens of local splicing variations. *eLife*, 5:e11752+, February 2016.

19 S. Lawrence Zipursky, Woj M. Wojtowicz, and Daisuke Hattori. Got diversity? wiring the fly brain with dscam. *Trends in biochemical sciences*, 31(10):581–588, 2006.

A Transcriptome Segmentation Algorithm

Algorithm 1 Yanagi's Segments Library Generation

```
Require: Transcriptome Annotation (GTF File), Transcripts Sequences (FASTA Files)
 1: $TxDB \leftarrow makeTxDBFromGTFFile$                          ▷ Preprocessing Step
 2: $DExs \leftarrow disjoinExons(TxDB)$
 3: $TxDB \leftarrow adjustTxDB(TxDB, DExs)$
Step 2 – Segment Graph Construction
 4: procedure CONSTRUCT_SEG_GRAPH($TxDB, g, L$)                   ▷ SgG of gene g
 5:     $G \leftarrow emptygraph$
 6:     $St \leftarrow \phi$
 7:     $prev \leftarrow$ DUMMY_NODE
 8:     for each $Tx \in TxDB(g)$ do                              ▷ For each transcript in gene g
 9:         $loc \leftarrow start(Tx)$
10:         while $loc < end(Tx)$ do                             ▷ Iterate till the end of transcript
11:             $gr \leftarrow GenomicRange(Tx, loc, loc + L)$
12:             $Exs \leftarrow exons[TxDB(gr)]$
13:             $w, loc_{new} \leftarrow$ REFINE_NODE($Exs, loc, L$)          ▷ Node refinement step
14:             $node \leftarrow getOrCreateNode(G, < Exs, loc, w >)$
15:             $Txs(node) \leftarrow Txs(node) + Tx$
16:             $Next(prev) \leftarrow Next(prev) + node$                ▷ Make an edge
17:             if $Txs(prev) \neq Txs(node)$ then                   ▷ Mark branches in graph
18:                 for each $n \in Next(prev)$ do
19:                     $St \leftarrow St + n$
20:             $prev \leftarrow node$                                ▷ Advance loop
21:             $loc \leftarrow loc_{new}$
22:     return $G, St$                                             ▷ The SgG of gene g
Step 3 – Segments Library Generation
23: procedure GENERATE_SEGMENTS($G, St$)
24:     for each $node \in St$ do                               ▷ Iterate over branching points in G
25:         $seg \leftarrow newsegment$                            ▷ Initialize a new segment
26:         $seg.appendNode(node)$
27:         while $|Next(node)| = 1$ do                ▷ Aggregating chain of nodes into the segment
28:             $node \leftarrow Next(node)$
29:             $seg.appendNode(node)$
30:         $outputSegment(seg)$
```

Shrinkage Clustering: A Fast and Size-Constrained Algorithm for Biomedical Applications*

Chenyue W. Hu[1], **Hanyang Li**[2], **and Amina A. Qutub**[3]

1 Department of Bioengineering, Rice University, Houston, TX, USA
2 Department of Bioengineering, Rice University, Houston, TX, USA
3 Department of Bioengineering, Rice University, Houston, TX, USA
aminaq@rice.edu

Abstract

Motivation: Many common clustering algorithms require a two-step process that limits their efficiency. The algorithms need to be performed repetitively and need to be implemented together with a model selection criterion, in order to determine both the number of clusters present in the data and the corresponding cluster memberships. As biomedical datasets increase in size and prevalence, there is a growing need for new methods that are more convenient to implement and are more computationally efficient. In addition, it is often essential to obtain clusters of sufficient sample size to make the clustering result meaningful and interpretable for subsequent analysis.
Results: We introduce *Shrinkage Clustering*, a novel clustering algorithm based on matrix factorization that simultaneously finds the optimal number of clusters while partitioning the data. We report its performances across multiple simulated and actual datasets, and demonstrate its strength in accuracy and speed in application to subtyping cancer and brain tissues. In addition, the algorithm offers a straightforward solution to clustering with cluster size constraints. Given its ease of implementation, computing efficiency and extensible structure, we believe *Shrinkage Clustering* can be applied broadly to solve biomedical clustering tasks especially when dealing with large datasets.

1998 ACM Subject Classification I.5.3 Clustering

Keywords and phrases Clustering, Matrix Factorization, Cancer Subtyping, Gene Expression

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.11

1 Introduction

Cluster analysis is one of the most frequently used unsupervised machine learning methods in biomedicine. The task of clustering is to automatically uncover the natural groupings of a set of objects based on some known similarity relationships. Often employed as a first step in a series of biomedical data analyses, cluster analysis helps to identify distinct patterns in data and suggest classification of objects (e.g. genes, cells, tissue samples, patients) that are functionally similar or related. Typical applications of clustering include subtyping cancer based on gene expression levels [31], classifying protein subfamilies based on sequence similarities [5], distinguishing cell phenotypes based on morphological imaging metrics [26], and identifying disease phenotypes based on physiological and clinical information [22].

* This research was funded in part by NSF CAREER 1150645 and NIH R01 GM106027 grants to A.A.Q., and a HHMI Med-into-Grad fellowship to C.W. Hu.

Many algorithms have been developed over the years for cluster analysis [32], including hierarchical approaches [11] (e.g., *ward-linkage*, *single-linkage*) and partitional approaches that are centroid-based (e.g., *K-means* [15]), density-based (e.g., DBSCAN [7]), distribution-based (e.g., Gaussian mixture models [19]), or graph-based (e.g., Normalized Cut [25]). Notably, nonnegative matrix factorization (NMF) has received a lot of attention in application to cluster analysis, because of its ability to solve challenging pattern recognition problems and the flexibility of its framework [14]. NMF-based methods have been shown to be equivalent to a relaxed *K-means* clustering and Normalized Cut spectral clustering with particular cost functions [6], and NMF-based algorithms have been successfully applied to clustering biomedical data [4].

With few exceptions, most clustering algorithms group objects into a pre-determined number of clusters, and do not inherently look for the number of clusters in the data. Therefore, cluster evaluation measures are often employed and are coupled with clustering algorithms to select the optimal clustering solution from a series of solutions with varied cluster numbers. Commonly used model selection methods for clustering, which vary in cluster quality assessment criteria and sampling procedures, include *Silhouette* [24], *X-means* [23], *Gap Statistic* [29], *Consensus Clustering* [20], *Stability Selection* [13], and *Progeny Clustering* [10]. The drawbacks of coupling cluster evaluation with clustering algorithms include (i) computation burden, since the clustering needs to be performed with various cluster numbers and sometimes multiple times to assess the solution's stability; and (ii) implementation burden, since the integration can be laborious if algorithms are programmed in different languages or are available on different platforms.

Here, we propose a novel clustering algorithm *Shrinkage Clustering* based on symmetric nonnegative matrix factorization notions [12]. Specifically, we utilize unique properties of a hard clustering assignment matrix to simplify the matrix factorization problem and to design a fast algorithm that accomplishes the two tasks of determining the optimal cluster number and performing clustering in one. The *Shrinkage Clustering* algorithm is mathematically straightforward, computationally efficient, and structurally flexible. In addition, the flexible framework of the algorithm allows us to extend it to clustering applications with minimum cluster size constraints.

2 Methods

2.1 Problem Formulation

Let $X = \{X_1, \ldots, X_N\}$ be a finite set of N objects. The task of cluster analysis is to group objects that are similar to each other and separate those that are dissimilar to each other. The completion of a clustering task can be broken down to two steps: (i) deriving similarity relationships among all objects (e.g., Euclidean distance); (ii) clustering objects based on these relationships. The first step is sometimes omitted when the similarity relationships are directly provided as raw data, for example in the case of clustering genes based on their sequence similarities. Here, we assume that the similarity relationships were already derived and are available in the form of a similarity matrix $S_{N\times N}$, where $S_{ij} \in [0,1]$ and $S_{ij} = S_{ji}$. In the similarity matrix, a larger S_{ij} represents more resemblance in pattern or closer proximity in space between X_i and X_j, and vice versa.

Suppose $A_{N\times K}$ is a clustering solution for objects with similarity relationships $S_{N\times N}$. Since we are only considering the case of hard clustering, we have $A_{ik} \in \{0,1\}$ and $\sum_{k=1}^{K} A_{ik} = 1$. Specifically, K is the number of clusters obtained, and A_{ik} takes the value of 1 if X_i belongs to cluster k and takes the value of 0 if it does not. The product of A and its transpose

A^T represents a solution-based similarity relationship $\hat{S}$ (i.e. $\hat{S} = AA^T$), in which $\hat{S}_{ij}$ takes the value of 1 when X_i and X_j are in the same cluster and 0 otherwise. Unlike S_{ij} which can take continuous values between 0 and 1, $\hat{S}_{ij}$ is a binary representation of the similarity relationships indicated by the clustering solution. If a clustering solution is optimal, the solution-based similarity matrix $\hat{S}$ should be similar to the original similarity matrix S if not equal.

Based on this intuition, we formulate the clustering task mathematically as

$$\begin{aligned} &\min_{A} && \|S - AA^T\|_F \\ &\text{subject to} && A_{ik} \in \{0,1\}, \quad \sum_{k=1}^{K} A_{ik} = 1, \quad \sum_{i=1}^{N} A_{ik} \neq 0 \quad . \end{aligned} \tag{1}$$

The goal of clustering is therefore to find an optimal cluster assignment matrix A, which represents similarity relationships that best approximate the similarity matrix S derived from the data. The task of clustering is transformed into a matrix factorization problem, which can be readily solved by existing algorithms. However, most matrix factorization algorithms are generic (not tailored to solving special cases like Function 1), and are therefore computationally expensive.

2.2 Properties and Rationale

In this section, we explore some special properties of the objective Function 1 that lay the ground for *Shrinkage Clustering*. Unlike traditional matrix factorization problems, the solution A we are trying to obtain has special properties, i.e. $A_{ik} \in \{0,1\}$ and $\sum_{k=1}^{K} A_{ik} = 1$. This binary property of A greatly simplifies the objective Function 1 as below.

$$\begin{aligned} &\min_{A} \|S - AA^T\|_F \\ &= \min_{A} \sum_{i=1}^{N} \sum_{j=1}^{N} (S_{ij} - A_i \bullet A_j)^2 \\ &= \min_{A} \sum_{i=1}^{N} \Big(\sum_{j \in \{j|A_i = A_j\}} (S_{ij} - 1)^2 + \sum_{j \in \{j|A_i \neq A_j\}} S_{ij}^2 \Big) \\ &= \min_{A} \sum_{i=1}^{N} \sum_{j \in \{j|A_i = A_j\}} (1 - 2S_{ij}) + \sum_{i=1}^{N} \sum_{j=1}^{N} S_{ij}^2 \end{aligned}$$

Here, A_i represents the ith row of A, and the symbol $\bullet$ denotes the inner product of two vectors. Note that $A_i \bullet A_j$ take binary values of either 0 or 1, because $A_{ik} \in \{0,1\}$ and $\sum_{k=1}^{K} A_{ik} = 1$. In addition, $\sum_{i=1}^{N} \sum_{j=1}^{N} S_{ij}^2$ is a constant that does not depend on the clustering solution A. Based on this simplification, we can reformulate the clustering problem as

$$\min_{A} f(A) = \sum_{i=1}^{N} \sum_{j \in \{j|A_i = A_j\}} (1 - 2S_{ij}). \tag{2}$$

Let's now consider how the value of the objective Function 2 changes when we change the cluster membership of an object X_i. Suppose we start with a clustering solution A, in which X_i belongs to cluster k ($A_{ik} = 1$). When we change the cluster membership of X_i from k to k' with the rest remaining the same, we would obtain a new clustering solution A', in which $A'_{ik'} = 1$ and $A'_{ik} = 0$. Since S is symmetric (i.e. $S_{ij} = S_{ji}$), the value change of the objective Function 2 is

$$\begin{aligned}\triangle f_i &:= f(A') - f(A) \\ &= \sum_{j \in k'}(1-2S_{ij}) - \sum_{j \in k}(1-2S_{ij}) + \sum_{j \in k'}(1-2S_{ji}) - \sum_{j \in k}(1-2S_{ji}) \\ &= 2(\sum_{j \in k'}(1-2S_{ij}) - \sum_{j \in k}(1-2S_{ij})) \quad .\end{aligned} \tag{3}$$

2.3 Shrinkage Clustering: Base Algorithm

Based on the simplified objective Function 2 and its properties with cluster changes (Function 3), we designed a greedy algorithm *Shrinkage Clustering* to rapidly look for a clustering solution A that factorizes a given similarity matrix S. As described in Algorithm 1, *Shrinkage Clustering* begins by randomly assigning objects to a sufficiently large number of initial clusters. During each iteration, the algorithm first removes any empty clusters generated from the previous iteration, a step that gradually shrinks the number of clusters; then it permutes the cluster membership of the object that most minimizes the objective function. The algorithm stops when the solution converges (i.e. no cluster membership permutation can further minimize the objective function), or when a pre-specified maximum number of iterations is reached. *Shrinkage Clustering* is guaranteed to converge to a local optimum as shown in Theorem 1. The main and advantageous feature of *Shrinkage Clustering* is that it shrinks the number of clusters while finding the clustering solution. During the process of permuting cluster memberships to minimize the objective function, clusters automatically collapse and become empty until the optimization process is stabilized and the optimal cluster memberships are found. The number of clusters remaining in the end is the optimal number of clusters, since it stabilizes the final solution. Therefore, *Shrinkage Clustering* achieves both tasks of (i) finding the optimal number of clusters and (ii) finding the clustering memberships.

▶ **Theorem 1.** *Shrinkage Clustering monotonically converges to a (local) optimum.*

Proof. We first demonstrate the monotonically decreasing property of the objective Function 2 in each iteration of the algorithm. There are two steps taken in each iteration: (i) removal of empty clusters; and (ii) permutation of cluster memberships. Step (i) does not change the value of the objective function, because the objective function only depends on non-empty clusters. On the other hand, step (ii) always lowers the objective function, since a cluster membership permutation is chosen based on its ability to achieve the greatest minimization of the objective function. Combing step (i) and (ii), it is obvious that the value of the objective function monotonically decreases with each iteration. Since $\|S - AA^T\|_F \geq 0$ and $\|S - AA^T\|_F = \sum_{i=1}^{N} \sum_{j \in \{j|A_i = A_j\}}(1-2S_{ij}) + \sum_{i=1}^{N} \sum_{j=1}^{N} S_{ij}^2$, the objective function has a lower bound of $-\sum_{i=1}^{N} \sum_{j=1}^{N} S_{ij}^2$. Therefore, a convergence to a (local) optimum is guaranteed, because the algorithm is monotonically decreasing with a lower bound. ◀

2.4 Shrinkage Clustering with Cluster Size Constraints

It is well-known that *K-means* can generate empty clusters when clustering high-dimensional data with over 20 clusters, and *Hierarchical Clustering* often generate tiny clusters with few samples. In practice, clusters of too small a size can sometimes be full of outliers, and they are often not preferred in cluster interpretation since most statistical tests do not apply to

Algorithm 1 *Shrinkage Clustering: Base Algorithm*

```
Input: $S_{N\times N}$ (similarity matrix)
       $K_0$ (intial number of clusters)
Initialization:
a. Generate a random $A_{N\times K_0}$ (cluster assignment matrix)
b. Compute $\tilde{S} = 1 - 2S$
repeat
  1. Remove empty clusters:
      (a) Delete empty columns in $A$ (i.e. $\{j | \sum_{i=1}^{N} A_{ij} = 0\}$)
  2. Permute the cluster membership that minimizes Function 2 the most:
      (a) Compute $M = \tilde{S}A$
      (b) Compute $v$ by $v_i = \min_j M_{ij} - \sum_{j=1}^{K} (M \circ A)_{ij}$, where
          $\circ$ represents the element-wise product (Hadamard product)
      (c) Find the object $\bar{X}$ with the greatest optimization potential,
          i.e. $\bar{X} = \arg\min_i v_i$
      (d) Permute the membership of $\bar{X}$ to $C'$, where $C' = \arg\min_j M_{\bar{X}j}$
until $\sum_{i=1}^{N} v_i = 0$ or reaching max number of iterations
Output: $A$ (cluster assignment)
```

Algorithm 2 *Shrinkage Clustering with Cluster Size Constraints*

```
Additional Input: $\omega$ (minimum cluster size).
Updated Step 1:
  (a) Remove columns in $A$ that contain too few objects, i.e. $\{j | \sum_{i=1}^{N} A_{ij} < \omega\}$
  (b) Reassign objects in these clusters to clusters with the greatest minimization
```

small sample sizes. Though extensions to *K-means* were proposed to solve this issue [3], the attempt to control cluster sizes has not been easy. In contrast, the flexibility and the structure of *Shrinkage Clustering* offers a straightforward and rapid solution to enforcing constraints on cluster sizes. To generate a clustering solution with each cluster containing at least ω objects, we can simply modify Step 1 of the iteration loop in Algorithm 1. Instead of removing empty clusters in the beginning of each iteration, we now remove clusters of sizes smaller than a pre-specified size ω. The base algorithm (Algorithm 1) can be viewed as a special case of $w = 0$ in the size-constrained *Shrinkage Clustering* algorithm.

3 Results

3.1 Experiments on Similarity Data

3.1.1 Testing with Simulated Similarity Matrices

We first use simulated similarity matrices to test the performance of *Shrinkage Clustering* and to examine its sensitivity to the initial parameters and noise. As a proof of concept, we generate a similarity matrix S directly from a known cluster assignment matrix A by $S = AA^T$. Here, the cluster assignment matrix $A_{100\times 5}$ is randomly generated to consist of 100 objects grouped into 5 clusters with unequal cluster sizes (i.e. 15, 17, 20, 24 and 24 respectively). The similarity matrix $S_{100\times 100}$ generated from the product of A and A^T therefore represents an ideal case, where there is no noise, since each entry of S only takes a binary value of either 0 or 1.

Table 1 Clustering results of simulated similarity matrices with varying size constraints (ω), where C is the cluster generated by *Shrinkage Clustering*.

True Label	$\omega = 0$					$\omega = 20$				$\omega = 25$	
	C1	C2	C3	C4	C5	C1	C2	C3	C4	C1	C2
Cluster 1	0	0	24	0	0	0	24	0	0	0	24
Cluster 2	15	0	0	0	0	15	0	0	0	15	0
Cluster 3	0	0	0	24	0	0	0	24	0	0	24
Cluster 4	0	17	0	0	0	17	0	0	0	17	0
Cluster 5	0	0	0	0	20	0	0	0	20	20	0

We apply *Shrinkage Clustering* to this simulated similarity matrix S with 20 initial random clusters and repeat the algorithm for 1000 times. Each run, the algorithm accurately generates 5 clusters with cluster assignments $\tilde{A}$ in perfect match with the true cluster assignments A (an example shown in Table 1 under $\omega = 0$), demonstrating the algorithm's ability to perfectly recover the cluster assignments in a non-noisy scenario. The shrinkage paths of the first 5 runs (Figure 1A) illustrate that most runs start around a number of 20 clusters, and all of them shrink down gradually to a final number of 5 clusters when the solution reaches an optimum. We then test whether the algorithm is sensitive to the initial number of clusters (K_0) by running it with K_0 ranging from 5 (true number of clusters) to 100 (maximum number of clusters). In each case, the true cluster structure is recovered perfectly, demonstrating the robustness of the algorithm to different initial cluster numbers. The shrinkage paths in Figure 1B clearly show that in spite of starting with various initial numbers of clusters, all paths converge to the same number of clusters at the end.

Next, we investigate the effects of size constraints on *Shrinkage Clustering*'s performance by varying ω from 1 to 5, 10, 20 and 25. The algorithm is repeated 50 times in each case. We find that as long as ω is smaller than the true minimum cluster size (i.e. 15), the size constrained algorithm can perfectly recover the true cluster assignments A in the same way as the base algorithm. Once ω exceeds the true minimum cluster size, clusters are forced to merge and therefore result in a smaller number of clusters (example clustering solutions of $\omega = 20$ and $\omega = 25$ shown in Table 1). In these cases, it is impossible to find the true cluster structure because the algorithm starts off with fewer clusters than the true number of clusters and it works uni-directionally (i.e. only shrinks). Besides enabling supervision on the cluster sizes, size-constrained *Shrinkage Clustering* is also computationally advantageous. Figure 1C shows that a larger ω results in fewer iterations needed for the algorithm to converge, and the effect reaches a plateau once ω reaches certain sizes (e.g. $\omega = 10$ in this case). The shrinkage paths (Figure 1D) show that it is the reduced number of iterations at the beginning of a run that speeds up the entire process of solution finding when ω is large.

In reality, it is rare to find a perfectly binary similarity matrix similar to what we generated from a known cluster assignment matrix. There is always a certain degree of noise clouding our observations. To investigate how much noise the algorithm can tolerate in the data, we add a layer of Gaussian noise over the simulated similarity matrix. Since $S_{ij} \in \{0, 1\}$, we create a new similarity matrix S^N containing noise defined by

$$S^N = \begin{cases} |\varepsilon| & \text{if } S_{ij} = 0 \\ 1 - |\varepsilon| & \text{if } S_{ij} = 1 \end{cases},$$

where $\varepsilon \sim N(0, \sigma^2)$. Figure 1E illustrates the changes of the similarity distribution density as the standard deviation σ varies from 0 to 0.5. When $\sigma = 0$ (i.e. no noise), S^N is

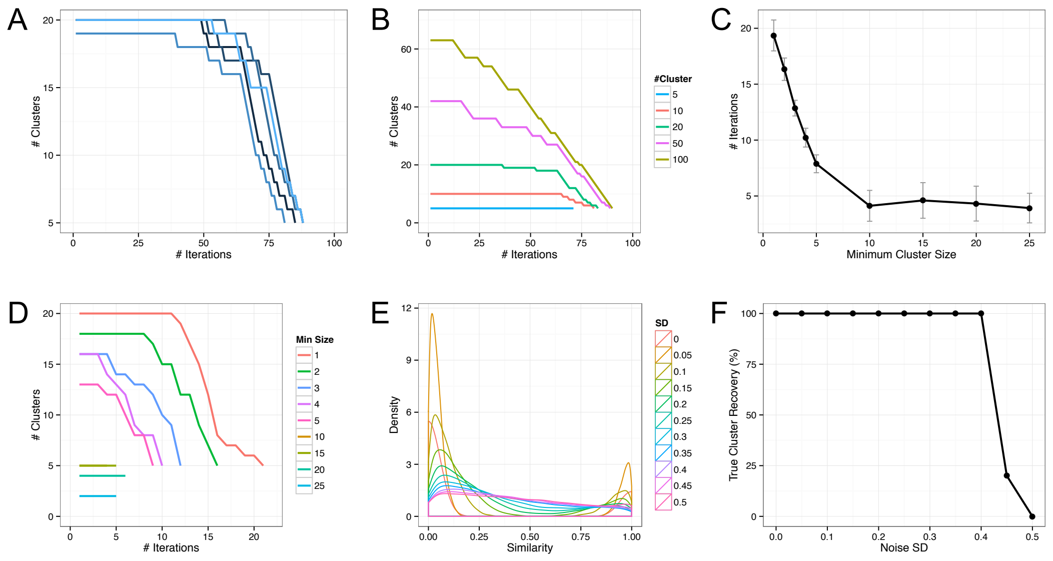


Figure 1 Testing results on simulated similarity data: (A) The first 5 shrinkage paths from running base algorithm with different initiations. (B) Example shrinkage paths with varying initial cluster numbers of 5, 10, 20, 50 and 100. (C) The average number of iterations spent with ω taking values of 1 to 5, 10, 15, 20 and 25. (D) Example shrinkage paths for ω of 1 to 5, 10, 15, 20 and 25 (path of $\omega = 10$ is in overlap with $\omega = 15$). (E) The similarity distribution for ε from 0 to 0.5. (F) The probability of successfully recovering the underlying cluster structure against different noise levels.

Bernoulli distributed. As σ becomes larger and larger, the bimodal shape is flattened by noise. When $\sigma = 0.5$, approximately 32% of the similarity relationships are reversed, and hence observations have been perturbed too much to infer the underlying cluster structure. The performances of *Shrinkage Clustering* in these noisy conditions are shown in Figure 1F. The algorithm proves to be quite robust against noise, as the true cluster structure is 100% recovered in all conditions except for when $\sigma > 0.4$.

3.1.2 Biological Case Study: TCGA Dataset

To illustrate the performance of *Shrinkage Clustering* on real biological similarity data, we apply the algorithm to subtyping tumors from the Cancer Genome Atlas (TCGA) dataset [27]. Derived from the TCGA database, the dataset includes 293 samples from 3 types of cancers, which are Breast Invasive Carcinoma (BRCA, 207 samples), Glioblastoma Multiforme (GBM, 67 samples) and Lung Squamous Cell Carcinoma (LUSC, 19 samples). The data is presented in the form of a similarity matrix, which integrates information from the gene expression levels, DNA methylation and copy number aberration. Since the similarity scores from the TCGA dataset are in general skew to 1, we first normalize the data by shifting its mean around 0.5 and by bounding values that are greater than 1 and smaller than 0 to 1 and 0 respectively. We then perform *Shrinkage Clustering* to cluster the cancer samples, the result of which is shown in comparison to the true cancer types (Table 2). We can see that the algorithm generates three clusters, successfully predicting the true number of cancer types contained in the data. The clustering assignments also demonstrate high accuracy, as the majority of samples are correctly clustered with samples of the same type.

Table 2 Clustering results of the TCGA dataset, where the clustering assignments from *Shrinkage Clustering* are compared against the three known tumor types.

	Cluster 1	Cluster 2	Cluster 3
BRCA	3	204	0
GBM	0	0	67
LUSC	17	2	0

Table 3 Performances of *Shrinkage Clustering* on Simulated, Iris and Wine data, where the clustering assignments are compared against the three simulated centers, three Iris species and three wine types respectively.

Simulated				Iris			Wine			
Center	C1	C2	C3	Species	C1	C2	Type	C1	C2	C3
$(-2, 2)$	0	49	1	*setosa*	50	0	1	0	59	0
$(-2, -2)$	0	1	49	*versicolor*	0	50	2	59	6	0
$(2, 0)$	50	0	0	*virginica*	0	50	3	0	6	48

3.2 Experiments on Feature-based Data

3.2.1 Testing with Simulated and Standardized Data

Since similarity matrices are not always available in most clustering applications, we now test the performance of *Shrinkage Clustering* using feature-based data that does not directly provide the similarity information between objects. To run *Shrinkage Clustering*, we first convert the data to a similarity matrix using $S = \exp(-(D(X)/\beta\sigma)^2)$, where $[D(X)]_{ij}$ is the Euclidean distance between X_i and X_j, σ is the standard deviation of $D(X)$, and $\beta = E(D(X)^2)/\sigma^2$. The same conversion method is used for all datasets in the rest of this paper.

As a proof of concept, we first generate a simulated three-cluster two-dimensional data set by sampling 50 points for each cluster from bivariate normal distributions with a common identity covariance matrix around centers at $(-2, 2)$, $(-2, 2)$ and $(0, 2)$ respectively. The clustering result from *Shrinkage Clustering* is shown in Table 3, where the algorithm successfully determines the existence of 3 clusters in the data and obtains a clustering solution with high accuracy.

Next, we test the performance of *Shrinkage Clustering* using two real data sets, the Iris [8] and the wine data [1], both of which are frequently used to test clustering algorithms; and they can be downloaded from the University of California Irvine (UCI) machine learning repository [2]. The clustering results from *Shrinkage Clustering* for both datasets are shown in Table 3, where the clustering assignments are compared to the true cluster memberships of the Iris and the wine samples respectively. In application to the wine data, *Shrinkage Clustering* successfully identifies a correct number of 3 wine types and produces highly accurate cluster memberships. For the Iris data, though the algorithm generates two instead of three clusters, the result is acceptable because the species *versicolor* and *virginica* are known to be hardly distinguishable given the features collected.

Table 4 Clustering Accuracy Comparison of the seven algorithms based on NMI (Normalized Mutual Information), Rand Index, F1 score and K (the optimal cluster number).

Type	Metric	Shrinkage	Spectral	K-means	Hierarchical	PAM	DBSCAN	Affinity
BCWD	NMI	0.50	0.29	0.46	0.09	0.50	0.29	0.27
	Rand	0.77	0.68	0.75	0.55	0.77	0.64	0.52
	F1	0.80	0.69	0.79	0.69	0.80	0.72	0.22
	K (2)	2	2	2	2	2	3	21
AIBT	NMI	0.56	0.20	0.58	0.17	0.54	0.31	0.40
	Rand	0.79	0.68	0.80	0.37	0.78	0.64	0.76
	F1	0.61	0.39	0.62	0.40	0.59	0.43	0.26
	K (4)	4	4	4	4	4	5	31

3.2.2 Biological Case Study 1: Breast Cancer Wisconsin Diagnostic (BCWD)

The BCWD dataset [28, 17] contains 569 breast cancer samples (357 benign and 212 malignant) with 30 characteristic features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The dataset is available on the UCI machine learning repository [2] and is one of the most popularly tested dataset for clustering and classification. Here, we apply *Shrinkage Clustering* to the data and compare its performance against six commonly used clustering methods: *Spectral Clustering* [33], *K-means* [15], *Hierarchical Clustering* [11] (Ward's method [30]), *PAM* [16], *DBSCAN* [7] and *Affinity Propagation* [9]. Since *K-means*, *Spectral Clustering*, *Hierarchical Clustering* and *PAM* do not inherently determine the optimal cluster number and require the cluster number as an input, we first run these algorithms with cluster numbers from 2 to 10, and then use the mean *Silhouette* width as the criterion to select the optimal cluster number. The accuracy of all clustering solutions is evaluated using four metrics: Normalized Mutual Information (NMI) [18], Rand Index [18], F1 score [18], and the optimal cluster number (K).

The performance results (Table 4) show that *Shrinkage Clustering* correctly predicts a 2 cluster structure from the data and generates the clustering assignments with high accuracy. Notably, *Shrinkage Clustering* is the only algorithm that correctly estimates the optimal cluster number among the three algorithms (together with *DBSCAN* and *Affinity Propagation*) that inherently determine cluster numbers. When comparing the cluster assignments against the true cluster memberships, we can see that *Shrinkage Clustering* is one of the two best performers (with *PAM*) across all accuracy metrics.

3.2.3 Biological Case Study 2: Allen Institute Brain Tissue (AIBT)

The AIBT dataset [21] contains RNA sequencing data of 377 samples from four types of brain tissues, i.e. 99 samples of temporal cortex, 91 samples of parietal cortex, 93 samples of cortical white matter, and 94 samples hippocampus isolated by macro-dissection. For each sample, the expression levels of 50282 genes are included as features, and each feature is normalized to have a mean of 0 and a standard deviation of 1 prior to testing. In contrast to the previous case study, the AIBT data is much larger in size with significantly more features being measured. Therefore, this would be a great example to test both the accuracy and the speed of clustering algorithms in face of greater data sizes and higher dimensions.

Similar to the previous case study, we apply *Shrinkage Clustering* and the six commonly used clustering algorithms to the data, and use mean *Silhouette* width to select the optimal

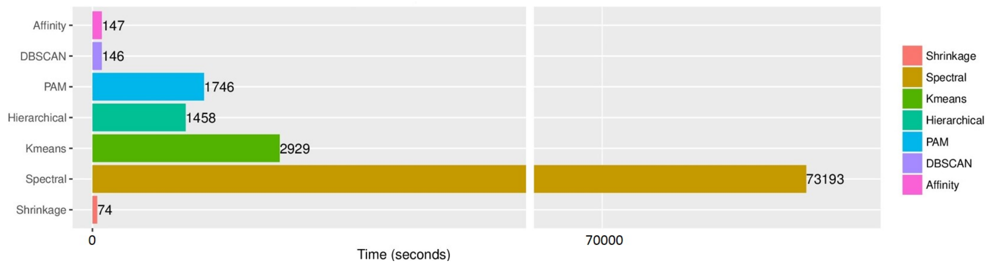


Figure 2 Computation time of all clustering algorithms in application to the AIBT data.

cluster number for algorithms that do not inherently determine the cluster number. The performances of all seven algorithms measured across the four accuracy metrics (i.e. NMI, Rand, F1, K) are shown in Table 4. Among the three algorithms with built-in cluster number determination, *Shrinkage Clustering* is the only algorithm that accurately predicts a 4 cluster structure, whereas *DBSCAN* and *Affinity Propagation* both over-estimate the total number of clusters present in the data. In addition, *Shrinkage Clustering* is the second best performer among all seven algorithms in terms of clustering quality, with comparable accuracy to the top performer (*K-means*).

Next, we record and compare the speed of the seven algorithms for clustering the data. The speed comparison results, shown in Figure 2, demonstrate the unparalleled speed of *Shrinkage Clustering* compared to the rest of the algorithms. *Shrinkage Clustering* is twice as fast as *DBSCAN* and *Affinity Propagation*, and it is faster than *Hierachical*, *PAM* and *K-means* by more than 20 times. In particular, the same data that takes *Shrinkage Clustering* only 73 seconds to cluster can take *Spectral clustering* more than 20 hours.

4 Discussion

From the biological case studies, we showed that *Shrinkage Clustering* is computationally advantageous in speed with high clustering accuracy comparable to the best performers of the six commonly used clustering algorithms. This advantage in speed mainly comes from the fact that *Shrinkage Clustering* integrates the clustering of the data and the determination of the optimal cluster number into one seamless process, so the algorithm only needs to run once in order to complete the clustering task. In contrast, algorithms like *K-means*, *PAM* and *Spectral Clustering* perform clustering on a single cluster number basis, therefore they need to be repeatedly run for all cluster numbers of interest before a clustering evaluation method can be applied. Notably, the clustering evaluation method *Silhouette* that we used in this experiment does not perform any repetitive clustering validation and therefore is a much faster method compared to other commonly used methods that require repetitive validation [10]. This means that *Shrinkage Clustering* would have an even greater advantage in computation speed compared to the methods tested in this paper if we use a cluster evaluation method that has a repetitive nature (e.g. *Consensus Clustering*, *Gap Statistics*, *Stability Selection*).

One prominent feature of *Shrinkage Clustering* is its flexibility to solve semi-supervised clustering problems, i.e. its ability to add constraints to the minimum cluster size. The size constraints can help prevent generating empty or tiny clusters (which are often observed in *Hierarchical Clustering* and sometimes in *K-means* applications), and can produce clusters of sufficiently large sample sizes as required by the user. This is particularly useful when we need to perform subsequent statistical analyses based on the clustering solution, since clusters of too small a size can make a statistical testing infeasible. For example, one application of cluster analysis in clinical studies is identifying subpopulations of cancer patients based on their gene expression levels, which is usually followed with a survival analysis to determine the prognostic value of the gene expression patterns. In this case, clusters that contain too few patients can hardly generate any significant or meaningful patient outcome comparison. In addition, it is difficult to take actions based on tiny patient clusters (e.g. in the context of designing clinical trials), since these clusters are hard to validate.

In summary, we developed a new NMF-based clustering method, *Shrinkage Clustering*, which shrinks the number of clusters to an optimum while simultaneously optimizing the cluster memberships. The algorithm performed with high accuracy on both simulated and actual data, exhibited excellent robustness to noise, and demonstrated superior speeds compared to some of the commonly used algorithms. The base algorithm has also been extended to accommodate requirements on minimum cluster sizes, which can be particularly beneficial to clinical studies and the general biomedical community. The size-constrained extension also illustrates the flexibility of the *Shrinkage Clustering* algorithm framework and the ease of adapting it towards semi-supervised learning tasks. Interesting future research directions include exploring *Shrinkage Clustering*'s capability to deal with missing data or incomplete similarity matrices, as well as extending the algorithm to handle semi-supervised clustering tasks with must-link and cannot-link constraints.

References

1 S. Aeberhard, D. Coomans, and O. De Vel. Comparison of classifiers in high dimensional settings. *Dept. Math. Statist., James Cook Univ., North Queensland, Australia, Tech. Rep*, (92-02), 1992.

2 Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

3 P. S. Bradley, K. P. Bennett, and Ayhan Demiriz. Constrained k-means clustering. *Microsoft Research, Redmond*, pages 1–8, 2000.

4 Jean-Philippe Brunet, Pablo Tamayo, Todd R. Golub, and Jill P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the national academy of sciences*, 101(12):4164–4169, 2004.

5 Elisa Boari de Lima, Wagner Meira Júnior, and Raquel Cardoso de Melo-Minardi. Isofunctional protein subfamily detection using data integration and spectral clustering. *PLoS Comput Biol*, 12(6):e1005001, 2016.

6 Chris Ding, Xiaofeng He, and Horst D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 606–610. SIAM, 2005.

7 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

8 Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

9 Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

10 Chenyue W. Hu, Steven M. Kornblau, John H. Slater, and Amina A. Qutub. Progeny clustering: A method to identify biological phenotypes. *Scientific reports*, 5, 2015.

11 Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

12 Da Kuang, Chris Ding, and Haesun Park. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 106–117. SIAM, 2012.

13 Tilman Lange, Volker Roth, Mikio L. Braun, and Joachim M. Buhmann. Stability-based validation of clustering solutions. *Neural computation*, 16(6):1299–1323, 2004.

14 Tao Li and Chris H. Q. Ding. Nonnegative matrix factorizations for clustering: A survey., 2013.

15 James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. California, USA, 1967.

16 Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. Cluster: cluster analysis basics and extensions. *R package version*, 1(2):56, 2012.

17 Olvi L Mangasarian, W. Nick Street, and William H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.

18 Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge University Press, 2008.

19 Geoffrey J. McLachlan and Kaye E. Basford. Mixture models. inference and applications to clustering. *Statistics: Textbooks and Monographs, New York: Dekker, 1988*, 1, 1988.

20 Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine learning*, 52(1-2):91–118, 2003.

21 Thomas J. Montine, Joshua A. Sonnen, Kathleen S. Montine, Paul K. Crane, and Eric B. Larson. Adult changes in thought study: dementia is an individually varying convergent syndrome with prevalent clinically silent diseases that may be modified by some commonly used therapeutics. *Current Alzheimer Research*, 9(6):718–723, 2012.

22 Wendy C. Moore, Deborah A. Meyers, Sally E. Wenzel, W. Gerald Teague, Huashi Li, Xingnan Li, Ralph D'Agostino Jr., Mario Castro, Douglas Curran-Everett, Anne M. Fitzpatrick, et al. Identification of asthma phenotypes using cluster analysis in the severe asthma research program. *American journal of respiratory and critical care medicine*, 181(4):315–323, 2010.

23 Dan Pelleg, Andrew W. Moore, et al. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *ICML*, pages 727–734, 2000.

24 Peter J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

25 Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

26 John H Slater, James C. Culver, Byron L. Long, Chenyue W. Hu, Jingzhe Hu, Taylor F Birk, Amina A. Qutub, Mary E. Dickinson, and Jennifer L. West. Recapitulation and modulation of the cellular architecture of a user-chosen cell of interest using cell-derived, biomimetic patterning. *ACS nano*, 9(6):6128–6138, 2015.

27 Nora Speicher and Thomas Lengauer. *Towards the identification of cancer subtypes by integrative clustering of molecular data.* PhD thesis, Universität des Saarlandes Saarbrücken, 2012.

28 W. Nick Street, William H. Wolberg, and Olvi L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *IS&amp;T/SPIE's Symposium on Electronic Imaging: Science and Technology*, pages 861–870. International Society for Optics and Photonics, 1993.

29 Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

30 Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

31 Pratyaksha Wirapati, Christos Sotiriou, Susanne Kunkel, Pierre Farmer, Sylvain Pradervand, Benjamin Haibe-Kains, Christine Desmedt, Michail Ignatiadis, Thierry Sengstag, Frédéric Schütz, et al. Meta-analysis of gene expression profiles in breast cancer: toward a unified understanding of breast cancer subtyping and prognosis signatures. *Breast Cancer Research*, 10(4):R65, 2008.

32 Christian Wiwie, Jan Baumbach, and Richard Röttger. Comparing the performance of biomedical clustering methods. *Nature Methods*, 12(11):1033–1038, 2015.

33 Achim Zeileis, Kurt Hornik, Alex Smola, and Alexandros Karatzoglou. kernlab-an S4 package for kernel methods in R. *Journal of statistical software*, 11(9):1–20, 2004.

Sparsification Enables Predicting Kissing Hairpin Pseudoknot Structures of Long RNAs in Practice

Hosna Jabbari[1], Ian Wark[1], Carlo Montemagno[1], and Sebastian Will[4]

1 **Ingenuity Lab, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada**
2 **Ingenuity Lab, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada**
3 **Ingenuity Lab, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada**
4 **Theoretical Biochemistry Group (TBI), Institute for Theoretical Chemistry, University of Vienna, Vienna, Austria**
`will@tbi.univie.ac.at`

Abstract

While computational RNA secondary structure prediction is an important tool in RNA research, it is still fundamentally limited to pseudoknot-free structures (or at best very simple pseudoknots) in practice. Here, we make the prediction of complex pseudoknots – including kissing hairpin structures – practically applicable by reducing the originally high space consumption. For this aim, we apply the technique of sparsification and other space-saving modifications to the recurrences of the pseudoknot prediction algorithm by Chen, Condon and Jabbari (CCJ algorithm). Thus, the theoretical space complexity of free energy minimization is reduced to $\Theta(n^3 + Z)$, in the sequence length n and the number of non-optimally decomposable fragments ("candidates") Z. The sparsified CCJ algorithm, `sparseCCJ`, is presented in detail. Moreover, we provide and compare three generations of CCJ implementations, which continuously improve the space requirements: the original CCJ implementation, our first modified implementation, and our final sparsified implementation. The two latest implementations implement the established HotKnots DP09 energy model. In our experiments, using 244GB of RAM, the original CCJ implementation failed to handle sequences longer than 195 bases; `sparseCCJ` handles our pseudoknot data set (up to about length 400 bases) in this space limit. All three CCJ implementations are available at `https://github.com/HosnaJabbari/CCJ`.

1998 ACM Subject Classification J.3 Life and Medical Sciences

Keywords and phrases RNA, secondary structure prediction, pseudoknots, space efficiency, sparsification

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.12

1 Introduction

Computational RNA secondary structure prediction has become an indispensable tool in the research on non-coding RNAs. Such RNAs perform essential roles – most prominently in regulating gene expression – in all kingdoms of live, in many cases mediated by their three-dimensional structures [10]. Despite the ubiquity of pseudoknots in these RNAs, most often only pseudoknot-free structure prediction methods are applied in biological research – severely limiting the practical capabilities to correctly predict, recognize and compare pseudoknotted structures.

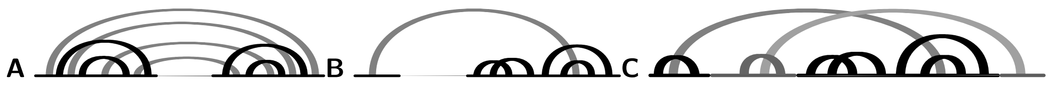


Figure 1 Examples of TGB and CCJ structures. Each arc represents a band of base pairs, which cross other bands. (A) TGB structure with two left, two right, and four middle bands; (B) TGB with nested substructure; (C) CCJ structure composed of two TGB structures (see decomposition of P).

The fundamental cause of this limitation is the computational complexity of pseudoknot prediction – an NP-hard problem [1, 7, 8]. Thus, the high complexity of pseudoknot prediction can be overcome only by either heuristics without optimality-guarantees or restrictions on the predictable pseudoknot class. In comparison, predicting minimum free energy (MFE) structures without pseudoknots is easy (due to tree-like dependencies): a pseudoknot-free secondary structure is either closed by a base pair connecting the first and last base in the sequence, or can be partitioned into two independent substructures on a prefix and suffix of the sequence, where energies of substructures add up to the total energy. As a result of the simple decomposition scheme, the MFE pseudoknot-free secondary structure prediction problem is solved by dynamic programming in $\Theta(n^3)$ time and $\Theta(n^2)$ space for standard energy loop models [12].

The most general dynamic programming algorithm for MFE prediction of pseudoknotted structures, Pknots, was proposed by Rivas and Eddy [14]. Pknots is a complex dynamic programming algorithm with time complexity of $\Theta(n^6)$, and space complexity of $\Theta(n^4)$. There are algorithms for predicting MFE pseudoknotted secondary structures that run in $\Theta(n^5)$ time and $\Theta(n^4)$ space [18, 8]. These algorithms handle a severely limited class compared to the Rivas and Eddy's algorithm. All can handle H-type pseudoknots, and some can handle kissing hairpin structures when these do not have arbitrary nested substructures [18]. There are also some algorithms that run in $\Theta(n^4)$ time [13, 8]; these handle classes of structures that are even more restricted than the $\Theta(n^5)$ algorithms. However, none of these algorithms handle kissing hairpin structures with arbitrary nested substructures. This is a serious, practically relevant limitation, given the biological importance of such structures [4, 9, 19].

We previously proposed a novel MFE-based algorithm, called CCJ [5], which significantly expands the class of structures that can be handled in $O(n^5)$ time. We described a more general method of formulating the dynamic programming recurrences for prediction of pseudoknotted RNA secondary structures that cover gapped regions. To improve the time complexity to $O(n^5)$ we introduced two new ideas into the dynamic programming recurrences: (i) a new class of structures called TGB structures (see Figs. 1A and 1B), with at most three groups of bands, and (ii) recurrences that handle TGB structures by transferring to the left, right, middle or the outer bands. By overlaying TGB structures (Fig. 1C), CCJ covers H-type pseudoknotted structures, kissing hairpins, and chains of four interleaved stems; moreover it recursively handles nested substructures of these types; we simply called this class of structures, *CCJ structures.*

We previously compared CCJ's prediction accuracy versus HotKnots V2.0 [2] and IPknot [16], and showed that CCJ outperforms these algorithms on some of our data sets [6]. While CCJ predicts such complex pseudoknotted secondary structures in $\Theta(n^5)$ time and $\Theta(n^4)$ space, which is a significant improvement over the existing MFE-based algorithms [18, 8], in practice it fails to run on sequences longer than 195 bases even given 244GB of RAM (as in our experiments).

Here, we apply the technique of sparsification [20, 3] to the CCJ algorithm with the main goal to reduce its extreme space complexity of $\Theta(n^4)$. Devising sparsified recurrences of

CCJ, resulting in the algorithm `sparseCCJ`, we improve the space complexity to $\Theta(n^3 + Z)$, where Z is the total number of candidates. This complexity is the result of replacing all four-dimensional dynamic programming matrices by (a constant number of) three-dimensional matrix slices and lists of *candidates*. This is still sufficient to calculate exactly the same optimal solutions as before, which can be shown based on inverse triangle inequality. The number of candidates is expected to be much smaller than the number of replaced matrix entries; moreover it cannot be larger.

The space-efficient retrieval of the optimal structure from this algorithm is enabled by implementing our recently presented technique of space-efficient sparse traceback [21]; this method requires additional space for trace arrows, but keeps the number of trace arrows low due to techniques like garbage collection.

Previous applications of the sparsification technique to RNA structure prediction discussed pseudoknot-free methods [20, 3] or used pseudoknotted methods with simplified energy models (base pair maximization only) [11]. Sparsified RNA–RNA interaction prediction [15] is the most complex case of structure prediction so far that was implemented for a realistic interaction energy model. While space-efficient sparsification is discussed in the paper, the space-efficient variant of the implementation could not recover the optimal interaction structure by traceback.

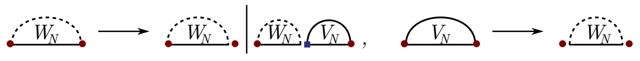
Contributions

We present – to the best of our knowledge – the first space-efficient sparsification of any pseudoknot prediction algorithm that uses a realistic, practically relevant pseudoknot energy model (with according parametrization). Moreover, we sparsify the particularly powerful pseudoknot prediction algorithm CCJ that covers the biologically important kissing hairpins. We implemented – in addition to the original CCJ implementation – a first space-improved CCJ variant and the resulting sparsified algorithm `sparseCCJ`, both using the current HotKnots DP09 energy model [2]. By comparing all three CCJ implementations, we study the (length-dependent) impact of various space savings and finally show that sparsification significantly improves the space requirements over non-sparse implementations of CCJ.

2 The original CCJ pseudoknot prediction algorithm

The original CCJ algorithm [5] is a dynamic programming algorithm (DP) that minimizes the free energy over all CCJ structures for a given input sequence S. As stated in the introduction, CCJ structures comprise kissing hairpins and chains of four interleaved stems, which can recursively contain CCJ structures as substructures. The optimal CCJ structure is then determined by standard traceback through the DP matrices.

Generally, DP algorithms can be described by presenting the recurrences that are used to calculate the entries of their DP matrices. In the case of RNA structure prediction, the DP matrices store minimum free energy (MFE) values for sequence fragments (under specific conditions). The reccurrences correspond to decompositions of these fragments such that the matrix entries can be recursively inferred from energies of smaller subproblems. For example, assuming an energy value of -1 for each canonical base pair, i.e. C–G, A–U or G–U, the MFE structure of an RNA sequence, S, can be found using matrices W_N and V_N, where the respective entries $W_N(i,j)$ and $V_N(i,j)$ are decomposed as follows



(1)

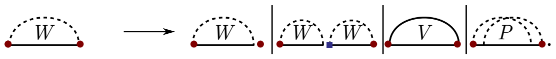
These grammar-rules represent a complete case distinction of possible structures. In the example of Eq. (1), $W_N(i,j)$ corresponds to the MFE structure from base i to base j; this structure can be decomposed according to Eq. (1), since either j is unpaired (left case) or j is paired to some inner position (recursion to V_N, in which solid arc represent a base pair); the closed structure corresponding to $V_N(i,j)$ is reduced to $W_N(i+1,j-1)$ (rightmost case). Moreover, the grammar rules allow – almost mechanically – inferring the recursion equations for base pair energy minimization. The graphical notation is designed to encode the required information: red dots on the right side always correspond to red dots on the left side of the rule, while blue squares mark *free* position indices. Generally, our recursions marginalize (i.e. minimize) over the recursion cases and the free indices in their respective range limited by the fixed indices. Thus, we translate the rules of Eq. (1) to

$$W_N(i,j) = \min\{W_N(i,j-1), \min_{i\leq k<j} W_N(i,k-1) + V_N(k,j)\},$$
$$V_N(i,j) = W_N(i+1,j-1) - 1 \qquad \text{if } S_i\text{–}S_j \text{ is canonical, } V_N(i,j) = \infty \text{ otherwise.}$$

In our discussion of CCJ and its sparsification, this level of presentation allows us to focus on the sparsification and avoid distracting details like the exact added energy contributions in each single step, which is not necessary for understanding the sparsification.

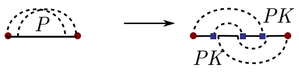
While pseudoknot-free recursions generally use only fragments that are connected at the sequence level, the CCJ algorithm requires 'gapped' fragments, where two subsequences are disconnected by a gap. Consequently, defining such fragments requires four sequence positions in total.

The MFE of the CCJ structure for subsequence $S_{i..l}$ is calculated in $W(i,l)$, which is decomposed as



The recurrence for $V(i,l)$ handles different types of loops closed by i and l; $P(i,l)$ is the minimum free energy of a CCJ pseudoknot for region $[i,l]$.

$P(i,l)$ is decomposed by the rule

 (2)

into two TGB structures (with MFEs in the PK matrix). Representing TGB structures requires using gapped fragments.

As indicated by the three blue boxes (free indices) in Eq. (2), each entry of P is minimized over three indices. Thus, already this step bounds the time and space complexity of the CCJ algorithm to $O(n^5)$ and $O(n^4)$ respectively.

$PK(i,j,k,l)$ is the MFE over all TGB structures of the gapped fragment $[i,j]\cup[k,l]$. Note that such structures have the additional restrictions: the positions i and l must be involved in some base pairs, which are not part of nested substructures; moreover, some base pair of a TGB structure must span the gap. The recurrence for PK uses terms P_L, P_M, P_O, and P_R, which (put informally) handle bands on the left, middle and right groups of the TGB structure, respectively. Both P_M and P_O are needed to handle bands in the middle group. The matrix entries $P_L(i,j,k,l)$, $P_M(i,j,k,l)$, $P_R(i,j,k,l)$, $P_O(i,j,k,l)$ are decomposed as illustrated in Fig. 2, in which WP, handles nested substructures in a pseudoloop. (For invalid index combinations, the matrix entries are set to $+\infty$, and do not have to be stored.)

Each of the matrices P_L, P_R, P_M, and P_O requires a base pair between the two ends at the respective positions left, right, middle, or outer. Each such matrix distinguishes the three

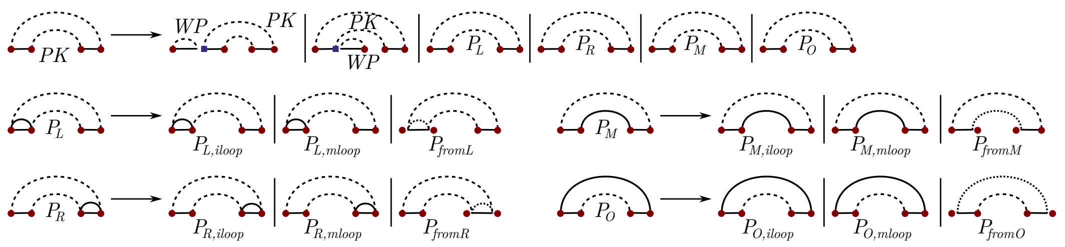


Figure 2 Decompositions for $PK(i,j,k,l)$, $P_L(i,j,k,l)$, $P_M(i,j,k,l)$, $P_R(i,j,k,l)$, $P_O(i,j,k,l)$ in grammar-rule like graphical notation.

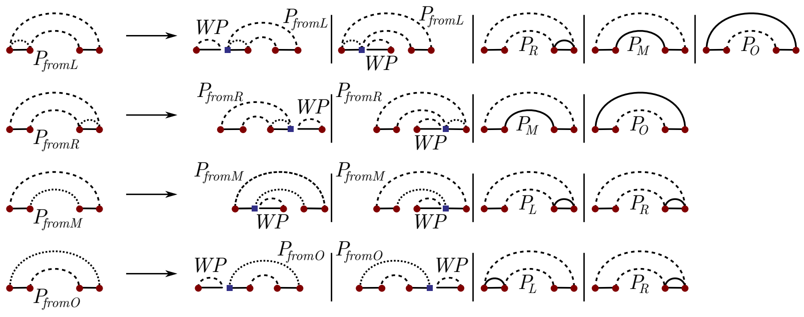
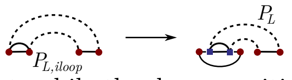


Figure 3 Decompositions of $P_{fromX}(i,j,k,l)$ for $X \in \{L,R,M,O\}$, which handle transitions from P_X in other matrices P_Y in graphical notation.

cases that this base pair closes an interior loop or a multi-loop or is the inner border of a band. In the latter case, the respective terms P_{fromX} ($X \in \{L,R,M,O\}$) handle transitions from base pairs in one group to base pairs in some other group (see Fig. 3).

Whenever we change a band via one of the matrices P_{fromX}, we must allow for nested substructures around the band. This is handled by the first two cases of the respective recurrence P_{fromX}. Moreover, in P_{fromX} we recurse to matrices $P_X(i,j,k,l)$ only if the requirements of these matrices are met.

Note that in P_{fromL}, it is not possible to transition to P_L. This is because the recurrences are designed so that bands are handled in rounds. Within a round, bands in the left are handled first, if any, then those on the right, if any, and then those on the middle, with bands handled by P_M (if any) handled before those handled by P_O. A middle band *must* be handled in each round; otherwise, for example, two “bands” on the left group, added in different rounds, would collapse into one, causing the recurrences to incorrectly add penalty terms for band “borders” that are not actually borders. For this reason, no row in P_{fromL} has a P_L term, and so a band on the left group cannot be handled directly after a band on the right group. Also, P_{fromO} does not have a row with a P_M term, to ensure that P_M cannot be used twice on the same round.

Interior loops in the left band are decomposed by the rule 

; the remaining cases (right, middle, outer) are analogous. Note that, while the decomposition of $P_{L,iloop}(i,j)$ has two free indices, these indices are constraint by setting a constant maximum size of interior loops (here 30 bases) as it is common practice. For handling interior loops, the original CCJ algorithm introduced the five-ary function $P_{L,iloop5}$ and applied a clever scheme to still use only $\Theta(n^4)$ space. However, this space consumption could not easily be reduced further by sparsification. Thus, avoiding $P_{L,iloop5}$, which is possible due to the HotKnots energy model, is essential to reduce the space complexity.

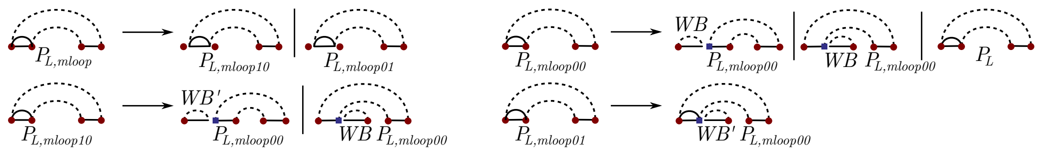


Figure 4 Decomposition of multi-loops in the left band.

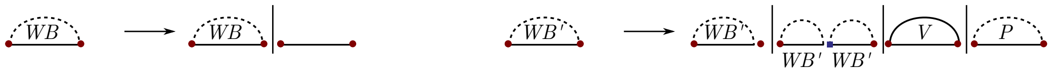


Figure 5 Decompositions of $WB(i, l)$ and $WB'(i, l)$.

Moreover, we modify the original handling of multi-loop cases to enable their sparsification. Fig. 4 shows our multi-loop handling for the left band. We handle multi-loops by passing through states $P_{L,mloop10}$, $P_{L,mloop01}$, and $P_{L,mloop00}$, which keep track of introduced inner multi-loop base pairs on the left or on the right. Finally, Fig. 5 shows the decompositions of $WB(i, l)$ and $WB'(i, l)$. The other "W"-matrices are decomposed analogously.

Further details of the original CCJ recurrences are available in the thesis [6], which also provides a detailed description of its sparsification.

3 Sparsification of the CCJ algorithm

By and large, "sparsification" allows keeping just the *required* dynamic programming matrix cells, which we refer to as *candidates*, (instead of the whole matrix) to find the MFE value. By storing a few candidates (as explained below) we avoid storing any four-dimensional CCJ matrix. (i) In recurrences in which the left-most index, i, does not change, we store the value of such recurrences in a constant number of three-dimensional matrix slices; we call the collection of these matrix slices *i-slices*. In many of these recursion cases, we recurse to matrix entries of the same i-slice (e.g. when inferring PK from P_L or, slightly more interestingly, P_R from P_{fromR}). In other cases, we recurse to the $(i + 1)$-slice (e.g. P_L from P_{fromL}) or $(i + c)$-slice, where c is constantly bounded. The latter occurs in the handling of interior loops ($P_{L,iloop}$ and analogously $P_{O,iloop}$), where c does not exceed the maximum interior loop size MLS. (ii) This leaves us with the recursion cases that recurse to some d-slice, where $d - i$ cannot be constantly bounded. Instead of storing all slices, we store only certain candidate entries in such slices. These matrix entries (called *candidates*) are stored in *candidate lists* for specific recursion cases together with their corresponding second, third, and fourth matrix indices, j, k, and l to keep track of band borders. In some cases, candidate lists can be shared between recursion cases. We presented more details on candidate list requirements in [11].

Space representation of the four-dimensional matrices by `sparseCCJ`

Only matrices corresponding to P_L and P_O occur in interior loops that span a band, and require to recurse to a different i-slice; for them, we store slices $i..i + MLS$. For matrices corresponding to P_{fromL}, P_{fromO}, $P_{L,mloop10}$, $P_{L,mloop01}$, $P_{O,mloop10}$, and $P_{O,mloop01}$, we only need to store slices i and $i + 1$. Matrices corresponding to recurrences of types $P_{X,iloop}$ and $P_{X,mloop}$ ($X \in \{L, R, M, O\}$) do not need to be stored and can be computed when needed. For the remaining matrices, we store only the current i-slice. Note that space is always reused in the next iteration; in case of ranges of slices, the memory access is 'rotated' without copying in memory. Matrices corresponding to the following recurrence cases require maintaining candidate lists: PK, P_L, P_O, P_{fromL}, P_{fromO}, $P_{L,mloop00}$, and $P_{O,mloop00}$.

To finalize sparsification, all recursion cases that recurse to these matrices – where the left-most index is not constantly bound – need to be modified. This affects all recursion cases, which insert any nested substructure to the left of the gapped region. This occurs exactly in the decompositions of PK, P_{fromL}, P_{fromO}, $P_{L,mloop10}$, $P_{L,mloop00}$, $P_{O,mloop10}$, and $P_{O,mloop00}$. Moreover, this affects the decomposition of P into two PK-fragments, where the latter is taken from the respective candidate list. We discuss the single modifications on the three examples of PK, $P_{L,mloop10}$, and P – the remaining cases are sufficiently similar to be sparsified analogously.

1. Consider the case of the P_{fromL} recurrence:

   $$\min_{i<d\leq j} WP(i,d-1) \;+\; PK(d,j,k,l).$$

   It suffices to minimize only over certain candidates $PK(d,j,k,l)$ that are *not optimally decomposable* in the following sense:

   $$\nexists e>d : PK(d,j,k,l) = WP(d,e-1) \;+\; PK(e,j,k,l). \tag{3}$$

   It can be shown easily that whenever $PK(i,j,k,l)$ is optimally decomposable, there is a candidate (i.e. a smaller, not optimally decomposable fragment) which yields the same minimum value. Remarkably, the candidate criterion can be efficiently checked by the dynamic programming algorithm. This is more directly seen from the equivalent candidate criterion

   $$PK(d,j,k,l) < min_{d<e\leq j} WP(d,e-1) \;+\; PK(e,j,k,l).$$

   Also this minimum can be computed by running only over candidates; moreover it must be calculated by the dynamic programming algorithm for computing $PK(d,j,k,l)$, such that the check is performed without additional overhead. This idea holds analogously for the other candidate checks.
2. Similarly, the minimization

   $$\min_{i<d\leq j} WB'(i,d-1) + P_{L,mloop00}(d,j,k,l)$$

   that occurs in the recurrence of $P_{L,mloop10}$ is restricted to candidates that satisfy

   $$\nexists e : P_{L,mloop00}(d,j,k,l) = WB(d,e-1) + P_{L,mloop00}(e,j,k,l). \tag{4}$$

   We can even strengthen the criterion, such that candidates must further satisfy

   $$\nexists e : P_{L,mloop00}(d,j,k,l) = P_{L,mloop00}(d,e,k,l) + WB(e+1,j). \tag{5}$$

3. In the minimization calculating $P(i,l)$, i.e.

   $$\min_{i<j<d<k<l} PK(i,j-1,d+1,k-1) + PK(j,d,k,l),$$

   it suffices to consider only candidates $PK(j,d,k,l)$. Entries $PK(j,d,k,l)$ are candidates if and only if they are *not optimally decomposable* in the following sense:

   $$\nexists e (j \leq e < d) : PK(j,d,k,l) = PK(j,e,k,l) + WP(e+1,d). \tag{6}$$

We emphasize that certain cases share the same candidates (allowing space savings). For example, the candidate criterion for decomposing $P_{L,mloop10}$ into WB' and $P_L, mloop00$ is identical to the one of decomposing $P_{L,mloop00}$ into WB' and $P_L, mloop00$. Similarly, we share the candidate lists of $P_{O,mloop10}$ and $P_{O,mloop00}$.

Finally, the sparsified CCJ recurrences can be computed based on the (constantly bounded) matrix slices and the candidates alone. Their correctness is a consequence of inverse triangle inequalities; for example in case of W we have $\forall x < y \leq z : W(x,z) \leq W(x,y-1) + W(y,z)$, which follows from the definition of W. Analogous inequalities hold for WP, WB, and WB.

▶ **Theorem 1** (Correctness of the CCJ sparsification). *The sparsified CCJ recurrences are equivalent to the non-sparsified CCJ recurrences.*

Proof. We show for fragments i,l and i,j,k,l by simultaneous induction on the fragment size (respectively, $l-i$ and $j-i+l-k$) that all changes in the definition of `sparseCCJ` (from the original to the sparsified recurrences of CCJ) are equivalent, in particular the values of the CCJ recursions and their corresponding sparsified versions are identical for each fragment.

It suffices to show the equivalence of the changes of the minimization cases explicitly. Moreover in each case, it suffices to show that there exists a minimum that is a candidate. By the induction hypothesis, the sparsified recursion for all smaller fragments do not have to be distinguished from the non-sparsified ones. We prove the three example cases PK, $P_{L,mloop10}$, and P explicitly; the remaining cases can be shown analogously.

1. Choose the largest d, $i < d \leq j$, s.t. $WP(i,d-1) + PK(d,j,k,l)$ is minimal. We show – by contradiction – that $PK(d,j,k,l)$ is a candidate, i.e. it satisfies Eq. (3). Assume Eq. (3) does not hold and choose e ($e > d$) such that $PK(d,j,k,l) = WP(d,e-1) + PK(e,j,k,l)$. Now, $WP(i,d-1) + PK(d,j,k,l) = WP(i,d-1) + WP(d,e-1) + PK(e,j,k,l) \geq WP(i,e-1) + PK(e,j,k,l)$ (using the inverse triangle inequality for WP). This contradicts the choice of d; thus $PK(d,j,k,l)$ must be a candidate.
2. Choose the largest d s.t. $WB'(i,d-1) + P_{L,mloop00}(d,j,k,l)$ is minimal. We show – by contradiction – that the candidate criterion, i.e. Eqs. (4) and (5), hold.
   - Assume Eq. (4) does not hold, then there exists some e s.t. $WB'(i,d-1)+P_{L,mloop00}(d,j,k,l) = WB'(i,d-1)+WB(d,e-1)+P_{L,mloop00}(e,j,k,l) \geq WB'(i,e-1) + P_{L,mloop00}(e,j,k,l)$; the latter inequality holds due to the definition of WB'.
   - Assume Eq. (5) does not hold, then there exists some e s.t. $P_{L,mloop00}(d,j,k,l) = P_{L,mloop00}(d,e,k,l) + WB(e+1,j)$. Consequently, the corresponding case of $P_{L,mloop10}(i,j,k,l)$ yields a smaller or equal value, since $WB'(i,d-1) + P_{L,mloop00}(d,j,k,l) = WB'(i,d-1) + P_{L,mloop00}(d,e,k,l) + WB(e+1,j) \geq P_{L,mloop10}(i,e,k,l) + WB(e+1,j)$.
3. For fixed $i < j < k < l$, choose the smallest d ($j < d < k$) s.t. $PK(i,j-1,d+1,k-1) + PK(j,d,k,l)$ is minimal. Assume Eq. (6) is violated, then there is some $e > d$, s.t. $PK(i,j-1,d+1,k-1) + PK(j,d,k,l) = PK(i,j-1,d+1,k-1) + PK(j,e,k,l) + WP(e+1,d) \geq_{(**)} PK(i,j-1,e+1,k-1) + PK(j,e,k,l)$, which contradicts the choice of d. For inequality (**), we observe that $PK(i,j-1,d+1,k-1) + WP(e+1,d) \leq PK(i,j-1,e+1,k-1)$ by definition of PK. ◀

Note that the presented sparsification would not have been possible for the multi-loop handling of the original CCJ algorithm (Fig. 6), which required us to modify these decompositions. In the original decomposition of $P_{L,mloop}$, the unbound access to d-slices

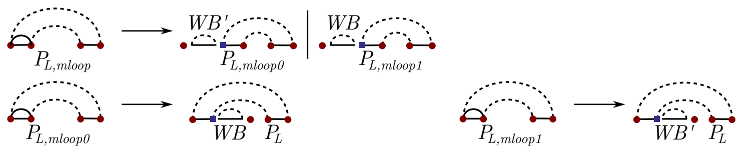


Figure 6 Decomposition of multi-loops in the left band by the original CCJ algorithm.

cannot easily be replaced by candidates – note the index offsets in the graphical notation, indicating that in the recurrence of $P_{L,mloop}(i,j,k,l)$, the WB' and WB fragments both start at $i+1$; similarly, there is a shift to $j-1$ in the recurrences of $P_{L,mloop0}(i,j,k,l)$ and $P_{L,mloop1}(i,j,k,l)$.

4 Space Complexity Analysis

In the previous sections, we sparsified all four-dimensional matrices of the CCJ algorithm with the goal of reducing its space complexity. As explained before, our sparsification allows us to replace all four-dimensional matrices by three-dimensional matrix slices. In seven recursion cases, we needed to rewrite minimizations, such that they compute equivalent results by recursing only to candidates (or entries of the same i-slice). In two of these cases, candidate lists can be shared.

Even if only a small fraction of the respective four-dimensional fragments are optimally decomposable (i.e. are not candidates), these changes will save space over the non-sparsified version. However, experience from previous sparsification (and our results) show that a large number of fragments is optimally decomposable, such that number of candidates is small.

We define Z as the total number of candidates. For traceback, we store a number of trace arrows, dynamically limiting their number; denote their maximum number by T. Then, the total space complexity of `sparseCCJ` is $O(n^3+Z+T)$.

5 Results

In this section we provide implementation and data set details, and show a comparison of `sparseCCJ` in terms of time and memory usage to its predecessors, original CCJ and modified CCJ.

5.1 Implementation

We implemented three versions of CCJ algorithm in C++; the first version is strictly based on the original CCJ recurrences, but the energy values are similar to that of DP09 energy model in HotKnots V2.0 [2]; we refer to this version as “original”. There are few energy functions in the original CCJ energy model that are not explicitly in the DP09 model. We have set values of these functions to 0, in order to make the models as similar as possible. The second version has modified recurrences to *match* the energy model of HotKnots V2.0, and is referred to as “modified”. The main difference between this version and the original version is in calculating energy of interior loops that span the band. The energy of an interior loop or multi-loop depends on whether or not the external base pair of the loop is pseudoknotted. If it is not, we call the loop *ordinary*, and otherwise say that the loop *spans a band*. If the external base pair of an ordinary interior loop, not including stacks, is $i.l$ and the other closing base pair is $d.e$, then similar to the DP09 energy model, the energy of the interior

Table 1 Summary of data sets used in this work.

Name	# of sequences	Structure Type	sequence lengths	Reference
HK-PK	88	pseudoknotted	26–400	test set of [2]
HK-PK-free	337	pseudoknot-free	10–194	test set of [2]
IP-pk168	168	pseudoknotted	21–137	test set of [16]
DK-pk16	16	pseudoknotted	34–377	test set of [17]

loop is calculated by a call to the function $e_{int}(i, d, e, l)$. If the internal loop spans a band we call the function $e_{intP}(i, d, e, l)$ instead. The third version uses the sparsification technique, and is referred to as "sparse" (`sparseCCJ`). Similar to [21], we utilize trace arrows to keep track of accessible cells and employ a garbage collection technique to remove trace arrows from unreachable cells. Using these techniques in `sparseCCJ` required extensions like trace arrows between different matrices, which were only briefly mentioned in [21]; otherwise it demonstrates the generality of this approach.

5.2 Data sets

We use a large data set of over 600 RNA strands of length between 10–400 bases to analyze the performance of our algorithm. This data set was compiled from three non-overlapping data sets with various pseudoknotted and pseudoknot-free structures. Table 1 provides a summary of these data sets.

5.3 Benchmark Results

We ran all three versions of CCJ implementations on Amazon Cloud (r4.8xlarge instance consisting of 32 Xeon E5-2686 Broadwell 2.3 GHz CPUs, and 244GB of DDR3 RAM) and compared their time and memory requirements for instances of our data set. First, we verified that the two versions that implement DP09 energy model of HotKnots V2.0, i.e. the modified CCJ and the sparse CCJ implementation, indeed produced exactly the same results. This equivalence must hold in correct implementations due to Theorem 1. We focus on (run time and space) *performance* of sparse CCJ, in this work. Fig. 7 shows – in log scale – the memory consumption (left), our main objective in this work, and run times (right), both as functions of sequence length. We observe significant improvements in space from the original implementation over the modified one to the sparse implementation. At the same time, one observes a comparably small run time penalty in our (little run time-optimized) sparse implementation. However, given today's heavily parallel computation platforms (with comparably costly main memory), differences in run-time are generally less relevant than space improvements.

We further investigated whether a specific class of structures (i.e. pseudoknotted vs. pseudoknot-free) would benefit stronger than the other from sparsification, and found out that both classes benefit equally from sparsification (see Fig. 8); in general longer sequences benefit more (as seen in Fig. 7). Closer look at Fig. 7 shows that while `sparseCCJ` has variation in memory usage within the same length, these variations are minimal. We looked at numbers of candidates and trace arrows (9) in sparse CCJ, which together explain the space requirements.

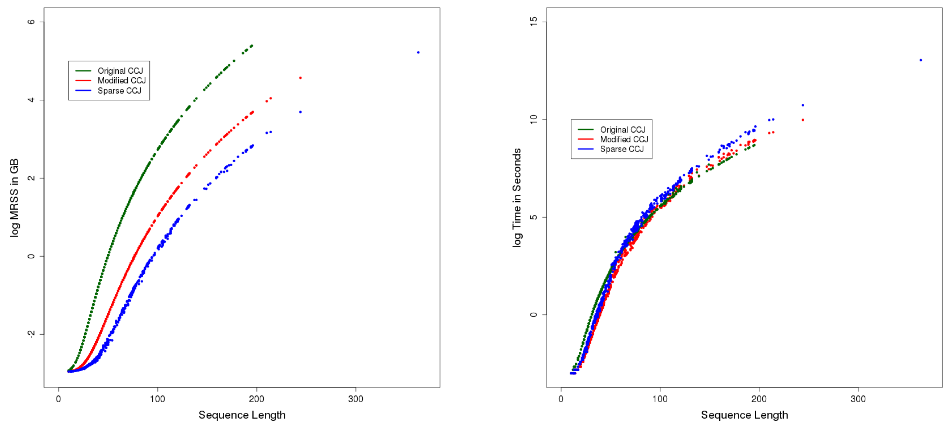


Figure 7 Memory usage (left, presented as log of maximum resident set size in GB) and run time (right, presented as log of time in second) vs. length for the three CCJ implementations.

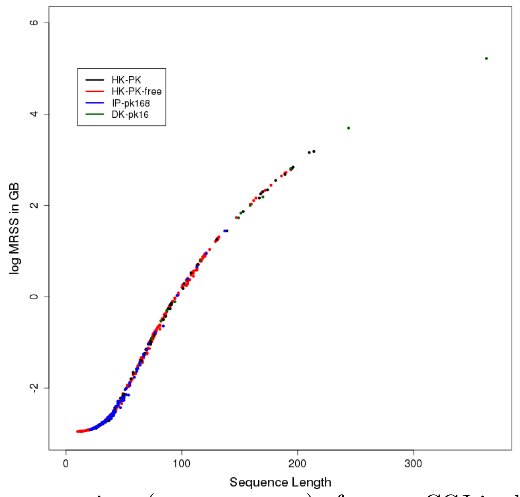


Figure 8 Performance comparison (memory usage) of sparse CCJ in different class of structures.

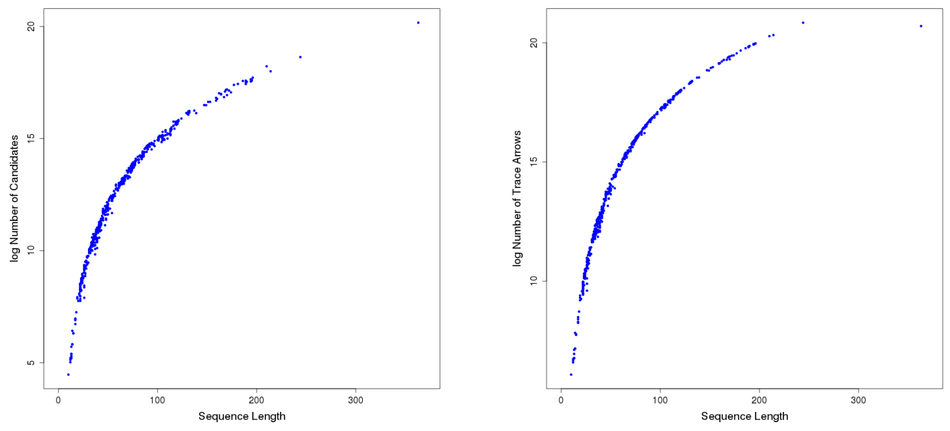


Figure 9 Total number of candidates (measured as log(total number of candidates)) and trace arrows (measured as log(maximum number of trace arrows)) used in SparseCCJ versus sequence length.

6 Conclusion

We have presented the first application of the sparsification to a complex pseudoknot structure prediction algorithm – supporting kissing hairpins with arbitrarily nested substructures – with a realistic energy model. While previous applications of the sparsification technique mainly focused on speed improvements, we solely aimed at reducing the space requirements, which is the main factor limiting the practical applicability of complex RNA pseudoknotted secondary structure prediction. Our comparison to two previous CCJ variants provides interesting insights into general potentials for space improvements of complex RNA-related algorithms. Finally, our space savings in `sparseCCJ` open the door for large scale biologically-relevant application of pseudoknot structure prediction covering all important pseudoknot classes.

Acknowledgements. We thank Anne Condon for discussing details of the CCJ algorithm and first ideas on space savings.

References

1 T. Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Disc. App. Math.*, 104(1–3):45–62, 2000.

2 M. S. Andronescu, C. Pop, and A. E. Condon. Improved free energy parameters for RNA pseudoknotted secondary structure prediction. *RNA (New York, N.Y.)*, 16(1):26–42, January 2010.

3 R. Backofen, D. Tsur, S. Zakov, and M. Ziv-Ukelson. Sparse RNA folding: Time and space efficient algorithms. *Journal of Discrete Algorithms*, 9(1):12–31, March 2011. `doi:10.1016/j.jda.2010.09.001`.

4 K.-Y. Chang and I. Tinoco. The structure of an RNA kissing hairpin complex of the HIV TAR hairpin loop and its complement. *Journal of Molecular Biology*, 269(1):52–66, May 1997. `doi:10.1006/jmbi.1997.1021`.

5 H. L. Chen, A. Condon, and H. Jabbari. An o(n(5)) algorithm for MFE prediction of kissing hairpins and 4-chains in nucleic acids. *Journal of computational biology : a journal of computational molecular cell biology*, 16(6):803–815, June 2009. `doi:10.1089/cmb.2008.0219`.

6 H. Jabbari. *Algorithms for prediction of RNA pseudoknotted secondary structures.* PhD thesis, University of British Columbia, March 2015.

7 R. B. Lyngsø. Complexity of pseudoknot prediction in simple models. In *ICALP'04*, pages 919–931, 2004.

8 R. B. Lyngsø and C. N. Pedersen. RNA pseudoknot prediction in energy-based models. *J. Comput. Biol.*, 7(3-4):409–427, 2000.

9 W. J. Melchers, J. G. Hoenderop, H. J. Bruins Slot, C. W. Pleij, E. V. Pilipenko, V. İ. Agol, and J. M. Galama. Kissing of the two predominant hairpin loops in the coxsackie B virus 3' untranslated region is the essential structural feature of the origin of replication required for negative-strand RNA synthesis. *Journal of Virology*, 71(1):686–696, January 1997.

10 T. R. Mercer, M. E. Dinger, and J. S. Mattick. Long non-coding RNAs: insights into functions. *Nature Reviews Genetics*, 10(3):155–159, March 2009. `doi:10.1038/nrg2521`.

11 M. Möhl, R. Salari, S. Will, R. Backofen, and S. C. Sahinalp. Sparsification of RNA structure prediction including pseudoknots. *Algorithms for Molecular Biology*, 5(1):39+, December 2010. `doi:10.1186/1748-7188-5-39`.

12 R. Nussinov and A. B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, November 1980. `doi:10.1073/pnas.77.11.6309`.

13 J. Reeder and R. Giegerich. Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, 5, 2004.

14 E. Rivas and S. R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.*, 285(5):2053–2068, 1999.

15 R. Salari, M. Möhl, S. Will, S. C. Sahinalp, and R. Backofen. Time and Space Efficient RNA-RNA Interaction Prediction via Sparse Folding. In Bonnie Berger, editor, *Research in Computational Molecular Biology*, volume 6044 of *Lecture Notes in Computer Science*, chapter 31, pages 473–490. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-12683-3_31.

16 K. Sato, Y. Kato, M. Hamada, T. Akutsu, and K. Asai. IPknot: fast and accurate prediction of RNA secondary structures with pseudoknots using integer programming. *Bioinformatics*, 27(13):i85–i93, July 2011.

17 J. Sperschneider, A. Datta, and M. J. Wise. Predicting pseudoknotted structures across two RNA sequences. *Bioinformatics (Oxford, England)*, 28(23):3058–3065, December 2012.

18 Y. Uemura, A. Hasegawa, S. Kobayashi, and T. Yokomori. Tree adjoining grammars for RNA structure prediction. *Theor. Comput. Sci.*, 210(2):277–303, 1999.

19 M. H. Verheije, R. C. L. Olsthoorn, M. V. Kroese, P. J. M. Rottier, and J. J. M. Meulenberg. Kissing interaction between 3' noncoding and coding sequences is essential for porcine arterivirus RNA replication. *Journal of Virology*, 76(3):1521–1526, February 2002. doi:10.1128/jvi.76.3.1521-1526.2002.

20 Y. Wexler, C. Zilberstein, and M. Ziv-Ukelson. A study of accessible motifs and RNA folding complexity. *Journal of computational biology: a journal of computational molecular cell biology*, 14(6):856–872, 2007. doi:10.1089/cmb.2007.r020.

21 S. Will and H. Jabbari. Sparse RNA folding revisited: space-efficient minimum free energy structure prediction. *Algorithms for molecular biology: AMB*, 11, 2016.

Vaquita: Fast and Accurate Identification of Structural Variation Using Combined Evidence

Jongkyu Kim[1] and Knut Reinert[2]

1 **Department of Mathematics and Computer Science, Freie Universität Berlin, Berlin, Germany; and**
Max Planck Institute for Molecular Genetics, Berlin, Germany
`j.kim@fu-berlin.de`

2 **Department of Mathematics and Computer Science, Freie Universität Berlin, Berlin, Germany; and**
Max Planck Institute for Molecular Genetics, Berlin, Germany
`knut.reinert@fu-berlin.de`

Abstract

Motivation: Comprehensive identification of structural variations (SVs) is a crucial task for studying genetic diversity and diseases. However, it remains challenging. There is only a marginal consensus between different methods, and our understanding of SVs is substantially limited. In general, integration of multiple pieces of evidence including split-read, read-pair, soft-clip, and read-depth yields the best result regarding accuracy. However, doing this step by step is usually cumbersome and computationally expensive.

Result: We present Vaquita, an accurate and fast tool for the identification of structural variations, which leverages all four types of evidence in a single program. After merging SVs from split-reads and discordant read-pairs, Vaquita realigns the soft-clipped reads to the selected regions using a fast bit-vector algorithm. Furthermore, it also considers the discrepancy of depth distribution around breakpoints using Kullback-Leibler divergence. Finally, Vaquita provides an additional metric for candidate selection based on voting, and also provides robust prioritization based on rank aggregation. We show that Vaquita is robust in terms of sequencing coverage, insertion size of the library, and read length, and is comparable or even better for the identification of deletions, inversions, duplications, and translocations than state-of-the-art tools, using both simulated and real datasets. In addition, Vaquita is more than eight times faster than any other tools in comparison.

Availability: Vaquita is implemented in C++ using the SeqAn library. The source code is distributed under the BSD license and can be downloaded at `http://github.com/seqan/vaquita`.

1998 ACM Subject Classification J.3 Life and Medical Sciences

Keywords and phrases Structural variation

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.13

1 Introduction

Next generation sequencing (NGS) provides us remarkable opportunity to find genetic variants that are directly linked to diseases such as cancer [13] and rare genetic disorders [2]. Therefore, there has been a growing attention in identifying such variants. The size of genetic variations ranges from a single base pair to megabases [15]. Among them, structural variations (SVs), i.e. variations that are usually larger than 50 nucleotides in size, play a major role in many phenotypic differences. In contrast to single-nucleotide polymorphisms

(SNPs) or small indels, SVs are much more diverse in type and size and are often harder to find confidently [1]. Consequently, it is not surprising that there is only a marginal consensus between different variant callers [1]. It is fair to say that the current understanding of SVs is substantially limited and large-scale studies often rely on multiple variant callers that use different methods to obtain the most comprehensive list of SVs. However, the integration of multiple outputs is often cumbersome due to the required prior knowledge of different algorithms and their parameters and also suffers from limited computational resources. Therefore, there is an urgent need for a better method that can detect SVs more accurately and efficiently.

The algorithms for SV identification can be categorized into four types [1]. First, we can use split-read evidence. The reads spanning a breakpoint have to be split to be able to map multiple loci. For example, Pindel [24] splits discordant reads and tries to find breakpoints by mapping them to different positions. However, it is difficult to find SVs in some part of a genome such as repeat-rich regions using just the spilt-read information.

Additionally, read-pair information can be used to identify SVs. With the prior knowledge of the proper orientations and distribution of insertion sizes in paired-end sequencing libraries, read-pairs with improper orientation and/or insertion size can be identified and used to detect SVs. However, read-pair information alone does not provide base-pair resolution accuracy. Accordingly, a variant caller like Delly [19] considers read-pair information together with split-read information.

Many recently-developed short-read mappers [12, 10] provide local alignments. These mappers produce soft-clipped reads, meaning that only a part of a sequence is mappable to the reference genome. The unmapped sequences are relatively short and erroneous, which make them difficult to map to a unique position. To resolve this issue, CREST [23] assembles contigs around potential breakpoints and map them to a reference genome using Blat [9].

Lastly, read-depth information is also useful in finding copy number variations. However, the depth of coverage of sequencing data is usually non-uniform [14]. Thus, a significance testing such as event-wise testing [25] is required to distinguish the true signals from background noise. Moreover, read-depth information alone cannot provide base-pair resolution accuracy.

Often, integrating results from multiple approaches yield better performance regarding accuracy. In this aspect, LUMPY [11] uses a probabilistic framework to combine split-read and read-pair information by default, and MetaSV [16] focuses on connecting multiple external tools.

Our method, Vaquita, integrates split-read, read-pair, soft-clipped, and read-depth information in a single program to achieve maximum accuracy while also maintaining speed. Vaquita utilize all four types of information without contributions from external tools. The overall workflow of Vaquita is depicted in Figure 1.

2 Methods

2.1 Breakpoint and structural variation identification

The overall workflow of Vaquita is depicted in Figure 1(a). We define a breakpoint using the coordinate information of two genomic segments (intervals) and their orientation with respect to each other. We call the two intervals *left* and *right* intervals according to their genomic coordinates. We also define three types of orientation as shown in Figure 1(b), namely, normal, inverted and swapped. Reads and read-pairs with inverted and swapped orientations are considered to be discordant, and suggesting a breakpoint. These discordant

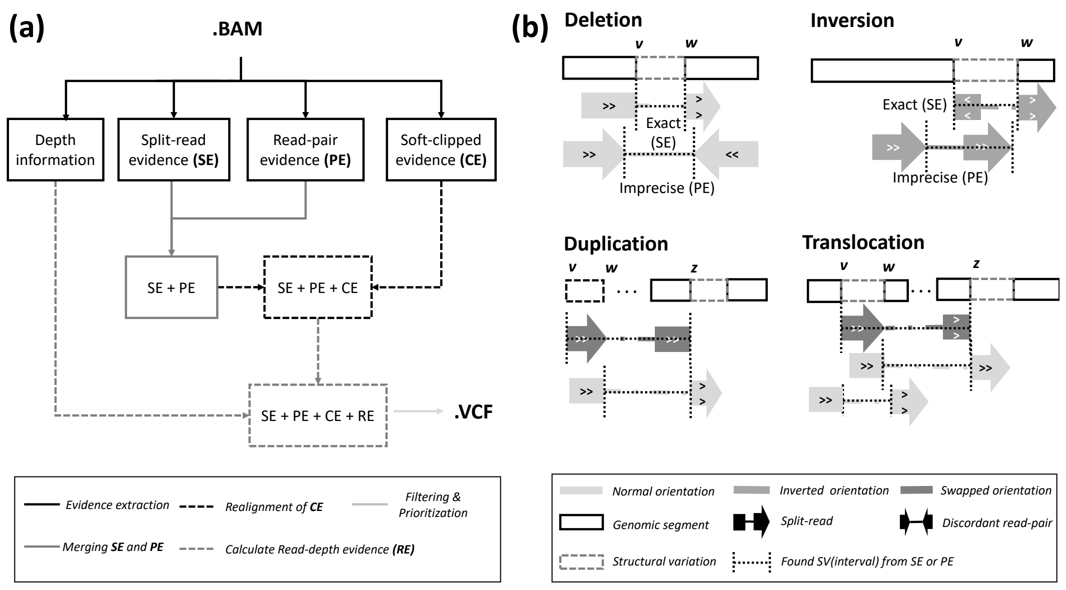


Figure 1 (a) The overall process of Vaquita. (b) The four types of structural variations. The dotted rectangles colored with gray denote the modification of the reference genome. The deletion and inversion in the figures show structural variations from v to w. The duplication and translocation in the figures illustrate copied or moved segments from v to w, and z is the target position.

reads also constitute of split-read evidence (SE) and read-pair evidence (PE), which are the number of reads and read-pairs that support a breakpoint, respectively. For read-pairs with normal orientation, we estimate m and s which denote the median and median absolute deviation of insertion size distribution in a sequencing dataset. Then we apply a cutoff that is $m + s \times 9$ by default. The adjustment of this cutoff affects the accuracy of the result. A weak cutoff yields a sensitive result, but at the cost of specificity. The value 9 is empirically decided after testing values from 5 to 10 (data not shown). Note that this default value is the same as Delly2 and was used in large scale studies such as 1000 Genome Project [21]. We define four SV types, namely *deletion, inversion, duplication*, and *translocation* that are illustrated in Figure 1(b). We identify deletions and inversions from breakpoints with normal and inverted orientation, respectively. The two types of breakpoints are required to find duplications and translocations. The definition is based on previous studies [22, 19]. Nevertheless, there are alternative definitions of SVs which are not always mutually exclusive. Other types of SVs can be identified by the user from the reported breakpoints.

2.2 Candidate merging: SE + PE

Two breakpoints with the same orientation can be merged if both the left and right intervals are adjacent or overlapping. A distance of 50 bases is set by default in assessing adjacency. When two breakpoints are merged, the minimum and maximum positions of each left and right intervals are selected to define the merged breakpoint. The original positions are kept in a list, and the median positions are reported as final positions in the last step. We merge all the breakpoints identified by SE or PE according to this principle. For efficiency, the reference genome is divided into equally sized regions that are 1000 bp by default. The *left* and *right* intervals of SVs belong to one or more regions according to their size and genomic coordinates. The entire merging process can be efficiently done by identifying breakpoints in

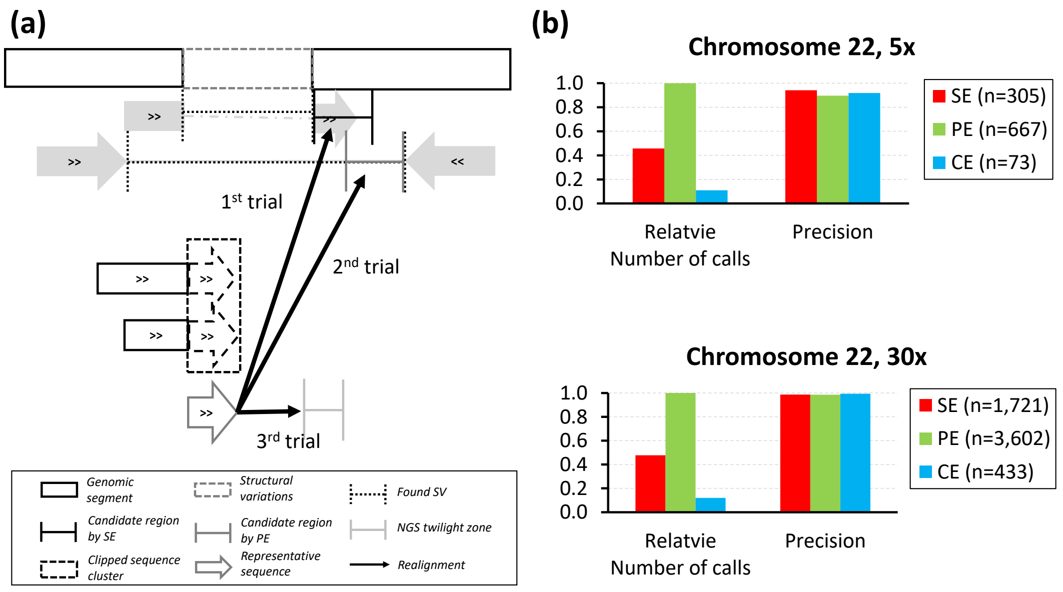


Figure 2 (a) The realignment process for soft-clipped evidence (CE). (b) The relative number of structural variation calls and precision of each evidence types in simulation datasets.

the same region. In the worst case, all the SVs are distinct and fall into the same region. In this scenario, the comparison step takes $O(n^2)$ where n is the number of SVs. However, in practice, the distribution of SVs are sparse, and only a small number of SVs are expected to co-exist in a single region. If all the SVs exist in unique regions, the time complexity is $O(n)$ since only one hashing is required. This process is conducted in parallel while decompressing a `.bam` file, which is usually an I/O bound process.

2.3 Realignment of soft-clipped reads: SE + PE + CE

Mapping the clipped part of sequences is challenging because they are short and erroneous. One can assemble longer contigs to map them correctly and uniquely. However, the entire process is computationally expensive. Instead, Vaquita selects a representative sequence without assembly and reduces the search space by surveying only the pre-selected regions. Initially, Vaquita identifies clusters of soft-clipped sequences according to the genomic coordinates of their mapped parts. Subsequently, it locates the longest unmapped sequence in a cluster and uses it as a representative sequence. Then, it tries to map those representative sequences to candidate regions identified by SE or PE using a fast bit-vector algorithm for approximate string matching [17], using lenient criteria. The time complexity of the algorithm is $O(nr/k)$ where n and r are the sizes of the read and the reference, and k is the word size of the machine which is 64 in modern hardwares including ours. Often, relatively small deletions are difficult to find using read-pair information because the sizes of SVs fall within the variance of the insertion size. This region has been defined as the NGS twilight zone [22]. To address this, Vaquita also examines the genomic sequences around the clipped position for queries that failed the mapping. The size of the additional searching region is set to $m + s \times 9$ by default, where m and s are median and median absolute deviation of the insertion size distribution. The default value is based on criterion for identifying discordant read-pairs in Section 2.1. Only soft-clipped parts that are equal to or longer than

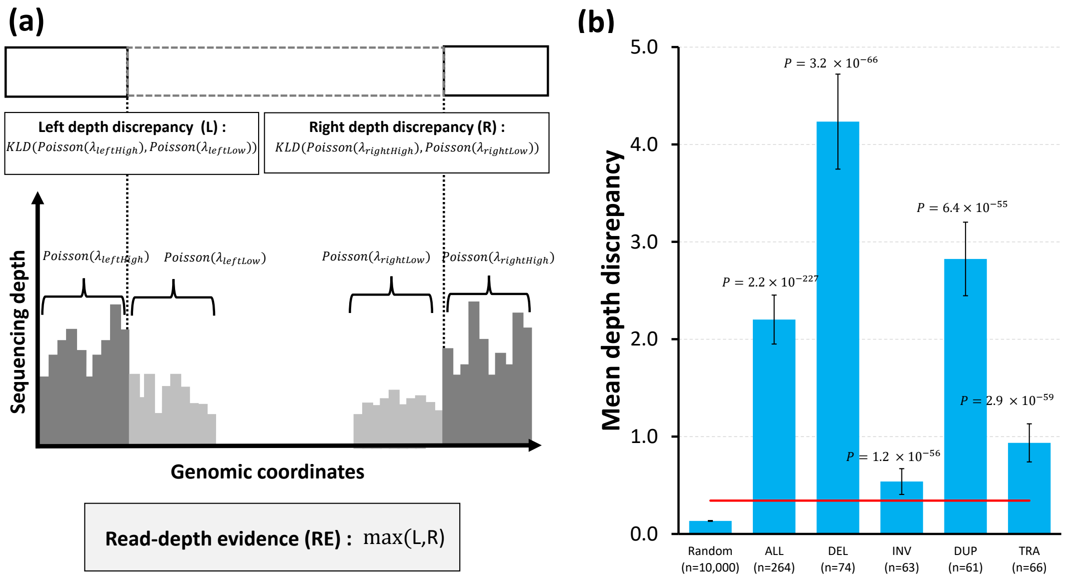


Figure 3 (a)The read-depth evidence. (b) The depth discrepancy distribution of random positions and four types of structural variations in the simulation dataset (Chromosome 22 and 30x coverage). P indicates p-values obtained by two-tailed *Kolmogorov-Smirnov test* using a random sample. The red line shows the third quartile plus the interquartile range of the random sample.

20 nucleotides undergo realignment, allowing edit distance of 10% of the sequence size by default. The overall process is described in Figure 3(a). In the two simulation datasets, the quantity of CE was more than 20% of SE, and 10% of PE. Furthermore, the precision of CE is about the same as SE and PE, as shown in Figure 2(b). Note that Mason [6] we used in the simulation selects random positions to introduce variations. This limitation usually makes simulation tests less challenging since SVs are not randomly distributed. For example, many of the SVs are found in repeat-rich regions in real datasets. The higher precisions reported in Figure 2(b) reflect this limitation.

2.4 Calculation of depth discrepancy

The depth distribution of a sequencing sample has been previously reported to be non-uniform. Hence, the distributions around two randomly picked positions that are not adjacent to each other are likely to be different. We use this observation to discriminate true breakpoints from false positives, especially for unbalanced structural variations. For two genomic intervals i_1 and i_2, we calculate λ_1 and λ_2 that are the mean depth of each interval. We then assume that the local distributions follow a *Poisson* distribution and calculate the *Kullback-Leibler divergence (KLD)* from *Poisson*(λ_1) to *Poisson*(λ_2) as follows:

$$KLD_{\lambda_1,\lambda_2} = \lambda_1 - \lambda_2 + \lambda_1 \log \frac{\lambda_2}{\lambda_1}\,. \tag{1}$$

We use this as a metric of depth discrepancy between i_1 and i_2 and calculate the read-depth evidence (RE) for each breakpoint as described in Figure 3(a). As a rule, we always calculate the divergence from the higher depth region to the lower depth region and select the larger value between the discrepancies of the left and right side of the breakpoint. We use the window size of 20 bases for calculating local coverages. To efficiently calculate RE for all

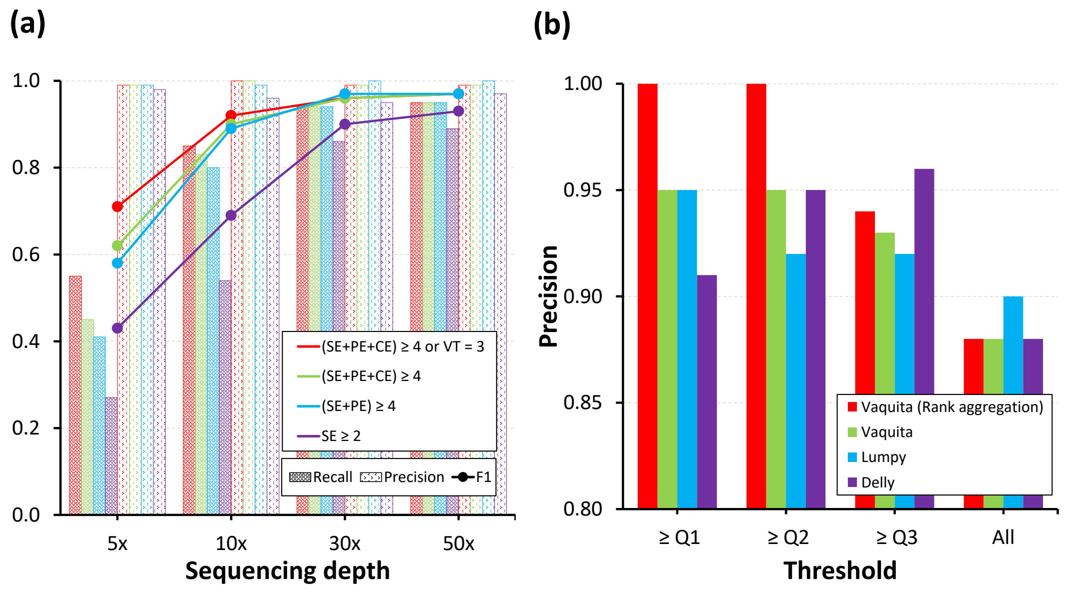


Figure 4 (a) The impact of evidence integration on breakpoint identification. The criterion VT=3 rescues SVs that are supported by all three evidence types as described in Section 2.5.2. (b) Prioritization performance of deletion calling in the 30x simulation dataset. Q1-Q3 indicates the first to the third quartile.

breakpoints, we maintain a lookup table regarding various λ_1 and λ_2. In the simulation datasets, the depth discrepancy distributions were significantly different from random samples as shown in Figure 3(b). As expected, RE is more effective in discriminating unbalanced structural variations like deletions and duplications than balanced structural variations such as inversions. By default, we do not use RE for inversions. However, one can turn on this option.

2.5 Combined evidence

2.5.1 SE + PE + CE

We assessed the impact of evidence integration in breakpoint identification starting from the SE only case. The result is shown in Figure 4(a). We combined the number of split-reads and read-pairs that supported a breakpoint or an SV and used it as a cutoff. We applied a cutoff of 4 as we explained the reason in Section 3.2. However, we had to apply a cutoff of 2 for SE only case since it was too stringent when using a single evidence type in low-coverage samples. Our experiment showed that more accurate results were achieved when additional types of information were considered. The impact is more dramatic in datasets with low-coverage. In the 5x dataset, we obtained the F1 score of 0.62 using the evidence from SE+PE+CE and only 0.43 using SE only. The effect is less pronounced in high-coverage datasets. For the 50x dataset we obtained 0.97 and 0.93, respectively.

2.5.2 Voting based metric for candidate selection

Often, variant callers such as Delly2 and LumpyExpress apply basic filtration using a sum of split-reads and read-pairs that support SVs. Instead of using a simple sum of signals from different types of evidence, Vaquita provides an additional metric for candidate selection based on voting. In this scheme, each type of evidence for a breakpoint is checked by a relatively lenient cutoff, and then we calculate the number of evidence types that pass the criteria that we denote as VT. For example, a structural variation with $VT = 3$ is supported by three evidence types. By default, we used ≥ 1 for SE and PE since this is the most lenient condition. For RE, we used $\geq (Q3 + IQR \times 1.0)$ where $Q3$ and IQR denote the third quartile and the interquartile range of depth discrepancy score from random positions. The red line in Figure 3(b) shows this default RE cutoff in a simulation dataset. Note that we add CE to SE to treat them as a single evidence type. In Figure 4(a), we applied $VT = 3$ as an additional criterion to rescue SV candidates in low-coverage samples and obtained better recalls without reducing precision. Therefore, we used this option by default for later analyses. One can also use this metric to filter out false positives at repeat-rich regions, instead of excluding those regions from the analysis.

2.5.3 Prioritization by rank aggregation

In practice, prioritization of SVs is an important task for downstream analysis. We formulate this problem to find an aggregated rank from the ranks based on multiple evidence types. At first, we define the goodness of a rank based on *Spearman's footrule distance* [3]. It is given as follows:

$$F(\phi) = \sum_{e}^{\Phi} \sum_{s}^{S} |\phi_e(s) - \phi(s)| \tag{2}$$

where $\phi_x(i)$ is the rank of the element i by the evidence type x, $\Phi = \{SE, PE, RE\}$, and S is the set of all structural variation candidates. In this scheme, the optimal rank ϕ^* is the one that minimize F. We also define the median rank ϕ^M that is defined as follows:

$$\phi^M(s) = median(\phi_{SE}(s), \phi_{PE}(s), \phi_{RE}(s)) \,. \tag{3}$$

The cost of calculating ϕ^M is $O(|\Phi| \cdot |S| \log |S|)$ using a quick sort. Hence, it is applicable to large datasets. Furthermore, $\phi^M = \phi^*$ if there is no tie [4] and, in the presence of ties, ϕ^M is still a 3-approximate solution of ϕ^* [5]. Hence, we calculated ϕ^M instead of ϕ^* for rank aggregation. To prevent arbitrary breaking up of ties, we obtain the ϕ using two different criteria. At first, we order the candidates according to the strength of the evidence, for example, the number of split-reads for SE. Secondly, we use the depth around the breakpoints as a tie breaker. In this scheme, the candidate that is in the lower covered region receives the higher rank. The impact of rank aggregation is shown in Figure 4(b). All the parameters including the cutoff value was same as described in Section 3.2. In the figure, the top 50% ($\geq Q2$) of the structural variations detected by Vaquita turned out to be true positives after prioritization.

3 Result

3.1 Preliminaries

3.1.1 Datasets and variant callers

We generated a diploid that contains SVs based on chr22 of hg19/GRCh37 using Mason [6]. We set the size range from 30 to 5000 and used simulated rates of 4.0×10^{-6} for indel and 2.0×10^{-6} for inversion, duplication, and translocation, respectively. We also introduced

SNPs and small indels with simulated rates of 2.0×10^{-4} and 4.0×10^{-5} to mimic the natural variation, but these were not the focus of the evaluation. From the SVs introduced in chr22, we generated a simulation dataset using ART [7]. We selected Mason since it can simulate the types of SVs that are defined previously, and ART because it provides simulation profiles for the platforms such as Illumina HiSeq-2500. We simulated Illumina paired-end sequencing data with the HS25 option. The depth, read-length, the mean and standard deviation of insertion sizes are shown in Figure 5. We used the 50x sequencing samples of a trio from the Illumina Platinum Genome, NA12878, NA12891 and NA12892, the accession numbers being ERR194147, ERR194160, and ERR194161, respectively. We also used the validation set from the Genome In A Bottle (GIAB) consortium that was constructed using multiple sequencing platforms, including a long-read technology [18]. We compared the performance of Vaquita with five variant callers relying on different sets of evidence types. Delly2 [19], LumpyExpress [11] and Pindel [24] use split-read and read-pair information. We also considered CREST [23] that uses read-depth and soft-clipped reads information, and GASVPro [20] that uses paired-end and read-depth information. All the reads were aligned to hg19/GRCh37 using BWA-MEM [12] with default parameters. We also used BLAT for CREST and SAMBLASTER for LumpyExpress.

3.1.2 Validation process

We only considered breakpoints and SVs that are ≥ 50 nucleotides in size. The identified breakpoints were considered as true positive if we could find a match in the validation set that had more than 80% of reciprocal overlap. For variant callers that report intervals rather than exact positions like GASVPro, we considered the identified variations as valid if there was a match in the validation set that had both ends within the identified intervals. We used in-house scripts to interpret each of `.vcf` files from different variant callers according to our definition of SVs in Figure 1(b). We also used default parameters for each variant callers and noted for when otherwise.

3.2 Performance comparison using simulation data

The comparison with other variant callers using the simulation datasets is shown in Figure 5. We combined the number of split-reads with read-pairs and applied 4 as the minimum cutoff for all variant callers. We used this single cutoff throughout all the comparison in the manuscript to see the performance in overall and noted for when otherwise. Note that there can be best parameters for each variant callers for each test condition, and sometimes 4 is not always the default value. For example, Lumpy uses 4 while Delly2 uses 3 by default. We also applied a mapping quality cutoff of 20 for when a variant caller supported such functionality. Vaquita included voting based candidates with the default parameters explained in the method section. All the variant callers yielded better accuracy as the depth increases. Notably, Vaquita, LumpyExpress, and Delly2 constantly ranked as the top three. Vaquita clearly outperformed the top three in the 5x and 10x datasets, mainly because of voting based rescue. However, the difference in performance decreased for the 30x and 50x datasets. Although we could not observe significant differences in high-coverage samples, the result of Figure 4(b) suggests that the prioritization performance of Vaquita is better than the others. Note that Delly2 showed the highest precision when using Q3 as the threshold in Figure 4(b), However, the difference between Q1 and Q3 cases was only 0.05. This result can be explained by the limitation of simulation based testing that we mentioned in Section 2.3. However, this pattern is not observed when using real datasets as shown in Figure 6(a). We could

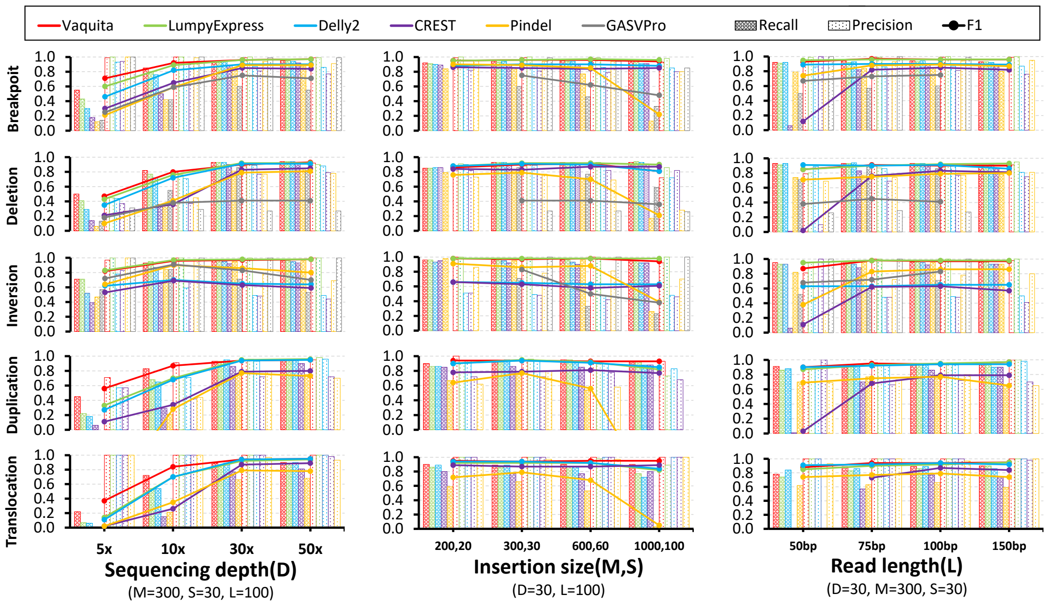


Figure 5 The assessment of accuracy using simulated datasets. M and S indicate the mean and standard deviation of the insertion size.

not observe notable differences from Vaquita, LumpyExpress, Delly2 and CREST from the insert size variations. However, Pindel and GASVPro's accuracy were dramatically reduced in larger insert size samples. Vaquita, LumpyExpress, and Delly2 showed robust accuracy across the read length. However, CREST and Pindel that undergo split-read alignments internally showed poor performance in the samples with read lengths of 50bp. GASVPro was unable to find duplications and translocations (defined according to our criteria), reported alternative SV types, and was unable to run on smaller insert size (200bp) or longer read length (150bp) samples.

From such results, we selected Vaquita, LumpyExpress, and Delly2 as the top three variant callers and their performance was further analyzed using real datasets.

3.3 Performance comparison using real datasets

3.3.1 Overlap between variant callers

The matched fraction of deletions calls compared to GIAB were 0.55, 0.57, and 0.48 for Vaquita, LumpyExpress, and Delly2, respectively. We also investigated the prioritization performance by selecting top 25%, 50%, and 75% of SVs using the rank aggregation for Vaquita, and the evidence summation for LumpyExpress and Delly2 in Figure 6(a). The difference between the overall matching fraction and that of top 25% ($\geq Q1$) were 0.30, 0.26 and 0.19 for Vaquita, LumpyExpress, and Delly2, respectively. This result suggests that the rank aggregation based on multiple evidence types is still effective in this relatively high-coverage samples. In Figure 6(b), Vaquita and Delly2 contain about 28% and 25% of unique breakpoints, while LumpyExpress only has 10%. However, the total number of breakpoints calls were 6,681 and 6,658 for Vaquita and Delly2, while only 5,420 for LumpyExpress. This result suggests that the cutoff used in the comparison are more stringent for LumpyExpress. Notably, LumpyExpress only identified 154 inversions while Vaquita and Delly2 identified 816 and 602 inversions, respectively. One possible explanation is that Vaquita and Delly2

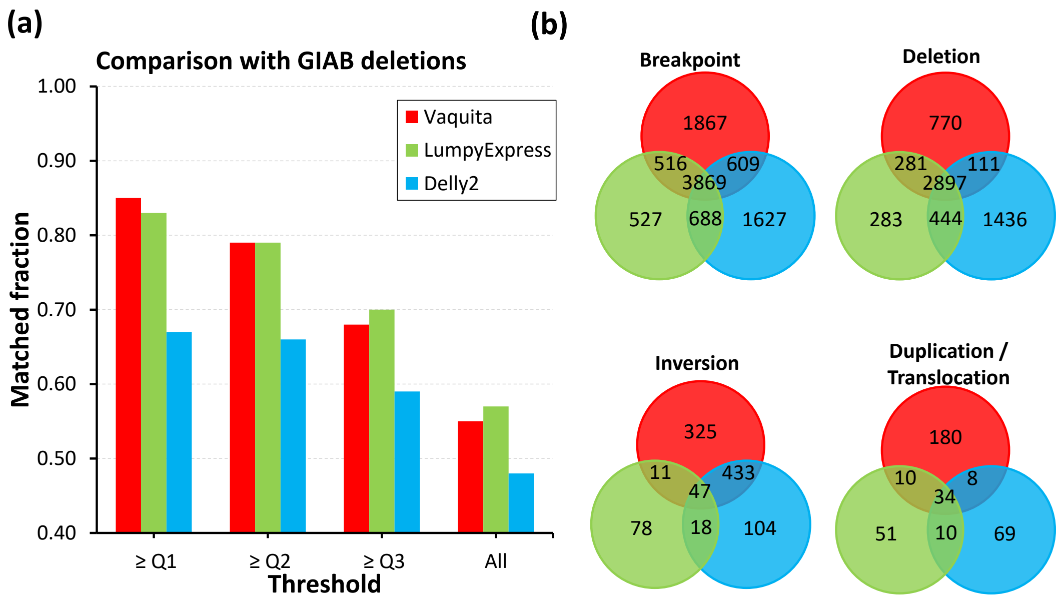


Figure 6 The comparion between variant callers using the NA12878 dataset. (a) Comparison with deletion calls from Genome in a Bottle consortium. (b) The result overlaps between tools.

have an internal realignment process while LumpyExpress is not. However, this explanation is not rigorously tested.

3.3.2 Trio analysis

In Figure 7, the Mendelian errors of breakpoints were 0.23, 0.11, and 0.18 for Vaquita, LumpyExpress, and Delly2, respectively. The error count of LumpyExpress being the lowest can be explained by the high fraction of structural variations that were overlapping in both parents. For LumpyExpress, these fractions were substantially higher than the others, especially in deletions. We additionally investigated the rate of overlaps between two parents and found that LumpyExpress had 0.42 of overlaps in breakpoints while Vaquita and Delly had 0.37 and 0.38, respectively. Note that a recent study using hydatidiform moles and long-reads sequencing technology suggested that 32% of overlaps in deletions and insertions between two genetically unrelated individuals [8]. Therefore, 42% of overlaps between two parents were higher than expected although the reason is not clear. The main difference of the Mendelian errors between Vaquita and Delly2 were due to inverisons. For inverisons, the Mendelian errors were 0.30 for Vaquita, and 0.13 for Delly2. This much of difference is not consistent with the previous analysis using simulation datasets. Therefore, we suspect that several types of inversion couldn't be simulated by Mason properly. However, Vaquita obtained the Mendelian errors of 0.19 and 0.46 for deletions and duplications while Delly2 obtained 0.20 and 0.48, respectively.

3.4 Runtime performance

We compared the CPU time reported by the `time` command in Debian Linux. In the comparison, Vaquita was significantly faster than the other tools for analyzing the human WGS sample with 50x coverage (NA12878). We found that Vaquita is 8.2 times faster than LumpyExpress and 9.6 times faster than Delly2 which only finds one variant type in

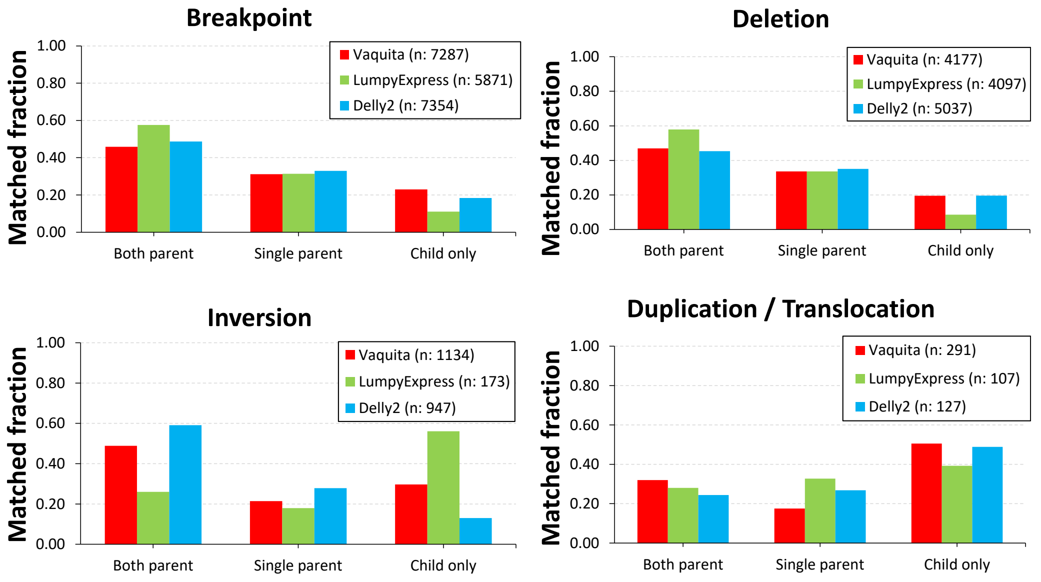


Figure 7 The SVs identified from the child dataset(NA12878) were compared to the SVs from the parent's datasets (NA12891 and NA12892).

a single run. Vaquita only required less than 40 minutes in our test environment. We can explain this speed-up based on three observations. First, the merging step of Vaquita that we explained in Section 2.2 is very fast in practice since structural variations are sparsely distributed across the human genome. Second, the realignment step is also very fast and took less than 5 minutes in total for the test case. Third, the `.bam` file processing of SeqAn library is also faster than other implementation since it internally separates several threads for decompression of `bgzf` stream. Regarding the last reason, we modified the original source code of the library so that the library was fixed to a single thread for the decompression process. Although it is still a separated thread, we used CPU time for the comparison. Note that LumpyExpress calls an external tool for `.bam` file parsing, and Delly2 should be ran multiple times to find all variant types. Therefore, we concluded that the comparison is not specifically biased to Vaquita. The peak memory consumption were 12.5G for Vaquita, 6.4G for LumpyExpress, and 320M for Delly2. This relatively large memory consumption was due to inefficient implementation for storing positions and can be improved in the future version.

All the tests were done on a Debian GNU/Linux 8 machine with two Intel Xeon E5-2667V2 Octa core CPUs at 3.3 GHz, 387 GB of RAM, and 2 TB of SATA SSDs on RAID5 configuration. We did not attempt to use multi-threading for each variant caller.

4 Discussion and conclusion

Vaquita was developed to integrate split-read, read-pair, soft-clipped, and read-depth information and provides effective evidence combination strategy based on voting and rank aggregation. In the benchmark using the simulation datasets, Vaquita showed relatively robust performance across different sequencing depths, insert sizes and read lengths. In the comparison with GIAB deletions, about 55% of deletions found by Vaquita were matched and showed better prioritization results compared to LumpyExpress and Delly2. Vaquita also identified more breakpoints than the others and about 28 percent of them were unique. In the trio analysis, Vaquita showed similar number of Mendelian errors compared to Delly2 and higher number of errors compared to LumpyExpress. The difference between these errors can be explained by the prevalence of overlapping variations in both parents (LumpyExpress), or by errors in inversions (Delly2). The runtime of Vaquita was significantly faster than those of LumpyExpress and Delly2 by a factor of more than 8 times. As a future goal, we will provide an improved functionality to confidently integrate other orthogonal datasets, including long-read datasets.

Acknowledgements. The authors thank Dr. Birte Kehr for her suggestions regarding validation process and many valuable comments on the manuscript. The authors also thank Dr. Bernhard Renard and Kathrin Trappe for their helpful discussion. J.K was supported by the International Max Planck Research School for Computational Biology and Scientific Computing and the Efficient Algorithms for Omics Data group at the MPI for Molecular Genetics.

References

1 Can Alkan, Bradley P. Coe, and Evan E. Eichler. Genome structural variation discovery and genotyping. *Nature reviews. Genetics*, 12(5):363–376, 2011. `arXiv:NIHMS150003`, `doi:10.1038/nrg2958`.

2 Kym M. Boycott, Megan R. Vanstone, Dennis E. Bulman, and Alex E. MacKenzie. Rare-disease genetics in the era of next-generation sequencing: discovery to translation. *Nature reviews. Genetics*, 14(10):681–91, 2013. `doi:10.1038/nrg3555`.

3 Persi Diaconis. Group representations in probability and statistics. *Lecture Notes-Monograph Series*, 11:i–192, 1988.

4 Cynthia Dwork, Ravi Kumar, Moni Naor, and D Sivakumar. Rank aggregation methods for the Web. *Proceedings of the 10th international conference on World Wide Web*, pages 613–622, 2001. `doi:10.1145/371920.372165`.

5 Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 47–58, 2004. `doi:10.1145/1055558.1055568`.

6 Manuel Holtgrewe. Mason – A Read Simulator for Second Generation Sequencing Data. Technical report, Freie Universität Berlin, 2010.

7 Weichun Huang, Leping Li, Jason R. Myers, and Gabor T. Marth. ART: A next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012. `doi:10.1093/bioinformatics/btr708`.

8 John Huddleston, Mark Jp Chaisson, Karyn Meltz Steinberg, Wes Warren, Kendra Hoekzema, David S Gordon, Tina A Graves-Lindsay, Katherine M Munson, Zev N Kronenberg, Laura Vives, Paul Peluso, Matthew Boitano, Chen-Shin Chin, Jonas Korlach, Richard K Wilson, and Evan E Eichler. Discovery and genotyping of structural variation from long-read haploid genome sequence data. *Genome research*, page gr.214007.116, 2016. URL: `http://www.ncbi.nlm.nih.gov/pubmed/27895111`, `doi:10.1101/gr.214007.116`.

9 W James Kent. BLAT – The BLAST-Like Alignment Tool. *Genome Research*, 12:656–664, 2002. `doi:10.1101/gr.229202.`

10 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat Methods*, 9(4):357–359, 2012. `arXiv:{\#}14603`, `doi:10.1038/nmeth.1923`.

11 Ryan M. Layer, Colby Chiang, Aaron R. Quinlan, and Ira M. Hall. LUMPY: a probabilistic framework for structural variant discovery. *Genome biology*, 15(6):R84, 2014. `arXiv:1210.2342`, `doi:10.1186/gb-2014-15-6-r84`.

12 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv*, 00(00):3, 2013. URL: `http://arxiv.org/abs/1303.3997`.

13 Matthew Meyerson, Stacey Gabriel, and Gad Getz. Advances in understanding cancer genomes through second-generation sequencing. *Nature reviews. Genetics*, 11(10):685–96, 2010. `doi:10.1038/nrg2841`.

14 Alison M Meynert, Morad Ansari, David R FitzPatrick, and Martin S Taylor. Variant detection sensitivity and biases in whole genome and exome sequencing. *BMC bioinformatics*, 15:247, 2014. `doi:10.1186/1471-2105-15-247`.

15 Ryan E. Mills, Klaudia Walter, Chip Stewart, Robert E. Handsaker, Ken Chen, Can Alkan, Alexej Abyzov, Seungtai Chris Yoon, Kai Ye, R. Keira Cheetham, Asif Chinwalla, Donald F. Conrad, Yutao Fu, Fabian Grubert, Iman Hajirasouliha, Fereydoun Hormozdiari, Lilia M. Iakoucheva, Zamin Iqbal, Shuli Kang, Jeffrey M. Kidd, Miriam K. Konkel, Joshua Korn, Ekta Khurana, Deniz Kural, Hugo Y. K. Lam, Jing Leng, Ruiqiang Li, Yingrui Li, Chang-Yun Lin, Ruibang Luo, Xinmeng Jasmine Mu, James Nemesh, Heather E. Peckham, Tobias Rausch, Aylwyn Scally, Xinghua Shi, Michael P. Stromberg, Adrian M. Stütz, Alexander Eckehart Urban, Jerilyn A. Walker, Jiantao Wu, Yujun Zhang, Zhengdong D. Zhang, Mark A. Batzer, Li Ding, Gabor T. Marth, Gil McVean, Jonathan Sebat, Michael Snyder, Jun Wang, Kenny Ye, Evan E. Eichler, Mark B. Gerstein, Matthew E. Hurles, Charles Lee, Steven A. McCarroll, and Jan O. Korbel. Mapping copy number variation by population-scale genome sequencing. *Nature*, 470(7332):59–65, feb 2011. `doi:10.1038/nature09708`.

16 Marghoob Mohiyuddin, John C. Mu, Jian Li, Narges Bani Asadi, Mark B. Gerstein, Alexej Abyzov, Wing H. Wong, and Hugo Y K Lam. MetaSV: An accurate and integrative structural-variant caller for next generation sequencing. *Bioinformatics*, 31(16):2741–2744, 2015. `doi:10.1093/bioinformatics/btv204`.

17 Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415, 1999. `doi:10.1145/316542.316550`.

18 Hemang Parikh, Marghoob Mohiyuddin, Hugo Y K Lam, Hariharan Iyer, Desu Chen, Mark Pratt, Gabor Bartha, Noah Spies, Wolfgang Losert, Justin M Zook, and Marc Salit. Svclassify: a Method To Establish Benchmark Structural Variant Calls. *BMC genomics*, 17(1):64, 2016. `doi:10.1186/s12864-016-2366-2`.

19 T. Rausch, T. Zichner, A. Schlattl, A. M. Stutz, V. Benes, and J. O. Korbel. DELLY: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics*, 28(18):i333–i339, 2012. `doi:10.1093/bioinformatics/bts378`.

20 Suzanne S. Sindi, Selim Onal, Luke Peng, Hsin-Ta Wu, and Benjamin J. Raphael. An integrative probabilistic model for identification of structural variation in sequencing data. *Genome biology*, 13(3):R22, 2012. URL: `http://www.ncbi.nlm.nih.gov/pubmed/22452995`, `doi:10.1186/gb-2012-13-3-r22`.

21 Peter H. Sudmant, Tobias Rausch, Eugene J. Gardner, Robert E. Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, Miriam K. Konkel, Ankit Malhotra, Adrian M. Stütz, Xinghua Shi, Francesco Paolo Casale, Jieming Chen, Fereydoun Hormozdiari, Gargi Dayama, Ken Chen, Maika Malig, Mark J. P. Chaisson, Klaudia Walter, Sascha Meiers, Seva Kashin, Erik Garrison, Adam Auton, Hugo Y. K. Lam, Xinmeng Jasmine Mu, Can Alkan, Danny Antaki, Taejeong Bae, Eliza Cerveira, Peter Chines, Zechen Chong, Laura Clarke, Elif Dal, Li Ding, Sarah Emery, Xian Fan, Madhusudan Gujral, Fatma Kahveci, Jeffrey M. Kidd, Yu Kong, Eric-Wubbo Lameijer, Shane McCarthy, Paul Flicek, Richard A. Gibbs, Gabor Marth, Christopher E. Mason, Androniki Menelaou, Donna M. Muzny, Bradley J. Nelson, Amina Noor, Nicholas F. Parrish, Matthew Pendleton, Andrew Quitadamo, Benjamin Raeder, Eric E. Schadt, Mallory Romanovitch, Andreas Schlattl, Robert Sebra, Andrey A. Shabalin, Andreas Untergasser, Jerilyn A. Walker, Min Wang, Fuli Yu, Chengsheng Zhang, Jing Zhang, Xiangqun Zheng-

Bradley, Wanding Zhou, Thomas Zichner, Jonathan Sebat, Mark A. Batzer, Steven A. McCarroll, Ryan E. Mills, Mark B. Gerstein, Ali Bashir, Oliver Stegle, Scott E. Devine, Charles Lee, Evan E. Eichler, and Jan O. Korbel. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015. doi:10.1038/nature15394.

22 Kathrin Trappe, Anne-Katrin Katrin Emde, Hans-Christian Christian Ehrlich, and Knut Reinert. Gustaf: Detecting and correctly classifying SVs in the NGS twilight zone. *Bioinformatics (Oxford, England)*, 30(24):1–8, 2014. doi:10.1093/bioinformatics/btu431.

23 Jianmin Wang, Charles G Mullighan, John Easton, Stefan Roberts, Sue L Heatley, Jing Ma, Michael C Rusch, Ken Chen, Christopher C Harris, Li Ding, Linda Holmfeldt, Debbie Payne-Turner, Xian Fan, Lei Wei, David Zhao, John C Obenauer, Clayton Naeve, Elaine R Mardis, Richard K Wilson, James R Downing, and Jinghui Zhang. CREST maps somatic structural variation in cancer genomes with base-pair resolution. *Nature methods*, 8(8):652–4, 2011. arXiv:NIHMS150003, doi:10.1038/nmeth.1628.

24 Kai Ye, Marcel H. Schulz, Quan Long, Rolf Apweiler, and Zemin Ning. Pindel: A pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, 25(21):2865–2871, 2009. arXiv:NIHMS150003, doi:10.1093/bioinformatics/btp394.

25 Seungtai Yoon, Zhenyu Xuan, Vladimir Makarov, Kenny Ye, and Jonathan Sebat. Sensitive and accurate detection of copy number variants using read depth of coverage. *Genome Research*, 19(9):1586–1592, 2009. doi:10.1101/gr.092981.109.

Assessing the Significance of Peptide Spectrum Match Scores*

Anastasiia Abramova[1] and Anton Korobeynikov[2]

1 **Department of Statistical Modeling, Saint Petersburg State University, Saint Petersburg, Russia; and Center for Algorithmic Biotechnology, Saint Petersburg State University, Saint Petersburg, Russia**

2 **Department of Statistical Modeling, Saint Petersburg State University, Saint Petersburg, Russia; and Center for Algorithmic Biotechnology, Saint Petersburg State University, Saint Petersburg, Russia**

Abstract

Peptidic Natural Products (PNPs) are highly sought after bioactive compounds that include many antibiotic, antiviral and antitumor agents, immunosuppressors and toxins. Even though recent advancements in mass-spectrometry have led to the development of accurate sequencing methods for nonlinear (cyclic and branch-cyclic) peptides, requiring only picograms of input material, the identification of PNPs via a database search of mass spectra remains problematic. This holds particularly true when trying to evaluate the statistical significance of Peptide Spectrum Matches (PSM) especially when working with non-linear peptides that often contain non-standard amino acids, modifications and have an overall complex structure.

In this paper we describe a new way of estimating the statistical significance of a PSM, defined by any peptide (including linear and non-linear), by using state-of-the-art Markov Chain Monte Carlo methods. In addition to the estimate itself our method also provides an uncertainty estimate in the form of confidence bounds, as well as an automatic simulation stopping rule that ensures that the sample size is sufficient to achieve the desired level of result accuracy.

1998 ACM Subject Classification G.3 [Probability and Statistics] Statistical Software, J.3 [Computer Applications] Biology and Genetics

Keywords and phrases mass spectrometry, natural products, peptide spectrum matches, statistical significance

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.14

1 Introduction

Tandem mass-spectrometry (MS/MS) is an attractive alternative to nuclear magnetic resonance (NMR) spectroscopy that can be used to sequence non-linear (cyclic and branch-cyclic) peptides. Usually MS/MS is coupled with a database search algorithm capable of locating candidate peptides within the database of protein sequences, computing the peptide-spectrum match scores and estimating the statistical significance of the PSMs found.

A number of recent studies have been focusing on trying to compute the statistical significance of the PSMs. Since this particular problem is very similar to the thoroughly researched issue of having to compute the statistical significance of sequence match scores, many different approaches were proposed. For example, in [2] it was proposed to approximate

* This work was supported by the Saint Petersburg State University (grant number 15.61.951.2015).

the statistical significance of PSMs by first modeling the distribution of the PSM scores (e.g., by Gumbel distribution) and further using this distribution to calculate the probability of interest. Unfortunately, while useful in many other applications, this approximation approach, often fails when one has to estimate extremely small PSM probabilities typical for mass spectrometry (e.g., values as small as 10^{-10} are often required to achieve 1% FDR [12]).

Linear peptides and additive scoring functions use a polynomial-time algorithm [11] to compute the PSM p-values. It would seem, however, that the same approach cannot be applied to non-linear peptides. A groundbreaking breakthrough [15] gave rise to MS-DPR, an algorithm capable of computing the p-values of the PSM using the Markov Chain Direct Probability Redistribution approach. Unfortunately, while the algorithm has great appeal and appears to be quite universal, MS-DPR does not give any indication as to the accuracy of the calculated estimates and its overall performance greatly depends on the size of the sample, i.e. the number of simulations used to compute the p-value. The algorithm, however, does not provide any guidelines as to how one should go about selecting the correct size for the initial sample so as to assure quality end results.

Fortunately, the *rare probability estimation* problem itself is not new and has been very well studied within the framework of such fields as particle physics, stochastic simulation, financial mathematics, chemistry and telecommunication theory among the others.

We are using several state-of-art methods of the Monte Carlo sampling theory, including the Markov Chain Monte Carlo, importance sampling, the Wang-Landau algorithm and the efficient variance estimates for Markov Chains to derive a novel method, capable not only of estimating the statistical significance of the PSMs, but also of constructing confidence bounds for the p-value of interest and provide a way to predict the size of the sample that would be required to achieve the desired level of result accuracy.

2 Methods

2.1 Probabilistic model of a spectrum of an arbitrary peptide

We use the same probabilistic model to compute the statistical significance of PSM as presented in [15, 14]. For the sake of completeness we will describe it below.

A *PNP graph* G of a peptide P is defined as a graph with nodes $V(G)$ corresponding to amino acids in P and edges $E(G)$ corresponding to *generalized* peptide bonds [14][1]. The mass $Mass(G)$ of a PNP graph is defined as the total mass of its amino acids, i.e. $Mass(G) = \sum_{v \in V(G)} m(v)$.

A peptide bond is called a *bridge* if its removal disconnects the graph. A pair of bonds is called a *2-cut* if neither of them are bridges but removing both of them simultaneously disconnects the graph. Let $\mathcal{C}_b$ be the set of bridges of G and $\mathcal{C}_2$ be the set of pairs of 2-cut edges and we define the *set of cuts* of G as $\mathcal{C}(G) = \mathcal{C}_b(G) \cup \mathcal{C}_2(G)$.

Any cut $C' \in \mathcal{C}$ induces two masses (theoretical peaks) $m_b(C')$ and $m_y(C')$ of the connected components of G resulting from the cut C'. Note that these two peaks are *complementary* with a total mass equal to the molecular mass of the compound, $Mass(G)$. This means that for the PNP graph G and its set of cuts $\mathcal{C}$ there exist two vectors of masses $\vec{m}_b = (m_b^{(1)}, \dots, m_b^{(|\mathcal{C}|)})$ and $\vec{m}_y = (m_y^{(1)}, \dots, m_y^{(|\mathcal{C}|)})$. The vector $\vec{m}_b$ is called the *theoretical spectrum* of P and further be referred as $TheoreticalSpectrum(P)$.

[1] Generalized peptide bonds include N-C-O linkage amide bonds as well as C-C-O linkage bonds between thiazoles/oxazoles and dehydroalanines/dehydrobutyrines and other amino-acids. The notion of generalized peptide bonds is useful as illustrated by identification of the thiazole/oxazole containing PNP plantazolicin from *B. amyloliquefaciens*, lanthipeptide SapB from *S. coelicolor*, and complex PNPs such as two-rings containing actinomycin from *Streptomyces sp. CNS654* [14].

The $\mathit{TheoreticalSpectrum}(P)$ can also be represented via a *fragmentation matrix* $H = \{h_{ij}\}$ of size $|\mathcal{C}| \times |V(G)|$ with the elements $h_{ij} = 1$ if $j \in V(G_1(C^{(i)}))$ and 0 otherwise. Here $C^{(i)} \in \mathcal{C}$ and $G \setminus C^{(i)} = G_1(C^{(i)}) \cup G_2(C^{(i)})$. Rows of the fragmentation matrix correspond to different, potentially observable fragmentations. Each row specifies which amino acids are to be found on one of the connected component of graph G after the removal of some nodes. This means that $\mathit{TheoreticalSpectrum}(P) = H\vec{\mu}$, where $\vec{\mu}$ is a vector of the masses of the amino acids.

$\mathit{SPCScore}(P, \mathit{Spectrum})$ is defined as the *Shared Peak Count*, the number of peaks shared between $\mathit{TheoreticalSpectrum}(P)$ and the filtered MS/MS spectrum $\mathit{Spectrum}$ [5]. Two peaks are considered as shared if their masses are within a pre-defined threshold (typically 0.02 Da for high-resolution spectra). From here on we will consider $\mathit{Spectrum}$ to be fixed and we will denote $\mathit{Score}(\vec{\mu}) = \mathit{SPCScore}(\mathit{Spectrum}, H\vec{\mu})$.

We will use $\mathcal{M}$ to denote a set of vectors that satisfy the following condition:

$$\mathcal{M} = \{\vec{\mu} = (\mu_1, \ldots, \mu_{|V(G)|}), |\ \mu_i > 0, \sum_{i=1}^{|V(G)|} \mu_i = \mathit{Mass}(G)\}. \tag{1}$$

This set represents a variety of amino acid mass-vectors (with possible non-standard amino-acids that are typical for non-ribosomal peptides, modifications, etc. mixed in). Our goal is to calculate the probability

$$p = \mathbb{P}(\mathit{SPCScore}(\mathit{Spectrum}, H\vec{\mu}) \geq S^*) = \mathbb{P}(\mathit{Score}(\vec{\mu}) \geq S^*), \tag{2}$$

where $\vec{\mu}$ is a random variable uniformly distributed on set $\mathcal{M}$ and S^* is a fixed threshold (usually $S^* = \mathit{SPCScore}(\mathit{Spectrum}, P)$). Note that the probability (2) defined above depends on the particular choice of the set $\mathcal{M}$. We could obtain different models of PSM significance via changing the scoring function $\mathit{SPCScore}$ and/or the set $\mathcal{M}$. For example, if we consider an integer simplex $\mathcal{M}'$ (so all the μ_i would be integers), additive scoring functions and linear peptides, then we will end with the PSM statistical significance model as used by MS-GF+ [11]. The estimates presented below could easily be adopted to a different model.

2.2 Monte Carlo and the Importance Sampling Approach

The probability (2) could be estimated by using the Monte Carlo sampling approach. Consider the set

$$\mathcal{S} = \{\vec{\mu} \in \mathcal{M} : \mathit{Score}(\vec{\mu}) \geq S^*\}.$$

Denote by $\mathbb{1}_{\mathcal{S}}$ an indicator function of the set $\mathcal{S}$, i.e. $\mathbb{1}_{\mathcal{S}}(\vec{\mu})$ equals 1 if $\vec{\mu} \in \mathcal{S}$ (equivalently, $\mathit{Score}(\vec{\mu}) \geq S^*$) and 0 otherwise. Let $\vec{\mu}_1, \ldots, \vec{\mu}_N$ be N iid random variables with a uniform distribution on the set $\mathcal{M}$. Then

$$\widehat{p}_{MC} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\mathcal{S}}(\vec{\mu}_i)$$

is an unbiased and consistent estimate of p.

Variance $D(\widehat{p}_{MC})$ of $\widehat{p}_{MC}$ equals to $\frac{p(1-p)}{N}$ and tends to 0 as $N \to \infty$. However, its relative error

$$RE(\widehat{p}_{MC}) = \frac{D(\widehat{p}_{MC})}{p^2} = \frac{p(1-p)}{Np^2} = \frac{1}{Np} - \frac{1}{N} \longrightarrow \infty,\ p \to 0, \tag{3}$$

is unbounded indicating that the performance deteriorates when the event is rare. For example, if a relative error at 1% is desired and the probability is of order 10^{-6} then we need to take N such that $\sqrt{(10^6-1)/N} \leq 0.01$. This implies that $N \approx 10^{10}$ which is an unfeasible task for most computer systems. Therefore we need to look for an estimate that would have its variance smaller than $D(\widehat{p}_{MC})$, ideally having the relative error bounded (or at least growing much slower).

Importance sampling is a general Monte-Carlo approach to reduce the variance in the estimation of quantities that can be written as an expectations. Importance sampling generates the "interesting" events more often by sampling from a different distribution and correcting for this bias afterward, which results in a more accurate estimate with a reasonable number of samples.

Formally, let $\mathcal{P}$ be the distribution of $\vec{\mu}_i$ and f the corresponding density. Consider another distribution $\mathcal{Q}$ with density q. Let $\vec{\nu}_1, \ldots, \vec{\nu}_N$ be a sequence of iid random variables having the distribution $\mathcal{Q}$. Then *importance sampling estimator* of p is defined as

$$\widehat{p}_{IS} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\vec{\nu}_i)}{q(\vec{\nu}_i)} \mathbb{1}_{\mathcal{S}}(\vec{\nu}_i). \tag{4}$$

Note that this estimator is also consistent and unbiased.

Suppose that the density q has the following form:

$$q(x) = cw(x)f(x), \tag{5}$$

where $c > 0$ is a normalizing constant and $w(x)$ is some biasing factor. Then $\widehat{p}_{IS}$ would not depend on f and could be written as

$$\widehat{p}_{IS} = \frac{\sum_{i=1}^{N} \mathbb{1}_{\mathcal{S}}(\vec{\nu}_i)/w(\vec{\nu}_i)}{\sum_{i=1}^{N} 1/w(\vec{\nu}_i)}, \tag{6}$$

where $\vec{\nu}_1, \ldots, \vec{\nu}_N$ is a sequence of iid random variables with density function q.

2.3 Metropolis-Hastings Algorithm with Wang-Landau weighting

In order to calculate $\widehat{p}_{IS}$ we need to sample from the distribution $\mathcal{Q}$ with density (5). This might be a non-trivial task, since the normalizing constant c of $q(x)$ is unknown and weights $w(x)$ could be arbitrary.

Usually this task is approached by using the Metropolis-Hastings [7] algorithm that allows for the usage of unnormalized densities. Our goal is to construct a Markov Chain having $\mathcal{Q}$ as an equilibrium distribution and obtain the desired sequence $\{\vec{\nu}_n\}$ of random variables via sampling from this distribution. Then the ergodicity of the Markov chain will ensure that as $N \to \infty$ still $\widehat{p}_{IS}$ will converge to the target probability p almost surely [18, 17].

Algorithm 1: Metropolis-Hastings algorithm

```
Input  : Transition kernel $\gamma(x|y)$, current state of Markov Chain $\vec{\nu}_i$
Output : Next state of Markov-Chain $\vec{\nu}_{i+1}$
1 Sample random variable $\vec{\nu}$ from conditional probability distribution $\gamma(\cdot|\vec{\nu}_i)$
2 Sample uniform random variable $r$ on the interval $[0;1]$
3 Calculate the acceptance ratio $\alpha = \min\left(\frac{q(\vec{\nu}_i)\gamma(\vec{\nu}|\vec{\nu}_i)}{q(\vec{\nu})\gamma(\vec{\nu}_i|\vec{\nu})}, 1\right)$
4 if $r < \alpha$ then
5     $\vec{\nu}_{i+1} \leftarrow \vec{\nu}$
6 else
7     $\vec{\nu}_{i+1} \leftarrow \vec{\nu}_i$
8 end
```

The density q here is defined by (5) and sampling from proposal density $\gamma(\cdot|\vec{\nu}_i)$ is performed as follows:

```
Algorithm 2: Simulation from conditional density $\gamma(\cdot|\vec{\nu}_i)$
   Input  : Current state of Markov Chain $\vec{\nu} = (\nu_1, \dots, \nu_{|V(G)|}) \in \mathcal{M}$
   Output : Proposed state $\tilde{\nu}$
 1 Sample index $i$ uniformly on $\{1, \dots, |E(G)|\}$
 2 Consider $e_i \in E(G)$. Denote by $v_1$ a starting vertex of $e_i$ and by $v_2$ an ending vertex
 3 Sample $\delta$ uniformly on $[-m(v_1); m(v_2)]$
 4 Set $\tilde{\nu}_{v_1} \leftarrow \nu_{v_1} + \delta$, $\tilde{\nu}_{v_2} \leftarrow \nu_{v_2} - \delta$, and $\tilde{\nu}_j \leftarrow \nu_j$ for the rest $j$
```

The density f in our case is a uniform density on the set $\mathcal{M}$ and it is natural to assume that the weights $w(\vec{\nu})$ should be score-invariant. Therefore to calculate $\widehat{p}_{IS}$ we will consider a proposal distribution $\mathcal{Q}$ with the density $q(\vec{\nu}) = cw(Score(\vec{\nu}))f(\vec{\nu})$. In order to decrease the relative error $RE(\widehat{p}_{IS})$ we aim to choose $w(S) \approx 1/\mathbb{P}(Score(\vec{\nu}) = S)$. This way sampling from $\mathcal{Q}$ will yield a flat score distribution reducing the variance of $\widehat{p}_{IS}$.

For this purpose we adapt a variant of Wang-Landau algorithm [13, 19]. This algorithm is an adaptive modification of the Metropolis-Hastings algorithm, which can simultaneously construct the Markov Chain and estimates weights. We use this algorithm, however, only to estimate weights because the resulting random walk is not even Markovian and therefore one could not guarantee the consistency of the estimates and lack of bias.

```
Algorithm 3: Wang-Landau algorithm
    Input  : Minimum and maximum values of log-weight increments $C_{min}$, $C_{max}$
    Output : Set of weights $w(S)$
 1  Set $w[i] \leftarrow 0$ for $i \in S_{min}, \dots, S_{max}$, where $S_{min}$ and $S_{max}$ are minimum and
      maximum scores correspondingly
 2  $C \leftarrow C_{max}$
 3  while $C > C_{min}$ do
 4      Set $Hist[i] \leftarrow 0$ for $i \in S_{min}, \dots, S_{max}$
 5      Simulate $\vec{\nu}$ uniformly on $\mathcal{M}$
 6      while Hist is not sufficiently flat do
 7          Run a step of Metropolis-Hastings algorithm with density
              $q(\vec{\nu}) = cw(Score(\vec{\nu}))f(\vec{\nu})$. Denote obtained new state by $\tilde{\nu}$
 8          $w[Score(\tilde{\nu})] \leftarrow w[Score(\tilde{\nu})]/C$
 9          $Hist[Score(\tilde{\nu})] \leftarrow Hist[Score(\tilde{\nu})] + 1$
10      end
11      $C \leftarrow \sqrt{C}$
12  end
13  return $w(S)$
```

Typically we use $C_{min} = \exp(0.6)$, $C_{max} = \exp(0.0000367)$. The criterion for a "sufficiently flat" histogram is that counts in every bin of the histogram are larger than 70% and smaller than 130% of the value expected in a perfectly flat histogram.

Use of Metropolis-Hastings algorithm coupled with Wang-Landau sampling is a common technique used recently for rare event sampling (see [9] for an extensible review). In [21] it was used to calculate the probabilities of sequence local alignment scores (however, neither accuracy estimates in the form of variance nor the sample sizes required to achieve the desired accuracy of estimates were given).

2.4 Variance Estimation

The estimate alone is useless without knowing how accurate it is. Regardless of the length of the simulation, there will be an unknown Monte Carlo error, $\widehat{p}_{IS} - p$. While it is impossible to assess this error directly, we can obtain its approximate sampling distribution through a Markov Chain central limit theorem (CLT) [18]. That is, if

$$\sqrt{N}\,(\widehat{p}_{IS} - p) \to N(0, \sigma_p^2),$$

as $N \to \infty$ with some $\sigma_p^2 > 0$. Denote by λ_p^2 the posterior variance associated with p. Then it is important to note that due to the correlation present in a Markov chain $\sigma_p^2 \neq \lambda_p^2$.

For now, suppose we have an estimator $\widehat{\sigma}_N^2$ such that $\widehat{\sigma}_N^2 \to \sigma_p^2$ almost surely as $N \to \infty$. This allows construction of a $(1-\delta)100\%$ confidence interval C_N for p by

$$C_N = (\widehat{p}_{IS} - z_{\delta/2}\widehat{\sigma}_N/\sqrt{N}; \widehat{p}_{IS} + z_{\delta/2}\widehat{\sigma}_N/\sqrt{N}), \tag{7}$$

where $z_{\delta/2}$ is a quantile of a standard Normal distribution. The width w_δ of C_N is given by

$$w_\delta = 2z_{\delta/2}\widehat{\sigma}_N/\sqrt{N}$$

and allows reporting the uncertainty of estimate $\widehat{p}_{IS}$.

There are many strongly consistent variance estimation techniques applicable for $\widehat{p}_{IS}$ including batch means [4, 10], spectral variance estimators[4] and regenerative simulation [8, 16].

Unfortunately, all these methods require storing the entire trajectory of the Markov chain to allow for the recalculations as the batch size increases with N. This might quickly become a problem if the fixed accuracy criterion is used as a stopping rule for the simulation process. Indeed, while storage capabilities overall are gradually becoming less and less of an issue, still, in order to obtain proper estimates in this case one would need to recalculate them over the length of the entire chain over and over again, which would make the process prohibitively computationally expensive.

Most likely the first recursive approach to update a σ_p^2 estimate when new observations come with $O(1)$ memory and computational complexity was proposed in [22]. The challenge here is to figure out a way to determine the batch sizes recursively to preserve consistency of the estimates and have a small mean square error, while simultaneously keeping the computational and computer memory requirements low. We are using a novel recursive estimator for σ_p^2 proposed in [23] that in the most situations works better than the estimator from [22] while preserving the $O(1)$ storage requirements.

2.5 Stopping Rule

In order to be able to process big MS/MS databases we need to carry out PSM significance estimation *en masse* in a fully automated manner. It follows that in this case performing chain diagnostics by hand or using a fixed time Markov chain stopping rule is out of the question. Recently in [3] an automated sequential stopping procedure was proposed that terminates the simulation when the computation uncertainty is small relative to the posterior uncertainty. In [6] it was shown that this stopping rule is equivalent to stopping when the effective sample size is sufficiently large.

Let $\widehat{\lambda}_N$ be an estimator of λ_p and consider a relative standard deviation fixed-width stopping rule, i.e.

$$N_\epsilon = \inf\left\{N > 0 : 2z_{\delta/2}\widehat{\sigma}_N/\sqrt{N} \leq \epsilon\widehat{\lambda}_N\right\}. \tag{8}$$

From [3] it follows that if $\widehat{\lambda}_N \to \lambda_p$ a.s. and $\widehat{\sigma}_N \to \sigma_p$ a.s. as $N \to \infty$, then as $\epsilon \to 0$ the simulations will terminate with probability 1 and $\mathbb{P}(p \in C_{N_\epsilon}) \to 1 - \delta$. In practice we are using $\epsilon = 0.02$ and the $\widehat{\sigma}_N$ estimate from [23].

A useful modification of this stopping criterion comes from the specific MS/MS database search problem statement. In certain situations the aim is not to estimate the probability of interest (2), but to decide whether p satisfies $p < p_0$ with p_0 being some fixed threshold. Usually p_0 is much larger compared to p (e.g. $p_0 = 10^{-7}$ and $p < 10^{-10}$). Therefore in addition to checking a condition (8) for a particular N we could also check if $p_0 \notin C_N$. If this is indeed so, then it automatically implies that either $p < p_0$ or $p > p_0$ with probability $1 - \delta$ as $N \to \infty$. This addition to the stopping rule might result in a significant reduction of the amount of simulations required, since it would depend on a much larger p_0 and not p.

2.6 Outline of the Algorithm

Gathering all the parts of the proposed method together we end with the following algorithm to compute $\widehat{p}_{IS}$.

Algorithm 4: Importance Sampling estimator for p

```
  Input  : A peptide $P$ and spectrum $Spectrum$
  Output : An estimate $\widehat{p}_{IS}$ of statistical significance $p$ and confidence interval $C_N$
1 Construct PNP graph $G$ of a peptide $P$ and determine the set of cuts $\mathcal{C}$
2 Construct fragmentation matrix $H$ and let $Score(\mu) = SPCScore(Spectrum, H\mu)$
3 Determine the set of weights $w(S)$ using Wang-Landau algorithm (see algorithm 3)
4 while Stopping criterion (8) is not satisfied do
5     Simulate next state $\vec{\nu}_N$ using Metropolis-Hastings algorithm (see algorithm 1)
6     Update estimates $\widehat{\sigma}_N$ and $\widehat{\lambda}_N$
7 end
8 Calculate $\widehat{p}_{IS}$ using (6) and confidence interval $C_N$ via (7).
```

3 Results

To confirm the validity of our approach, we have made a point to verify the accuracy of our calculations in a number of different ways. First, we have chosen a number of linear, cyclic and branch-cyclic peptides and selected several PSMs that had not extremely small probability of interest (say, within the $10^{-8} - 10^{-6}$ range). This allowed us to calculate them via direct Monte Carlo sampling, construct confidence intervals and compare the variances. For cyclic peptides we have chosen five examples from [15, Table 1], namely cyclic peptides $(10, 20, 40)$, $(10, 20, 40, 80)$, $(10, 20, 40, 80, 160)$, $(10, 20, 40, 80, 160, 320)$, and $(10, 20, 40, 80, 160, 320, 640)$. The branch-cyclic example is *Surfactin* test dataset for the Dereplicator algorithm described in [14].

We denote $\widehat{p}_{MC}$ as the probability estimate calculated via Monte Carlo sampling, $\widehat{p}_{IS}$ as the probability estimate calculated via the proposed algorithm (importance sampling via MCMC), and $\widehat{p}_{DPR}$ as the probability calculated by the MS-DPR algorithm from [14]. Note that the latter does not provide any accuracy estimate and therefore we were unable to construct confidence interval for $\widehat{p}_{DPR}$. $\widehat{p}_{MC}$ were calculated via $N = 50 \cdot 10^6$ simulations, $\widehat{p}_{IS}$ were calculated using the stopping rule (8) with $\epsilon = 0.02$, and $\widehat{p}_{DPR}$ were calculated by Dereplicator, using default settings.

Table 1 Comparison of Monte Carlo, MCMC and MS-DPR approaches: estimates.

Peptide	$\widehat{p}_{IS}$	$\widehat{p}_{MC}$	$\widehat{p}_{DPR}$
PPAEDSQK	$4.87 \cdot 10^{-7}$	$4.20 \cdot 10^{-7}$	$6.6 \cdot 10^{-7}$
GQGDPGSNPNK	$4.70 \cdot 10^{-7}$	$6.40 \cdot 10^{-7}$	$1.5 \cdot 10^{-8}$
HSNAAQTQTGEANR	$2.39 \cdot 10^{-6}$	$2.22 \cdot 10^{-6}$	$4.9 \cdot 10^{-8}$
GEEEPSQGQK	$1.03 \cdot 10^{-6}$	$1.04 \cdot 10^{-6}$	$3.6 \cdot 10^{-7}$
$(10, 20, 40)$	0.00184	0.00184	0.00197
$(10, 20, 40, 80)$	$7.35 \cdot 10^{-6}$	$7.34 \cdot 10^{-6}$	$9.36 \cdot 10^{-6}$
$(10, 20, 40, 80, 160)$	$6.76 \cdot 10^{-9}$	N/A	$4.49 \cdot 10^{-9}$
$(10, 20, 40, 80, 160, 320)$	$1.74 \cdot 10^{-12}$	N/A	$1.56 \cdot 10^{-12}$
$(10, 20, 40, 80, 160, 320, 640)$	$4.08 \cdot 10^{-16}$	N/A	N/A
Surfactin	$1.18 \cdot 10^{-5}$	$1.13 \cdot 10^{-5}$	$1.01 \cdot 10^{-5}$

Table 2 Comparison of Monte Carlo, MCMC and MS-DPR approaches: 95% confidence intervals.

Peptide	Conf. interval, $\widehat{p}_{IS}$		Conf. interval, $\widehat{p}_{MC}$	
PPAEDSQK	$4.74 \cdot 10^{-7}$	$4.99 \cdot 10^{-7}$	$2.40 \cdot 10^{-7}$	$6.00 \cdot 10^{-7}$
GQGDPGSNPNK	$4.53 \cdot 10^{-7}$	$4.87 \cdot 10^{-7}$	$4.18 \cdot 10^{-7}$	$8.62 \cdot 10^{-7}$
HSNAAQTQTGEANR	$2.30 \cdot 10^{-6}$	$2.48 \cdot 10^{-6}$	$1.81 \cdot 10^{-6}$	$2.63 \cdot 10^{-6}$
GEEEPSQGQK	$9.96 \cdot 10^{-7}$	$1.07 \cdot 10^{-6}$	$7.57 \cdot 10^{-7}$	$1.32 \cdot 10^{-6}$
$(10, 20, 40)$	$1.80 \cdot 10^{-3}$	$1.88 \cdot 10^{-3}$	$1.82 \cdot 10^{-3}$	$1.85 \cdot 10^{-3}$
$(10, 20, 40, 80)$	$7.12 \cdot 10^{-6}$	$7.58 \cdot 10^{-6}$	$6.60 \cdot 10^{-6}$	$8.10 \cdot 10^{-6}$
$(10, 20, 40, 80, 160)$	$6.4 \cdot 10^{-9}$	$7.10 \cdot 10^{-9}$	N/A	N/A
$(10, 20, 40, 80, 160, 320)$	$1.51 \cdot 10^{-12}$	$1.97 \cdot 10^{-12}$	N/A	N/A
$(10, 20, 40, 80, 160, 320, 640)$	$3.60 \cdot 10^{-16}$	$4.55 \cdot 10^{-16}$	N/A	N/A
Surfactin	$1.14 \cdot 10^{-5}$	$1.22 \cdot 10^{-5}$	$1.03 \cdot 10^{-5}$	$1.23 \cdot 10^{-5}$

Tables 1 and 2 summarize these results. As can be seen from these tables, the confidence intervals constructed from $\widehat{p}_{IS}$ lie within the confidence intervals for $\widehat{p}_{MC}$ and often have significantly smaller lengths. $\widehat{p}_{DPR}$ falls outside the confidence intervals and often is biased downwards. We must note that this property of $\widehat{p}_{DPR}$ could easily lead to false discoveries and certainly inflates the number of significant PSMs in the applications. Also, the sample size N of $50 \cdot 10^6$ was not enough to estimate $\widehat{p}_{MC}$ for $(10, 20, 40, 80, 160)$, $(10, 20, 40, 80, 160, 320)$, and $(10, 20, 40, 80, 160, 320, 640)$ and MS-DPR failed to calculate $\widehat{p}_{DPR}$ for the last peptide.

In the next series of experiments we study the variance of $\widehat{p}_{IS}$ and compare it to that of $\widehat{p}_{MC}$. In order to do so, we calculate $\widehat{p}_{MC}$ using the same number of simulations N as it was used to calculate $\widehat{p}_{IS}$[2]. Table 3 shows the reduction of variance of $\widehat{p}_{IS}$ compared to the $\widehat{p}_{MC}$. Overall, it could be observed that the smaller the probability is, the larger does the difference between the variances of MCMC and Monte Carlo estimators end up being.

Finally, in order to verify the scalability and applicability of the proposed method, the run of the DEREPLICATOR algorithm was performed on the entirety of the Global Natural Products Social (GNPS) molecular network [20] database. This allowed us to compare the

[2] We have to increase the sample size to obtain observations with desired target score 13 to allow $\widehat{\sigma}^2_{MC}$ estimation for GQGDPGSNPNK.

Table 3 Comparison of Monte Carlo and MCMC approaches: variances.

Peptide	$\hat{\sigma}^2_{IS}$	$\hat{\sigma}^2_{MC}$	$\hat{\sigma}^2_{MC}/\hat{\sigma}^2_{IS}$	Sample Size
PPAEDSQK	$2.09 \cdot 10^{-10}$	$4.94 \cdot 10^{-7}$	2358.98	5000000
GQGDPGSNPNK	$2.33 \cdot 10^{-10}$	$1.49 \cdot 10^{-7}$	639.49	20000000[2]
HSNAAQTQTGEANR	$5.56 \cdot 10^{-9}$	$2.24 \cdot 10^{-6}$	403.23	2800000
GEEEPSQGQK	$1.23 \cdot 10^{-9}$	$7.89 \cdot 10^{-7}$	642.19	3800000
$(10, 20, 40)$	$5.47 \cdot 10^{-4}$	$1.88 \cdot 10^{-3}$	3.43	1500000
$(10, 20, 40, 80)$	$9.93 \cdot 10^{-8}$	$7.60 \cdot 10^{-6}$	76.53	7500000
Surfactin	$1.15 \cdot 10^{-7}$	$1.00 \cdot 10^{-5}$	86.96	2000000

Table 4 Comparison of $\widehat{p}_{IS}$ and $\widehat{p}_{DPR}$ on GNPS data. The number of target and decoy database matches and FDR estimates at different significance levels are shown.

	MSDPR			MCMC		
$-\log_{10} p$	target	decoy	$\widehat{FDR}$ %	target	decoy	$\widehat{FDR}$ %
7	762	188	19.78	744	179	19.39
8	619	110	15.08	610	104	14.56
9	505	52	9.33	473	51	9.73
10	443	33	6.93	415	30	6.74
11	393	21	5.07	354	20	5.34
12	354	15	4.06	312	12	3.70
13	322	11	3.30	271	7	2.51
14	293	11	3.61	238	2	0.83
15	264	7	2.58	201	1	0.49
16	238	5	2.05	169	0	0.0
17	211	2	0.93	138	0	0.0
18	188	0	0.0	104	0	0.0
19	157	0	0.0	87	0	0.0
20	139	0	0.0	76	0	0.0

devised algorithm with the MS-DPR estimate used by DEREPLICATOR by default. Table 4 shows an overview of the obtained results. The False Discovery Rate estimate is calculated in DEREPLICATOR via the target-decoy approach [1]. Table 4 shows that $\widehat{p}_{IS}$ yields a smaller number of significant decoy matches compared to $\widehat{p}_{DPR}$ and therefore less FDR. This could easily be explained by the fact that $\widehat{p}_{DPR}$ are biased downwards.

4 Summary

We presented the importance sampling-based estimator that is capable of accurately and quickly assess the significance of peptide spectrum matches. Given its generic nature, it could be easily modified to be used with a great number of different score functions, fragmentation models and amino acid mass distributions. The proposed estimation algorithm has been integrated into DEREPLICATOR[3] and VARQUEST[4] tools and publicly available as a part of these packages.

[3] `http://cab.spbu.ru/software/dereplicator/`
[4] `http://cab.spbu.ru/software/varquest/`

Acknowledgement. The authors would like to extend a special thanks to Alexey Gurevich for running Dereplicator on GNPS, Seungjin Na for making the spectra and PSMs available for linear peptides and Alex Shlemov for all the fruitful discussions that were of great help in improving the algorithm.

References

1 J. E. Elias and S. P. Gygi. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nat. Methods*, 4(3):207–214, 2007.

2 David Fenyö and Ronald C. Beavis. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. *Analytical Chemistry*, 75(4):768–774, 2003.

3 James M. Flegal and Lei Gong. Relative fixed-width stopping rules for Markov Chain Monte Carlo simulations. *Statistica Sinica*, 25(2):655–675, 2015.

4 James M. Flegal and Galin L. Jones. Batch means and spectral variance estimators in Markov Chain Monte Carlo. *Ann. Statist.*, 38(2):1034–1070, 2010.

5 A. M. Frank. Predicting intensity ranks of peptide fragment ions. *J. Proteome Res.*, 8(5):2226–2240, 2009.

6 Lei Gong and James M. Flegal. A practical sequential stopping rule for high-dimensional Markov Chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 25(3):684–700, 2016.

7 W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

8 James P. Hobert, Galin L. Jones, Brett Presnell, and Jeffrey S. Rosenthal. On the applicability of regenerative simulation in markov chain monte carlo. *Biometrika*, 89(4):731, 2002.

9 Yukito Iba, Nen Saito, and Akimasa Kitajima. Multicanonical MCMC for sampling rare events: an illustrative review. *Annals of the Institute of Statistical Mathematics*, 66(3):611–645, 2014.

10 Galin L. Jones, Murali Haran, Brian S. Caffo, and Ronald Neath. Fixed-width output analysis for Markov Chain Monte Carlo. *Journal of the American Statistical Association*, 101(476):1537–1547, 2006.

11 Sangtae Kim, Nitin Gupta, and Pavel A. Pevzner. Spectral probabilities and generating functions of tandem mass spectra: A strike against dgegeecoy databases. *Journal of Proteome Research*, 7(8):3354–3363, 2008.

12 Sangtae Kim, Nikolai Mischerikow, Nuno Bandeira, J. Daniel Navarro, Louis Wich, Shabaz Mohammed, Albert J. R. Heck, and Pavel A. Pevzner. The generating function of CID, ETD, and CID/ETD pairs of tandem mass spectra: Applications to database search. *Molecular & Cellular Proteomics*, 9(12):2840–2852, 2010.

13 D. P. Landau, Shan-Ho Tsai, and M. Exler. A new approach to Monte Carlo simulations in statistical physics: Wang-landau sampling. *American Journal of Physics*, 72(10):1294–1302, 2004.

14 H. Mohimani, A. Gurevich, A. Mikheenko, N. Garg, L. F. Nothias, A. Ninomiya, K. Takada, P. C. Dorrestein, and P. A. Pevzner. Dereplication of peptidic natural products through database search of mass spectra. *Nat. Chem. Biol.*, 13(1):30–37, 2017.

15 H. Mohimani, S. Kim, and P. A. Pevzner. A new approach to evaluating statistical significance of spectral identifications. *J. Proteome Res.*, 12(4):1560–1568, 2013.

16 Per Mykland, Luke Tierney, and Bin Yu. Regeneration in Markov chain samplers. *Journal of the American Statistical Association*, 90(429):233–241, 1995.

17 G. O. Roberts. Markov chain concepts related to sampling algorithms. In W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors, *Markov Chain Monte Carlo in Practice*, pages 45–58. Chapman & Hall, London, 1996.

18 Luke Tierney. Markov chains for exploring posterior distributions. *Ann. Statist.*, 22(4):1701–1728, 1994.

19 F. Wang and D. P. Landau. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters*, 86:2050–2053, 2001.

20 M. Wang, J. J. Carver, V. V. Phelan, L. M. Sanchez, et al. Sharing and community curation of mass spectrometry data with Global Natural Products Social Molecular Networking. *Nat. Biotechnol.*, 34(8):828–837, 2016.

21 Stefan Wolfsheimer, Inke Herms, Sven Rahmann, and Alexander K. Hartmann. Accurate statistics for local sequence alignment with position-dependent scoring by rare-event sampling. *BMC Bioinformatics*, 12(1):47, 2011.

22 Wei Biao Wu. Recursive estimation of time-average variance constants. *Ann. Appl. Probab.*, 19(4):1529–1552, 2009.

23 Chun Yip Yau and Kin Wai Chan. New recursive estimators of the time-average variance constant. *Statistics and Computing*, 26(3):609–627, 2016.

abSNP: RNA-Seq SNP Calling in Repetitive Regions via Abundance Estimation[*]

Shunfu Mao[1], Soheil Mohajer[2], Kannan Ramachandran[3], David Tse[4], and Sreeram Kannan[5]

1 **Department of Electrical Engineering, University of Washington, Seattle, WA, USA**
`shunfu@uw.edu`

2 **Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA**
`soheil@umn.edu`

3 **Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA**
`kannanr@eecs.berkeley.edu`

4 **Department of Electrical Engineering, Stanford University, Stanford, CA, USA**
`dntse@stanford.edu`

1 **Department of Electrical Engineering, University of Washington, Seattle, WA, USA**
`ksreeram@uw.edu`

Abstract

Variant calling, in particular, calling SNPs (Single Nucleotide Polymorphisms) is a fundamental task in genomics. While existing packages offer excellent performance on calling SNPs which have uniquely mapped reads, they suffer in loci where the reads are multiply mapped, and are unable to make any reliable calls. Variants in multiply mapped loci can arise, for example in long segmental duplications, and can play important role in evolution and disease.

In this paper, we develop a new SNP caller named abSNP, which offers three innovations. (a) abSNP calls SNPs from RNA-Seq data. Since RNA-Seq data is primarily sampled from gene regions, this method is inexpensive. (b) abSNP is able to successfully make calls on repetitive gene regions by exploiting the quality scores of multiply mapped reads carefully in order to make variant calls. (c) abSNP exploits a specific feature of RNA-Seq data, namely the varying abundance of different genes, in order to identify which repetitive copy a particular read is sampled from.

We demonstrate that the proposed method offers significant performance gains on repetitive regions in simulated data. In particular, the algorithm is able to achieve near-perfect sensitivity on high-coverage SNPs, even when multiply mapped.

1998 ACM Subject Classification J.3 Life and Medical Sciences

Keywords and phrases RNA-Seq, SNP Calling, Repetitive Region, Multiply Mapped Reads, Abundance Estimation

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.15

[*] This work of SK and SM were supported, in part, by U.S. National Institute of Health grant 5R01HG008164-02 (SK and SM) and U.S. National Science Foundation CAREER grant 1651236 (SK). The work of DNT was supported in part by the Center for the Science of Information and in part by the NIH grant R01HG008164.

1 Introduction

Decoding individual-specific (or even tissue or cell-specific) variations with respect to a reference genome is an important task, downstream of DNA sequencing. Among the variations that can be detected with high throughput sequencing data, the most frequently called variants are single nucleotide polymorphisms (SNPs), which specify single base loci at which the target sequence differs from the reference allele. There are tens of millions of SNPs in a human genome (which has 3 billion bases), and a reliable detection of them (i.e. SNP calling) is an important task because they are relevant in predicting organismal traits as well as implicated in several diseases.

Existing SNP calling softwares (i.e. DNA-Seq SNP callers), such as GATK [6], GlfMultiples [1], SAMtools mpileup [16], FreeBayes [8] and VarScan [14], mainly rely on whole genome sequencing (WGS) or whole exome sequencing (WES). While WGS can call SNPs throughout the entire genome, many downstream pipelines only consider SNPs in gene exon regions, as their impact is easier to quantify. Therefore, WES is a widely-used cheaper alternative, which focuses on DNA reads from the exonic regions. A third strategy is to utilize RNA-seq reads from a tissue of interest and use those reads both for (a) expression estimation as well as (b) variant calling. Beside being fast and inexpensive [3], this third strategy is advantageous when the genes harboring SNPs of interest are likely to have non-negligible expression, such as in cancer tissue analysis. However existing DNA-Seq SNP callers are not suitable to properly handle RNA-Seq data directly. One reason is that RNA-Seq reads may be sampled from two different exons, and these splice junctions are typically not captured by these callers. In addition, RNA-Seq data also has a specific feature, namely the varying expression of different genes, with expression levels varying over several orders of magnitude.

To address the above mentioned challenges, designing SNP callers tailored for RNA-Seq data (RNA-Seq SNP callers) is necessary. There are only a limited number of works on RNA-Seq SNP calling, such as GATK, eSNV-detect [22], SNPiR [19] and SNVMix [10]. eSNV-detect and SNPiR essentially rely on SAMtools mpileup and GATK to call SNPs respectively, and SNVMix depends on SAMtools mpileup to prepare necessary statistics for SNP calling. Among these, only GATK is still under constant maintenance and development.

Even though most existing SNP callers (especially GATK) offer excellent performance on benchmark sets, these sets are usually only representative of regions in the genome without repeats. On the repetitive regions, where reads are not uniquely mapped, most callers are unable to make any reliable calls, since they simply discard all of the multiply mapped reads, and consequently corresponding SNP information will be missed. We note that even projects that are designed to catalogue the performance of SNP callers, such as Genome in a bottle project [23], consider high-quality calls only in non-repetitive regions. This limitation fundamentally comes from the fact that existing read aligners are unable to differentiate between multiply mapped reads, and therefore cannot make any predictions on the origin of the SNP with confidence. There are some studies regarding DNA-Seq SNP calling that consider multiply mapped reads, such as Sniper [21], SiRen [5] and GW-CALL [9], but to our best knowledge there is no work yet on RNA-Seq SNP Calling that addresses the problem of multiply mapped reads on repetitive genomic regions.

To fill this gap, we've designed **abSNP** ("a" stands for abundance and "b" stands for Bayesian principles. abSNP is written in Python and is freely available at `https://github.com/shunfumao/abSNP`), which is a novel RNA-Seq SNP calling software that is able to call SNPs even in repetitive regions. The key idea, as illustrated in Figure 1, is to use the products of abundance estimation (or called quantification), which include estimated

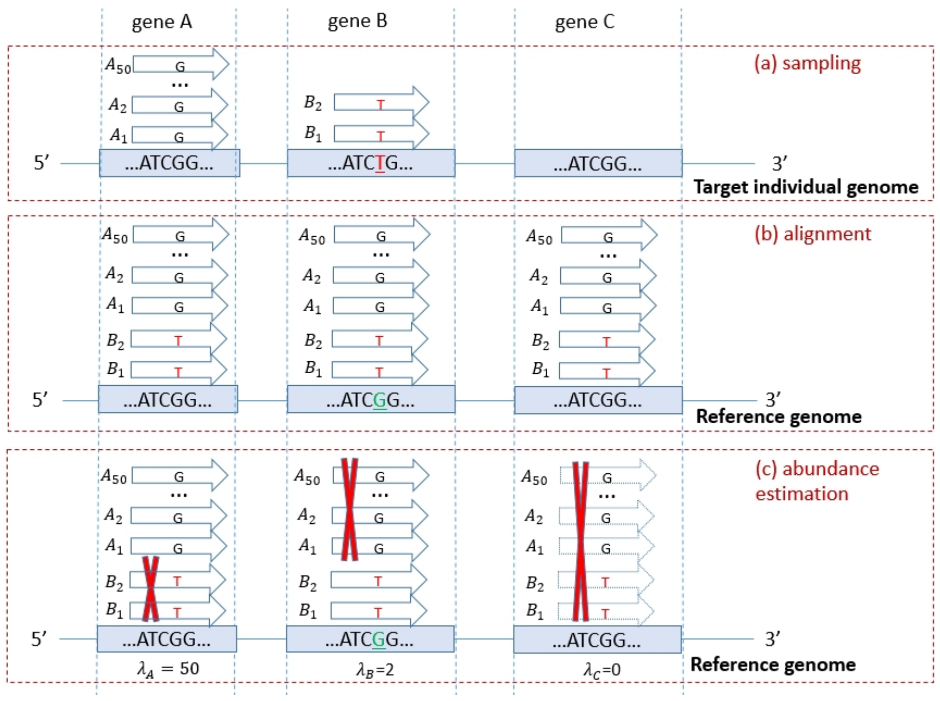


Figure 1 The utility of abundance estimation in SNP calling. Suppose target individual genome has (approximate) repetitive regions inside genes A, B and C, and gene B contains a SNP of $G \rightarrow T$. **Step (a)** shows sampling RNA reads from transcriptome (e.g. set of RNA transcripts), where reads $\{A_1, A_2, \ldots, A_{50}\}$ and $\{B_1, B_2\}$ are sampled from genes A and B, respectively. In **step (b)** we align reads onto a reference genome. However it is possible that all reads are mapped onto all three genes because of their similarity. One may rely on $\{B_1, B_2\}$ and call SNPs in all three genes, therefore bringing wrong calls (false positives) in gene A and C. Alternatively, existing methods discard all these reads, which results in a false negative (a true SNP not detected) of the SNP in gene B. Now let us consider an additional RNA-Seq abundance estimation procedure in **step (c)**: One byproduct we can obtain is rich information of mapping quality scores for reads. Suppose the mapped reads (in dotted shape) onto gene C are therefore known to have very low mapping scores (e.g. < 0.1), we then exclude them for SNP calling. The other product we can obtain is the estimated gene abundance. Suppose the abundance levels (the number of reads per locus here) of genes A, B and C are therefore known to be $\lambda_A = 50, \lambda_B = 2$ and $\lambda_C = 0$. Then we can say reads $\{B_1, B_2\}$ are more likely to come from gene B, while $\{A_1, A_2, \ldots, A_{50}\}$ are probably sampled from gene A. Consequently, we call the SNP correctly in gene B.

gene expression levels as well as rich information of read mapping quality scores. As far as we know, such kind of information has not been exploited yet for RNA-Seq SNP calling. We demonstrate that utilizing such information leads to significant gains in SNP calling performance. In comparison to existing callers that are unable to make any calls in multiply mapped regions, abSNP is able to get significantly increased sensitivity. In particular, in SNPs that have high coverage, abSNP demonstrates near perfect sensitivity, making it a viable alternative to existing SNP callers.

2 Method

2.1 Problem Statement

In this work, our goal is to call SNPs in diploid genome based on RNA-Seq data. The input to our caller is the set of RNA-Seq reads sampled from the transcriptome (i.e. set of RNA transcripts). Our goal is to identify SNPs located within the gene regions of the target individual, i.e., loci at which the target genome is different from a known reference genome. To this end, we use the standard technique of read alignment of the sampled reads onto the reference sequence, and compare the nucleotides on the mapped reads to those of the reference genome to call SNPs.

There are several challenges that need to be addressed: (i) The most important factor is due to existence of (approximately) repetitive regions in the target/reference; reads sampled from repetitive regions get mapped to multiple loci, and the algorithm has to figure out where they are sampled from. (ii) Not all the reads sampled from a locus carry SNP information. This is due to the heterozygous SNPs, in which one of the alleles contain a SNP, and the other one matches with the reference genome. (iii) A unique feature of RNA-Seq data is that there is potentially a wide gap between the number of reads sampled from the paternal and maternal alleles, due to the varying expression levels of the corresponding genes.

2.1.1 Assumptions

To handle the above-mentioned challenges, we develop abSNP based on the following three key assumptions in order to simplify our modeling of the problem:

(i) **Heterozygous SNPs:** We assume that SNP only appears in one of the paternal or the maternal allele, while the other allele is consistent with the reference genome. Our methods can also detect SNP occurring in both alleles (i.e. homozygous SNP), but further refinement is needed to distinguish whether a SNP occurs in one or both of the alleles.

(ii) **Equal allele contribution:** This means each paternal and maternal allele contributes equally to the abundance for each genomic locus[1].

(iii) **Single SNP across repetitive regions:** When there are repetitive regions, we assume that at most one copy has a SNP at a given locus. This assumption is valid since the probability of SNP is small ($p \approx 0.001$) and the probability of two SNPs p^2 is negligible. We note that each copy of a repetitive region can have many SNPs; just that they do not occur at the same base locus.

2.2 Definitions

We briefly review a set of terms and notations (as summarized in Figure 2) that are useful for presentation of the algorithm and the following discussions.

The proposed SNP calling procedure examines genomic loci one-by-one, to identify whether a SNP occurs at each locus or not. While processing the sequence at locus i, the base of the reference genome (**reference base**) at locus i is denoted by $r \in \{A, C, G, T\}$. In a typical scenario where no SNP exists, both the paternal and maternal alleles of the individual *target* also have base r at the current locus i. A locus i is called a SNP if among the two alleles of the individual target, one allele (**non-SNP allele**) has base r, and the other allele (**SNP allele**) has $x \neq r$ (recall the heterozygous SNP assumption).

After read alignment, a subset of J reads (denoted by $\mathcal{R} = \{R_1, R_2, \ldots, R_J\}$) sampled from the target alleles will be mapped onto the reference genome so that they cover locus i. We denote by λ_0 the expression level (abundance) of locus i, which can be estimated by quantifying the transcripts with observed reads, for example, by using RSEM [15]. For each

[1] This assumption is used to develop our algorithm, however, this assumption is not critical. In our actual evaluation each allele has a randomly assigned (thus different) expression levels.

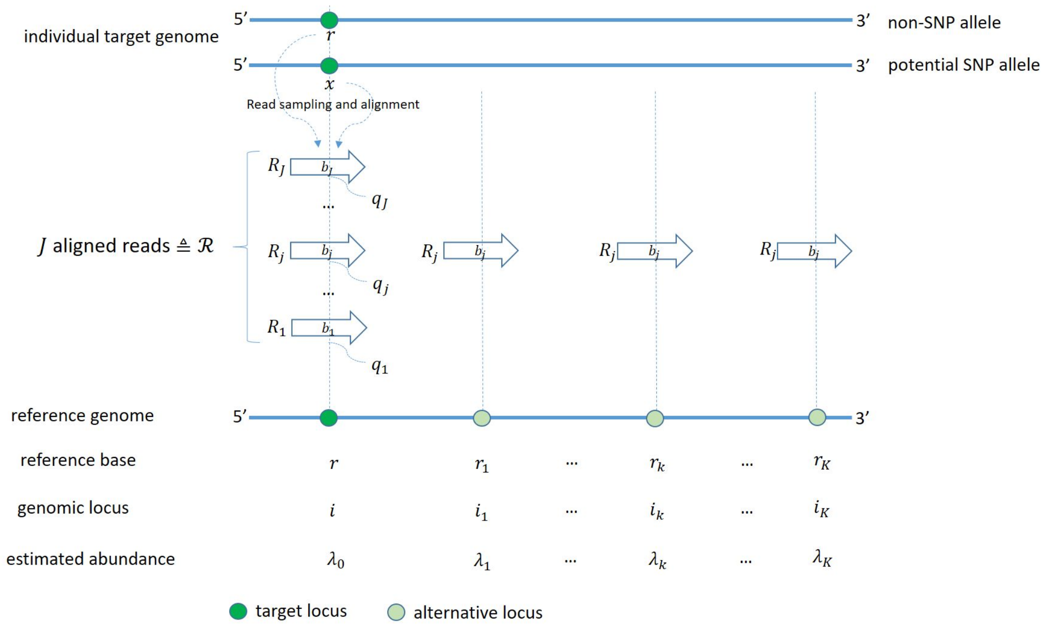


Figure 2 A typical scenario of SNP calling at locus i.

R_j, we denote its base covering locus i in alignment by b_j, and denote its base quality[2] at locus i by q_j. A read R_j is called a **SNP read** if its read base differs from the reference base, i.e. $b_j \neq r$.

As illustrated in Figure 2, these reads may be also mapped to other repetitive regions of the reference genome. The corresponding bases of multiply mapped reads will cover other loci of the reference, which are called **alternative loci** for locus i, and will be denoted by $\{i_1, i_2, \dots, i_K\}$. Their respective reference base and estimated abundance are denoted as r_k and λ_k for $k \in \{1, ..., K\}$.

2.3 Overall Flow

The core stages of the proposed caller are illustrated in Figure 3. The algorithm takes in raw reads, known transcriptome annotations (such as .gtf format), and reference genome, and first performs abundance estimation using RSEM [15], which estimates abundance for each RNA transcript. Based on RSEM results, we utilize the following outputs: (1) estimated abundance per genome locus, as well as (2) genome-based read alignments. Although it is also possible to use pure alignment softwares (e.g. STAR [7] or TopHat2 [13]) to obtain read alignments, our alternate process using RSEM offers rich mapping quality information which can be used to filter out noisy multiply mapped reads in a step called MAPQ filtering (Section 2.4). Based on the estimated abundance and read alignments, we perform Count Generation, where we collect necessary information (Figure 2 and Section 2.2) required for SNP calling per target locus. We then use our Bayesian SNP calling criteria to find SNP candidates for the target genome. This step is carried out using information at a given locus (including multiply mapped reads, their alternative loci and abundance). Since the SNP call

[2] The base quality is encoded in read file, and is different from mapping quality of a read, which is encoded in the read alignment file.

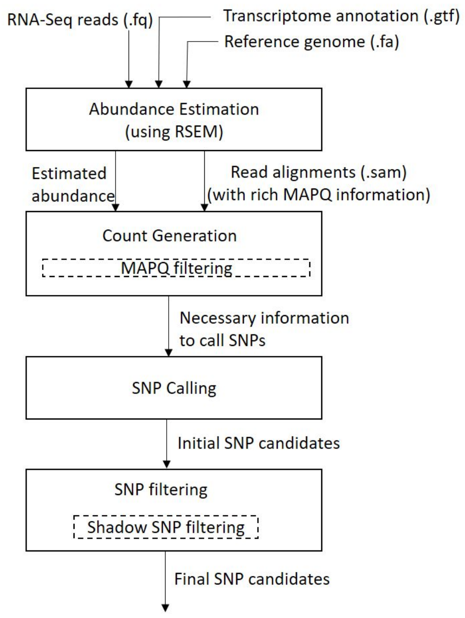


Figure 3 Overall Flow of abSNP.

at each locus is made independently, in order to share information between the multiple SNP calls, we have a filtering step that takes into account the calls at alternative loci; this step is called shadow SNP filtering, see Section 2.6.

2.4 MAPQ Filtering

MAPQ (MAPping Quality of read alignments) is a metric used to capture the confidence about mapping of a read to a reference region. As described in [17] as well as in official read alignment format [11], it is defined as: $-10\log_{10}(1 - P(\text{correct mapping}))$. Since a read can be multiply mapped onto different loci (i.e. repetitive genomic regions), a better knowledge of MAPQ for each alignment can potentially help us remove false alignments and consequently achieve a better SNP calling performance.

Though well defined, the MAPQ scores reported by existing RNA-Seq aligners (such as STAR and TopHat2) are usually uninformative and usually have same value for all of the multiply mapped reads. For example, in STAR (also similar in TopHat2), a uniquely mapped read will have MAPQ $= 255$ and a read multiply mapped onto N_{map} loci will have MAPQ $= -10\log_{10}(1 - \frac{1}{N_{map}})$ corresponding to $P(\text{correct mapping}) = \frac{1}{N_{map}}$. If the read maps equally well to all possible loci, it appears that there may be no way to get further information.

However, when one considers multiple reads, it is possible to get additional information, since each gene has a differing abundance, which when estimated, modifies the posterior probability of mapping. In other words, we can obtain a more informative mapping quality measure as a side product of RNA-Seq abundance estimation. Typically, an Expectation-Maximization (EM) algorithm is involved, which alternates between the two steps: (1) given the read alignments onto RNA transcripts, the abundance of transcripts is estimated; (2) given the abundance of transcripts, the read alignment probabilities are refined. This iterative procedure calculates the probability that a given read is assigned to a particular genomic locus, and therefore can be used as a sharper estimate of MAPQ.

Here we use RSEM [15], a software extensively used for abundance estimation, to provide us with refined MAPQ scores. We then filter out some of the low quality read alignments with MAPQ scores lower than certain threshold (e.g. 0.1) via the MAPQ filtering process. We empirically choose this threshold, since we find this helps effectively removing false alignments of multiply mapped reads that may cause false positives.

To the best of our knowledge, our algorithm is the first to use abundance estimators for RNA-Seq SNP calling. They provide us with not only better MAPQ scores, but also estimates of abundance levels required by our algorithm to detect (as in Section 2.5) and refine (as in Section 2.6) SNP calls. While we choose RSEM in our current implementation due to its popularity, it is also possible to replace RSEM with other abundance estimators such as eXpress [20].

2.5 SNP Calling Algorithm

Here we describe our core SNP calling algorithm. Our algorithm runs over all loci, and for a given locus i, it examines whether i consists of a SNP. Consequently, throughout this section we present the algorithm for a given locus i (as illustrated in Figure 2), and hence dependency of variables on i is eliminated, whenever it is clear from the context.

Based on the assumptions of equal allele contribution and heterozygous SNPs (Section 2.1.1), at locus i we have two target alleles: one allele with base r (identical to the reference sequence) and abundance $\frac{\lambda_0}{2}$, and the other allele with base $x \in \{A, C, G, T\}$ and abundance $\frac{\lambda_0}{2}$. There is a SNP at locus i if and only if $x \neq r$.

At locus i, we try to estimate the corresponding x using maximum a-posterior probability (MAP) estimation:

$$\hat{x} = \arg \max_{x \in \{A,C,G,T\}} P(X = x|\mathcal{R}) = \arg \max_{x \in \{A,C,G,T\}} P(\mathcal{R}|X = x)P(X = x) \tag{1}$$

where $\mathcal{R} = \{R_1, \ldots, R_J\}$ is the set of reads mapped over locus i of the reference genome, and X is a random variable, which represents possible base at locus i of the potential SNP allele.

The second equation holds due to $P(X = x|\mathcal{R}) = \frac{P(\mathcal{R}|X=x)P(X=x)}{P(\mathcal{R})}$ (according to the Bayes' theorem) and the fact that $P(\mathcal{R})$ is the same for all values of x. Here $P(X = x)$ can be further expressed as:

$$P(X = x) = \begin{cases} \frac{P_{\mathsf{SNP}}}{3} & \text{if } x \in \{A, C, G, T\} \setminus \{r\} \\ 1 - P_{\mathsf{SNP}} & \text{if } x = r \end{cases} \tag{2}$$

where P_{SNP} indicates the prior probability (i.e. general knowledge) for a SNP to occur per genomic locus[3].

In order to solve the optimization in (1) we also need to find $P(\mathcal{R}|X = x)$. A common approach is to assume reads are independent from each other (as used in [17]), so we have:

$$P(\mathcal{R}|X = x) = \Pi_{j=1}^{J} P_j = \Pi_{j=1}^{J} P(R_j = b_j|X = x, r, q_j, \lambda_{b_j}, \lambda_\Sigma) \tag{3}$$

Here P_j indicates the probability of the j-th read (i.e. R_j) having base b_j at locus i, given all the other assumptions, including base x at the target, base r in the reference, and all related quality scores and abundance levels. In particular, q_j denotes the quality score of base b_j at the read, λ_{b_j} denotes the (sum of the) abundance level(s) of alternative loci that

[3] P_{SNP} can be set based on the knowledge of SNP rate of the genome of interest. Suppose there are around 10 million SNPs across human genome of 3 billion bases, then we set P_{SNP} as $\frac{10^7}{3\times10^9} \approx 3 \times 10^{-3}$.

the read can be mapped to and the reference has b_j, i.e., $\lambda_{b_j} = \sum_{k=1}^{K} \lambda_k \mathbf{1}\{r_k = b_j\}$, where $\mathbf{1}\{\cdot\}$ is an indicator function. Finally, for the current locus i and read R_j, λ_Σ is the total estimated abundance level, given by $\lambda_\Sigma = \lambda_0 + \sum_{k=1}^{K} \lambda_k$. Hence, we can further expand P_j as:

$$P(R_j = b_j | X = x, r, q_j, \lambda_{b_j}, \lambda_\Sigma) = \frac{1}{\lambda_\Sigma} \begin{cases} \lambda_0 q_j + \lambda_{b_j} & \text{if } b_j = x = r \\ \lambda_0(\frac{q_j}{3} + \frac{1}{6}) + \lambda_{b_j} & \text{if } b_j = x \neq r \\ \lambda_0(\frac{q_j}{3} + \frac{1}{6}) + \lambda_{b_j} & \text{if } b_j = r \neq x \\ \lambda_0 \frac{1-q_j}{3} + \lambda_{b_j} & \text{if } b_j \notin \{x, r\} \end{cases} \tag{4}$$

To understand Equation (4), let's first consider an R_j with no alternative mappings (i.e. $\lambda_{b_j} = 0$, $\lambda_\Sigma = \lambda_0$). For $b_j = x = r$, P_j is the probability that read R_j is sampled from target individual's paternal or maternal allele at locus i (which is 1) and no error has occurred (which happens with probability q_j): thus we have $P_j = 1 \times q_j = q_j$. If $b_j = x \neq r$, there are two possibilities for observing R_j: either P_j is the probability of sampling R_j from the SNP allele at locus i (which is $\frac{1}{2}$, due to assumption of equal allele contribution) without error (which is q_j), or P_j is the probability of sampling R_j from the non-SNP allele (which is $\frac{1}{2}$) with error (which is $\frac{1-q_j}{3}$). Therefore, $P_j = \frac{1}{2} q_j + \frac{1}{2}\frac{1-q_j}{3} = \frac{q_j}{3} + \frac{1}{6}$. Similar reasoning applies to the remaining cases. For R_j with alternative mappings ($\lambda_\Sigma > \lambda_0$), the additional term $\frac{\lambda_{b_j}}{\lambda_\Sigma}$ represents the possibility of R_j being sampled from the alternative loci. For simplicity, we have assumed the alternative loci have no SNPs and the sampling from them is error free. Therefore, this possibility is $\sum_{k=1}^{K} \frac{\lambda_k}{\lambda_\Sigma} \mathbf{1}\{r_k = b_j\} = \frac{\lambda_{b_j}}{\lambda_\Sigma}$.

Once at locus i, we have obtained estimated $\hat{x}$ by using Equation (1) to (4), we will call a SNP at locus i if $\hat{x} \neq r$.

2.6 Shadow SNP Filtering

For SNPs called at their multiply mapped loci, we have assumed there is one true SNP among them (Section 2.1.1). We call the others as shadow SNPs because they are typically called when the reads sampled from some true SNP locus are multiply mapped onto these loci and thus propagate the false (i.e. shadow) SNP information. This is mainly due to the fact that our SNP caller operates on a locus-by-locus basis, and the SNP calls at the multiply mapped regions are not coordinated. This causes our SNP calls to violate Assumption (*iii*), i.e., there is a single SNP in repetitive regions. In order to compensate for this, we apply a filtering method, which tries to enforce that the called SNPs obey Assumption (*iii*). The basic idea is to keep only the most likely SNP among the SNPs called in the alternate loci.

To formulate, suppose we have a locus i for which we have called a SNP with N_b SNP reads with base value $b \neq r$ mapped at locus i, having abundance λ_0. Let us consider the other loci to which multiply mapped reads also get mapped to, among which loci $\{i_1, ...i_k, ..., i_K\}$ have also been called as SNP, and the abundance at locus i_k be λ_k.

We assume the number of reads with base b actually sampled at locus i is a Poisson random variable X_b with mean $\lambda = \frac{\lambda_i}{2}$. $\frac{\lambda_i}{2}$ is used here because we have assumed each allele has equal contribution to abundance. We now do a hypothesis testing whether SNP reads came from locus i or an alternate locus i_k. Let the confidence of N_b reads sampled at locus i be denoted as $P(X_b = N_b | \frac{\lambda_i}{2})$. Similarly, the confidence of N_b reads actually sampled at alternative locus i_k is $P(X_b = N_b | \frac{\lambda_k}{2})$.

We throw away SNP at locus i if the confidence of N_b SNP reads actually sampled at i is not high enough compared to at its alternative loci:

$$\max_{i_k} P(X_b = N_b | \frac{\lambda_k}{2}) \geq \alpha P(X_b = N_b | \frac{\lambda_i}{2}) \tag{5}$$

Here $\alpha \geq 0$ is a design parameter. When $\alpha = 0$, a SNP detected locus i containing SNP reads alternatively mapped elsewhere will always be filtered away, thus achieving minimal false positive. When $\alpha = 1$, it implies there is some other locus with higher confidence for N_b reads to be sampled from. Therefore we filter the current SNP away.

Empirically we find false positives increase faster than false negatives decrease as $\alpha(> 1)$ gets larger, so we only consider $0 \leq \alpha \leq 1$.

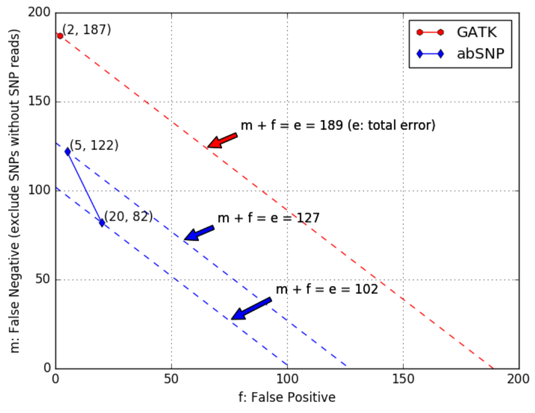
3 Results

In this section, we perform simulation studies to compare the results of abSNP with other alternatives for RNA-Seq SNP calling. It is difficult to obtain real data with ground truth for SNPs that have multiply mapped reads, as existing methods are unable to call these loci reliably. Therefore, we resort to simulation studies in order to evaluate the performance of abSNP and compare it against GATK, which has a best-practice guideline for RNA-Seq SNP calling. We demonstrate that while GATK is unable to make any calls on multiply mapped reads, abSNP can call SNPs with significant accuracy.

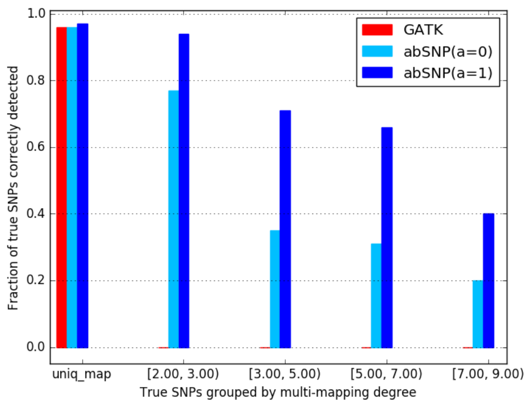
Simulation Setup: To evaluate performance, we have developed a RNA-Seq SNP simulator. The simulator takes as input a reference genome, a transcriptome annotation, the requested number of SNPs, and the read requirements (e.g. number, length, error rate). It assigns each transcript a random expression level according to a log-normal distribution. We explicitly account for the effect of allele-specific expression with maternal and paternal transcripts having different expression levels (in our case, we simulate these expression levels to be independent of each other). The requested number of SNPs are generated randomly in the gene regions where high expression levels (top 10 percent) are assigned, so that the majority of these true SNPs are expected to be covered by SNP reads. We then generate reads independently from the paternal and maternal transcriptomes that contain SNPs using the UC Riverside RNA-seq simulator [18], with error rate set at 1% (to mimic Illumina error rates). Note that due to the randomeness of read sampling, it is possible that some true SNPs are still covered by no or only a few SNP reads. We then pool the reads from the two alleles in order to generate the read dataset.

We generate 5 datasets each with $2M$ 100-bp reads, so that we can get a sense of the average performance. We choose human chromosome 15 of GRCh37 [4] as the reference genome and the relevant UCSC gene annotations [12] as the transcriptome, and generate 2000 SNPs for each dataset. To compare performance, we run both abSNP and GATK by taking simulated reads as input and obtaining SNP candidates as output. For abSNP, the process is described in Section 2.3. For GATK (version 3.4-46), we apply its best practice [2] and incorporate the annotated transcriptome to improve its read alignment.

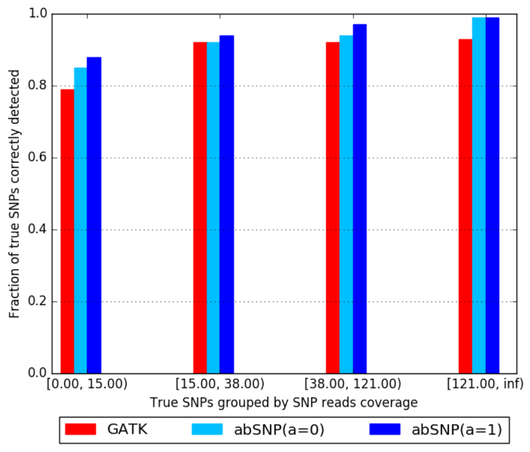
Overall Performance: The SNP calls are compared to the ground truth SNPs in order to estimate the number of ground-truth SNPs missed (false negative) and the number of falsely-called SNPs (false positive). For false negative, we have excluded the SNPs where no SNP reads are sampled because they are trivially to be not detected. The overall performance is plotted in Figure 4a. abSNP has a parameter α that can be tuned in order to change the tradeoff between the false negative and false positive (to make it more conservative or less conservative, as described in Section 2.6), and here we focus on two extreme points $\alpha = 0$ and $\alpha = 1$. We find that abSNP attains much less false negatives with a small increase in false positives. To quantify the effect, we measure the sum of false negative and false



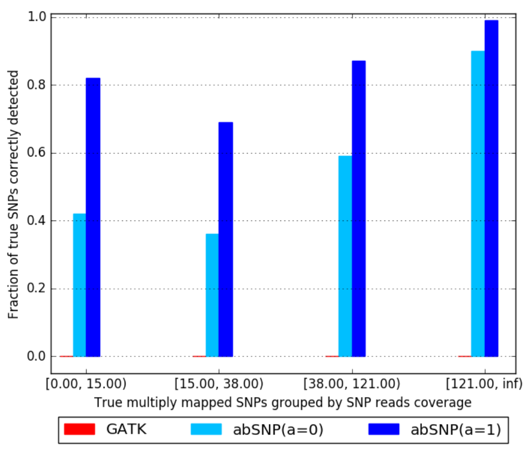
(a)



(b)



(c)



(d)

Figure 4 Performance Evaluation on Simulated Data (1K SNPs per allele, 2M 100-bp reads with error rate 0.01). We categorize true SNPs by their multi-mapping degree (based on GATK's read alignment) in (b) and by their SNP reads coverage in (c) and (d). A SNP is multiply mapped if all its SNP reads are multiply mapped, and its multi-mapping degree is the mean of multiple mappings of its SNP reads. Otherwise it's uniquely mapped with degree 1. SNP reads coverage is the number of SNP reads originally sampled from (instead of mapped onto) the SNPs. **(a)** abSNP has much less false negatives with small increase in false positives; with total error (which is the sum of the false positive and false negative, as demonstrated by 45-degree dashed lines where "m" stands for false negative error (*missed*), "f" stands for false positive and "e" stands for total error) reduced from 189 (GATK) to 102 (abSNP $\alpha = 1$). **(b)** abSNP and GATK share similar sensitivity for uniquely mapped SNPs. For multiply mapped SNPs, GATK fails to make any calls while abSNP is still able to capture these SNPs. **(c)** Both abSNP and GATK increase sensitivity as coverage increases. **(d)** While GATK fails to capture any multiply mapped SNPs across different coverages, abSNP is able to recover these SNPs with high accuracy provided their SNP reads coverages are high.

positive as the total number of errors, and this is plotted by a 45-degree line, from which we can see the gain from abSNP. The total error for abSNP($\alpha = 1$) is 102 whereas GATK makes 189 errors; showing the significant improvement in the error rate. We also point out that abSNP($\alpha = 0$) has only 5 false positives compared to 2 false positives for GATK, while the number of false negatives is reduced from 187 to 122, thus incurring a modest false positive increase can lead to significantly improved sensitivity.

Performance on multiply mapped reads: While the overall results indicate that abSNP can afford performance gain over GATK, the full picture emerges only when we stratify the performance results by the average number of mappings for each read. Consider Figure 4b, where each bar represents an average recovery fraction, and in the x-axis, the true SNPs are grouped based on their multi-mapping degree using GATK's intermediate read alignment. Let the true SNP at locus i (SNP$_i$) has U uniquely mapped SNP reads and V multiply mapped SNP reads (each of which has the number of multiple mappings as $v_1, ..., v_j, ..., v_V$ respectively, with any $v_j > 1$). SNP$_i$ is considered as multiply mapped only when all its SNP reads are multiply mapped (i.e. $U = 0$), and its multi-mapping degree is the mean of multiple mappings of its multiply mapped SNP reads.

For uniquely mapped SNPs (e.g. group uniq_map), abSNP and GATK have very similar sensitivity, with 96% for GATK, 96% for abSNP($\alpha = 0$) and 97% for abSNP($\alpha = 1$). Indeed, the false positives also remain similar between GATK and abSNP($\alpha = 0$). For multiply mapped SNPs (e.g. in groups $[2, 3)$ to $[7, 9)$), GATK fails to detect any SNPs because it will throw away all multiply mapped reads and thus captures no SNP information, while abSNP is still able to call many SNPs succesfully. Indeed, the mild-increase in false positives in abSNP also comes from the the multiply mapped loci. Actually there can be two factors contributing to the gains of abSNP - the first is due to MAPQ filtering, and the second is due to our SNP calling algorithm together with shadow SNP filtering. Both factors are needed in order to obtain the full performance improvement of abSNP, and are only possible due to the exploitation of abundance variation of the different transcripts. In particular, when abSNP becomes conservative (i.g. $\alpha = 0$) on false positives, MAPQ filtering plays a dominant role in our gain. When abSNP becomes less conservative (e.g. $\alpha \to 1$), our calling algorithm together with shadow SNP filtering will dominate the gain especially for SNPs of high multi-mapping degrees.

We choose a very strict definition of multiply mapped SNPs requiring no uniquely mapped SNP reads on that locus (i.e. $U = 0$). This strict choice is motivated from the fact that if there are a non-zero number of uniquely-mapped SNP reads, then existing algorithms can indeed make non-trivial calls. Also, when gene regions are repeated, due to paralogous gene families or long segmental duplications, we expect the SNP to be embedded inside a duplicated region, and hence have no uniquely mapped reads.

Dependence on coverage: We can also stratify the performance by coverage in Figure 4c, where the true SNPs are grouped based on their SNP reads coverage: the number of SNP reads originally sampled from these SNPs. Each group contains 25% of the true SNPs. Each bar represents an average recovery fraction. As SNP reads coverage increases, the sensitivity of all callers improves, with abSNP approaching 100% at the highest coverage group. We note that this is highly significant considering that the highest coverage bar also contains nearly 25% of the multiply mapped SNPs. Thus abSNP has the potential to detect multiply mapped SNPs with high accuracy provided their SNP reads coverage is high. To verify this, we only focus on the true SNPs that are multiply mapped (based on GATK's read alignment) and group them based on their SNP reads coverage as in Figure 4d, where each group also contains approximately 25% of the multiply mapped true SNPs. Whereas GATK does not recover these SNPs, abSNP has a tendency of better recovery as the coverage increases.

4 Discussion

While many algorithms have been developed in order to reveal SNPs in human genome (both coding and non-coding regions) based on different sequencing technologies, SNPs at repetitive genomic regions remain mostly unexplored, because the current SNP discovery mainly relies on methods that ignores all multiply mapped reads due to repetitive genomic regions. We have developed abSNP that is especially designed in order to fill this gap (in particular with regard to the usage of RNA-Seq), through Bayesian principles and filtering methods that utilize the unique products of RNA-Seq abundance estimation that contain rich mapping quality information and estimated abundance. We believe this is the first work to explore this kind of information through an abundance estimation procedure. Our simulated results have shown abSNP's promising performance gain over the widely used GATK best practice. The main gain over GATK is in multiply mapped reads, where GATK does not make any SNP calls, whereas abSNP can get most SNP calls right on the highly abundant gene regions. Our algorithm abSNP is freely available at Github for others to use.

There are many directions for future work: (1) Testing abSNP on real data-sets is an important direction for future work. This is complicated by the lack of ground-truth SNP calls in multiply mapped regions. The present gold-standard datasets focus on SNPs in non-repetitive regions (i.e. they may not belong to the category of multiply mapped SNPs discussed in Section 3), which is the reason for the excellent performance on these datasets. (2) Current version of abSNP does not utilize the pairing information in paired-end reads; this can be potentially utilized to improve performance. (3) abSNP does not factor RNA-editing into account, therefore the SNPs called are post-transcriptional. Thus abSNP in combination with DNA SNP calling can be used to quantify the impact of RNA-editing; although this requires strong statistical controls to reduce the impact of false-positives. (4) Currently abSNP assumes that both alleles have equal expression levels. While we have tested this in the simulation by having differing allele specific expressions, the algorithm can be potentially improved if the effect of allele-specific expression is accounted for. This is a chicken-and-egg problem since SNP calls are needed in order to quantify allele-specific expression, whereas, knowledge of allele-specific expression can improve SNP calls. Thus a joint SNP-calling and allele-specific expression detection can be useful. (5) In many cases, data from both DNA and RNA sequencing are available in order to make SNP calls, sometimes both from regular and diseased tissues. Extending abSNP to this framework is an interesting direction of research. (6) A potential application of abSNP is on real cancer datasets to detect somatic mutations.

Acknowledgements. We thank Ashvin Nair for his help in programming some portions of the algorithm. We also want to thank the anonymous reviewers for their useful comments.

References

1 Abecasis Lab. *GlfMultiples*. `http://genome.sph.umich.edu/wiki/GlfMultiples`.

2 Broad Institute. *GATK Best Practices workflow for SNP and indel calling on RNAseq data*. `https://software.broadinstitute.org/gatk/guide/article?id=3891`.

3 Elizabeth T. Cirulli, Abanish Singh, Kevin V. Shianna, Dongliang Ge, Jason P. Smith, Jessica M. Maia, Erin L. Heinzen, James J. Goedert, and David B. Goldstein. Screening the human exome: a comparison of whole genome and whole transcriptome sequencing. *Genome Biology*, 11(5):R57, 2010. `doi:10.1186/gb-2010-11-5-r57`.

4 The Genome Reference Consortium. *Human Genome Assembly GRCh37*. `https://www.ncbi.nlm.nih.gov/grc/human/data?asm=GRCh37`.

5 Kristal Curtis, Ameet Talwalkar, Matei Zaharia, Armando Fox, and David A. Patterson. *SiRen: Leveraging Similar Regions for Efficient and Accurate Variant Calling*, 2015. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-159.html`.

6 M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. Angel, M. A. Rivas, M. Hann, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. Gabriel, D. Altshuler, and M. J. Daly. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, 43(5):491–498, April 2011. doi:10.1038/ng.806.

7 A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 2012. doi:10.1093/bioinformatics/bts635.

8 Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing, 2012. arXiv:arXiv:1207.3907.

9 Maryam Ghareghani, Seyed Abolfazl Motahari, Shahram Khazaei, and Mostafa Tavassolipour. Gw-call: Accurate genome-wide variant caller. *bioRxiv*, 2016. doi:10.1101/079905.

10 R. Goya, M. G. F. Sun, R. D. Morin, G. Leung, G. Ha, K. C. Wiegand, J. Senz, A. Crisan, M. A. Marra, M. Hirst, D. Huntsman, K. P. Murphy, S. Aparicio, and S. P. Shah. SNVMix: predicting single nucleotide variants from next-generation sequencing of tumors. *Bioinformatics*, 26(6):730–736, February 2010. doi:10.1093/bioinformatics/btq040.

11 The SAM/BAM Format Specification Working Group. *Sequence Alignment/Map Format Specifiation*. https://samtools.github.io/hts-specs/SAMv1.pdf.

12 UCSC Genome Informatics Group. *UCSC Genome Browser*. https://genome.ucsc.edu/cgi-bin/hgTables.

13 Daehwan Kim, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L. Salzberg. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14(4):R36, 2013. doi:10.1186/gb-2013-14-4-r36.

14 D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson. VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Research*, 22(3):568–576, 2012. doi:10.1101/gr.129684.111.

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

16 H. Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, September 2011. doi:10.1093/bioinformatics/btr509.

17 H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858, November 2008. doi:10.1101/gr.078212.108.

18 Wei Li. *RNASeqReadSimulator: A Simple RNA-Seq Read Simulator*. http://alumni.cs.ucr.edu/~liw/rnaseqreadsimulator.html.

19 Robert Piskol, Gokul Ramaswami, and Jin Billy Li. Reliable identification of genomic variants from RNA-seq data. *The American Journal of Human Genetics*, 93(4):641–651, 2013. doi:10.1016/j.ajhg.2013.08.008.

20 Adam Roberts and Lior Pachter. Streaming fragment assignment for real-time analysis of sequencing experiments. *Nat Meth*, 10(1):71–73, January 2013. Brief Communication. doi:10.1038/nmeth.2251.

21 Daniel F. Simola and Junhyong Kim. Sniper: improved SNP discovery by multiply mapping deep sequenced reads. *Genome Biology*, 12(6):R55, 2011. doi:10.1186/gb-2011-12-6-r55.

22 X. Tang, S. Baheti, K. Shameer, K. J. Thompson, Q. Wills, N. Niu, I. N. Holcomb, S. C. Boutet, R. Ramakrishnan, J. M. Kachergus, J.-P. A. Kocher, R. M. Weinshilboum, L. Wang,

E. A. Thompson, and K. R. Kalari. The eSNV-detect: a computational system to identify expressed single nucleotide variants from transcriptome sequencing data. *Nucleic Acids Research*, 42(22):e172–e172, October 2014. `doi:10.1093/nar/gku1005`.

23 Justin M. Zook, Brad Chapman, Jason Wang, David Mittelman, Oliver Hofmann, Winston Hide, and Marc Salit. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nat Biotech*, 32(3):246–251, Mar 2014. Computational Biology. `doi:10.1038/nbt.2835`.

All Fingers Are Not the Same: Handling Variable-Length Sequences in a Discriminative Setting Using Conformal Multi-Instance Kernels

Sarvesh Nikumbh[1], **Peter Ebert**[1], **and Nico Pfeifer**[3]

1 **Department of Computational Biology and Applied Algorithmics, Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
`snikumbh@mpi-inf.mpg.de`

2 **Department of Computational Biology and Applied Algorithmics, Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
`pebert@mpi-inf.mpg.de`

3 **Department of Computational Biology and Applied Algorithmics, Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany; and**
Department of Computer Science, University of Tübingen, Tübingen, Germany
`npfeifer@mpi-inf.mpg.de`, `pfeifer@informatik.uni-tuebingen.de`

Abstract

Most string kernels for comparison of genomic sequences are generally tied to using (absolute) positional information of the features in the individual sequences. This poses limitations when comparing variable-length sequences using such string kernels. For example, profiling chromatin interactions by 3C-based experiments results in variable-length genomic sequences (restriction fragments). Here, exact position-wise occurrence of signals in sequences may not be as important as in the scenario of analysis of the promoter sequences, that typically have a transcription start site as reference. Existing position-aware string kernels have been shown to be useful for the latter scenario.

In this work, we propose a novel approach for sequence comparison that enables larger positional freedom than most of the existing approaches, can identify a possibly dispersed set of features in comparing variable-length sequences, and can handle both the aforementioned scenarios. Our approach, *CoMIK*, identifies not just the features useful towards classification but also their locations in the variable-length sequences, as evidenced by the results of three binary classification experiments, aided by recently introduced visualization techniques. Furthermore, we show that we are able to efficiently retrieve and interpret the weight vector for the complex setting of multiple multi-instance kernels.

1998 ACM Subject Classification I.5.1 Models, I.5.2 Design Methodology, J.3 Life and Medical Sciences

Keywords and phrases Multiple instance learning, conformal MI kernels, 5C, Hi-C

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.16

1 Introduction

In various studies since the elucidation of the human genome, many different definitions of promoters have been used in different studies. For example, Butler et al. defined a core promoter as a minimal stretch of contiguous DNA sequence ($\sim$ 40 nucleotides (nts)) that contains the transcription start site (TSS) and is sufficient for accurate transcription initiation [4], and a proximal promoter as a region in the immediate vicinity of the TSS, roughly 250 nts upstream and downstream. There are examples of many studies that consider either only an upstream region or using an arbitrary-sized window around the TSS (albeit fixed for the study) as promoter sequences. How does one know what size is appropriate in any independent new study or a study unifying such promoter sequences from multiple prior studies?

Discriminative machine learning methods like support vector machines (SVMs) [3] with their state-of-the-art performance on many relevant problems in computational biology (e.g., splice site prediction [15]) have been proven to be a very powerful tool. The earliest kernel-based approaches for computing similarities between biological sequences, e.g. spectrum [9] and mismatch kernel [10], allowed comparing sequences of different length, but they did not encode any positional information. Latter approaches, for example the weighted degree kernel [16] and oligo kernel [13], do consider positional information in the corresponding sequences, even with a certain amount of positional uncertainty [15]. Additionally, alignment-based sequence comparison also provides a position-dependent similarity score albeit with a gap penalty [17]. Thus, these approaches do allow deviations from exact matches but they are penalized. The oligomer distance histograms (ODH) kernel [11] allows comparing of sequences of different length by way of representing a sequence with a fixed-length feature vector. But it ignores information about the position of such oligomer pairs within the sequence.

These scenarios are outlined in Figure 1, panel Motivation, when comparing two sequences S1 and S2. Any position-aware kernel that also allows shifts can detect the signal in case (a) but not in case (b) where the signal is very far apart. Even if it does, it would penalize this deviation. Case (c) represents how ODH would detect this signal and thus consider the two sequences to be similar, but information on the position of this signal in the individual sequences is lost. This work is a step in the direction to tackle this issue: compare sequences allowing reasonable degree of positional freedom and not simultaneously penalizing this deviation or keeping it problem-dependent. This scenario can arise in case of Hi-C data where the pairs of loci interacting over a long-range are variable-length restriction fragments reported from the experiments and the causal signal in the two loci compared, for example, the enhancer and the promoter, does not have any positional restriction unlike the transcription start site in the promoter sequences.

In this work we approach the problem of handling variable-length sequences and allowing positional freedom when comparing them for problems such as identifying promoter architectures or analysis of long-range interaction partners collected from Hi-C experiments. We do so by breaking the individual sequences into segments (see Figure 1, panel Motivation, case (d)) and casting the problem into a multiple instance learning problem [6] where the instances in each bag are *parts* of the *whole* sequence. We employ conformal multi-instance kernels [2] to obtain the weightings for instances in each bag, thus rendering the capability to identify segments of a sequence informative for the prediction problem. We efficiently retrieve the weight vector for the complex setting of multiple conformed multi-instance kernels (outlined in Section 2.4.1). We also demonstrate how to interpret the nonlinear classifiers by adopting visualization techniques that were recently introduced [14] in the more basic setting (outlined in Section 2.4.2).

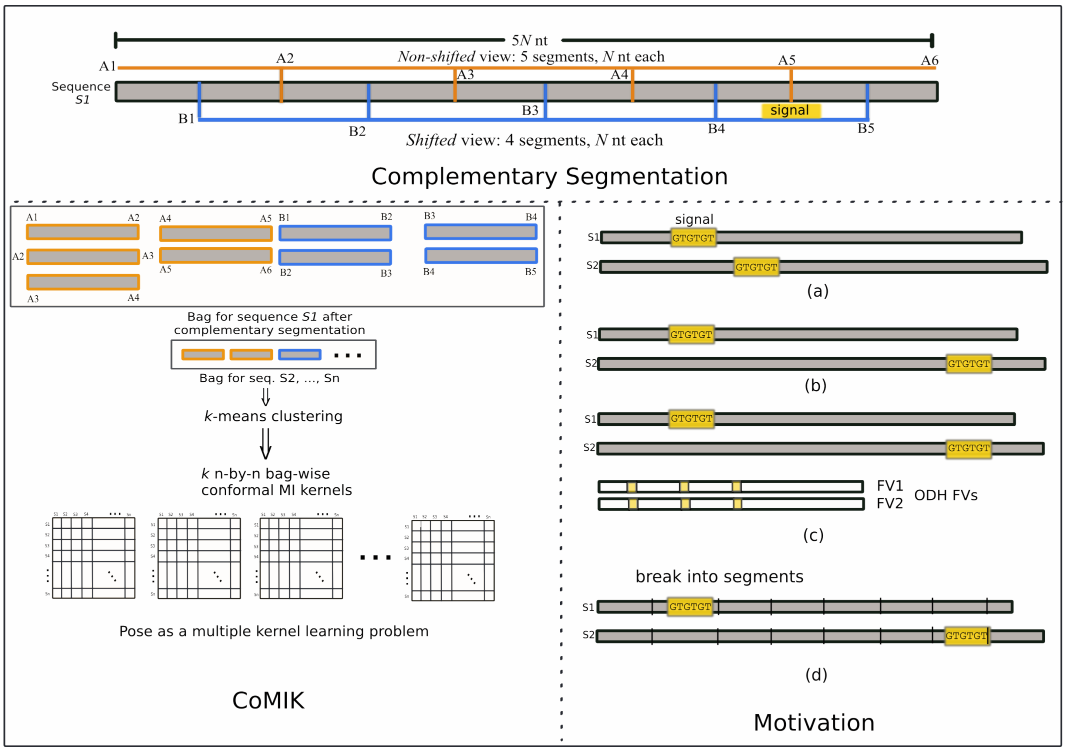

5N nt
Non-shifted view: 5 segments, N nt each
A1
A2
A3
A4
A5
A6
Sequence S1
B1
B2
B3
B4
B5
signal
Shifted view: 4 segments, N nt each
Complementary Segmentation
Bag for sequence S1 after complementary segmentation
Bag for seq. S2, ..., Sn
k-means clustering
k n-by-n bag-wise conformal MI kernels
Pose as a multiple kernel learning problem
CoMIK
GTGTGT
(a)
(b)
FV1
FV2
ODH FVs
(c)
break into segments
(d)
Motivation

Figure 1 A schematic of our approach *CoMIK*, and the motivation. The complementary segmentation procedure is illustrated in the top panel. The sequence is shown in gray. *Non-shifted* segments are shown with an orange border, and *shifted* segments with blue border. The 'Motivation' panel shows the various scenarios when comparing sequences S1 and S2. Here, a signal in the sequence is shown as a yellow box. 'FV1' and 'FV2' denote the ODH (oligomer distance histograms [11]) feature vectors for S1 and S2 respectively. The panel 'CoMIK' shows a schematic of our approach starting from a sequence to the complementary conformal multi-instance (MI) kernel (see Methods for details). S1, S2,..., Sn are the n sequences in the collection.

2 Methods

Towards pair-wise comparison of variable-length sequences allowing positional freedom, we segment the individual sequences thus representing each sequence as a collection of its segments and then compare all segments of one sequence to all those of the other. With this, the typical binary classification problem involving sequences is cast into a multiple instance learning problem [6]. We call our approach *CoMIK* for 'Conformal Multi-Instance Kernels'.

In the following, we begin by discussing our sequence segmentation procedure and the need for a complementary representation (Section 2.1). Further, we show how we exploit this design with the help of conformal transformations to the multi-instance (MI) kernel [2] to identify segments of a sequence which are important (due to features contained in the segment) towards its classification (Section 2.2). Subsequently, we discuss the procedure to obtain the SVM weight vector from the multiple conformal multi-instance kernels and visualizing the important features thus making it interpretable (Section 2.4).

2.1 Segment Instantiation with Complementary Views

Non-shifted Segment Instantiation: Given any arbitrary length sequence, we propose representing it by its segments where a segment is defined as a smaller part of the whole sequence. Beginning right at the start of the sequence, we create segments of a predetermined size along the sequence until it ends. The segment size is chosen *a priori* by the user depending on the problem (see Section 2.3 for details). The last segment is allowed to have a different size (either larger or smaller than other segments) to accommodate any remainder portion in case the sequence length is not an exact multiple of the segment size. We call this instantiation the *non-shifted* segment instantiation. A simple case of *non-shifted* segmentation is illustrated in Figure 1 (panel 'Complementary Segmentation'). This segmentation provides the non-shifted view of the whole sequence as the first segment starts at the beginning of the sequence and, in total, the segments span the entire sequence.

Shifted Segment Instantiation: There may still be signals at the boundaries of any two non-shifted segments (see Figure 1, panel 'Complementary Segmentation', signal at position A5) which may get overlooked when comparing sequences using just *non-shifted* segments. To cover for this scenario, we introduce an alternate instantiation called *shifted* segmentation whereby the boundaries due to initial segmentation of the sequence end up in the same segment in this representation. In this case, segmentation begins from the mid-point of the first non-shifted segment, and proceeds to create further segments along the sequence essentially covering the boundaries of the non-shifted segments. The portions of the sequence before B1 and beyond B5 can be omitted since they are already covered in the *non-shifted* view (see Figure 1). *Shifted* segments can either be of same size as the *non-shifted* segments or different. Thus, *shifted* segmentation provides a complementary view of the same sequence covering the portions which get overlooked by *non-shifted* segmentation.

Refer to Section 2.3 for a discussion on choosing an appropriate segment-size and its influence on the method.

2.2 Conformal Multi-Instance Kernels for Complimentary Set of Segments

Once segmented, we cast this problem into a multiple instance learning problem [6]. In this setting, each sample (X, y) contains a set of instances $x \in X$ and label y. The sets of instances are also called bags. Each sequence is thus treated as a bag and its segments as instances in the bag. One or more instances from a bag could be responsible for the bag to be classified as positive or negative due to the presence or absence of class-specific features. Since there is no restriction on the number of instances a bag can contain, this setting can inherently allow for considering arbitrary length sequences that result in an arbitrary number of instances per bag upon segmentation. Thus, there is a bag for each sequence containing *non-shifted* and *shifted* segments of the sequence as instances in its bag.

2.2.1 Multi-Instance Kernels

Gärtner et al. proposed the normalized set kernel (also known as the multi-instance kernel) for the multiple instance problem [8]. For each sample represented as a bag of instances, the kernel value between any two bags X and X', $k(X, X')$, is given as in Eq. 1.

$$k(X, X') := \frac{k_{\text{set}}(X, X')}{f_{\text{norm}}(X) f_{\text{norm}}(X')} . \tag{1}$$

Here $k_{\text{set}}(X, X') := \sum_{x \in X, x' \in X'} k(x, x')$ and $f_{\text{norm}}(X)$ is a suitable normalization function. One could normalize using either averaging ($f_{\text{norm}}(X) := \#X$, where $\#X$ denotes the number of instances in bag X) or feature space normalization ($f_{\text{norm}}(X) := \sqrt{k_{\text{set}}(X,X)}$). In this work, we used feature space normalization.

While the multi-instance kernel can successfully handle comparison between bags by comparing their individual instances, it has the issue that, in averaging, it looses any information related to the contributions of the individual instances. In other words, it treats all the instances in a bag equally. And, it is usually desirable to not only obtain a solution to a problem, but also to identify (a) the features that contribute to that specific solution, and (b) the parts which contain these features. Here, (b) amounts to knowing which instance(s) in a bag have features that helped determining the correct class label of the bag (positive or negative class). To this end, we propose using conformal multi-instance kernels [2] that allow us to obtain an instance weighting based on the contribution of these instances to learning the discriminant function.

2.2.2 Conformal Multi-Instance Kernels

Blaschko and Hofmann proposed the conformal multi-instance kernel as a modification to the normalized set kernel [2]. This modification is a conformal transformation parameterized by θ, $t_\theta > 0$, applied to the kernel function, meaning that the transformation preserves the angle between vectors in the mapped space. The idea is to magnify those regions in the feature space which are discriminative and shrinking those which are not discriminative. Selection of these candidate regions in the feature space is done by clustering the complete set of input instances and choosing the corresponding cluster centres as candidate regions or expansion points. The decision of whether the region characterized by any cluster centre is discriminative or not is made by solving the multiple kernel learning problem as explained further.

Blaschko and Hofmann [2] proposed (a) the conformal transformation $t_\theta(x)$ to be of the form given in Eq. 2.

$$t_\theta(x) = \sum_{e=1}^{E} \theta_e \tilde{\kappa}(x, c_e) \tag{2}$$

$$\tilde{\kappa}(x, c_e) = \exp\left(- \frac{||x - c_e||^2}{2\sigma^2} \right) \tag{3}$$

Here, c_e's denote the cluster centres indexed by $e \in \{1, \dots, E\}$ for a total of E expansion points; and (b) $\tilde{\kappa}$ to be a Gaussian (Eq. 3) whose bandwidth (σ) can be adjusted. The parameter θ_e in Eq. 2 tells how discriminative the region around a certain cluster centre is. A large value of θ_e denotes that the neighborhood of the corresponding expansion point c_e is a discriminative region. As mentioned, the θ_e values are learnt via multiple kernel learning (see subsection 'Resultant conformal multi-instance kernel' and Eq. 5). Thus, replacing $k(x, x')$ by its conformal transformation $t_\theta(x)t_\theta(x')k(x, x')$

$$k(X, X') = \frac{1}{f_{\text{norm}}(X) \cdot f_{\text{norm}}(X')} \sum_{x \in X} \sum_{x' \in X'} t_\theta(x) t_\theta(x') \underbrace{k(x, x')}_{\text{base kernel}} \tag{4}$$

Identifying expansion points. To identify a set of expansion points, one could use k-means clustering with all available instances to identify clusters, whose cluster centres, c_e's, are then treated as expansion points ($E = k$). Here, the individual instances are represented by their ODH feature vectors as discussed in Section 2.2.3. When dealing with too many instances, which could make the clustering process a bottleneck, Blaschko and Hofmann [2] suggest using the buckshot clustering approach [5] wherein, in order to identify E clusters from n instances, instead of using all n instances, one could perform k-means clustering using randomly sampled $\sqrt{En}$ instances out of n. This has been shown to identify qualitatively similar clusters and being highly scalable at the same time [2].

Resultant conformal multi-instance kernel. Upon substituting Eq. 2 in Eq. 4, and simplification (see [2] for more details), the conformal multi-instance kernel is given by

$$k(X, X') \approx \sum_{e=1}^{E} \theta_e^2 \left(\frac{1}{f_{\text{norm}}(X) \cdot f_{\text{norm}}(X')} \sum_{x \in X} \sum_{x' \in X'} \tilde{\kappa}(x, c_e) \tilde{\kappa}(x', c_e) \underbrace{k(x, x')}_{\text{base kernel}} \right) \tag{5}$$

Eq. 5 is then posed as a multiple kernel learning (MKL) [1] problem (linear in $\rho_e \equiv \theta_e^2$) to simultaneously learn the θ_e's and the SVM parameters α, also called λ in part of the literature.

Obtaining individual instance weights. Upon solving the MKL problem, once the sub-kernel weights (θ_e's) are obtained we can directly obtain $t_\theta(x)$ for any instance x of a bag X using Eq. 2.

2.2.3 Oligomer Distance Histograms (ODH) Kernel as Base Kernel

The choice of the base kernel to compare the individual instances depends on the problem. Here, we propose representing the individual segments of any sequence by its ODH representation [11] and using the ODH kernel [11] to compute similarities between them. In the ODH representation, any arbitrary-length sequence is represented by a feature vector that counts the occurrences of all pairs of short K-mers separated by d positions in the sequence.

For the DNA alphabet, $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, with $m_i \in \Sigma^K, i = 1, \ldots, M$ as all possible K-mers, any L-length sequence s can have K-mers separated by a maximum distance $D = L - K$. Thus, $d \in \{0, \ldots, D\}$. Here, distance between a K-mer pair is the difference between its starting positions in the sequence. For any K-mer pair (m_i, m_j), the distance histogram vector of its occurrences in sequence s is given as $\mathbf{h}_{ij}(\mathrm{s}) = [h_{ij}^0(\mathrm{s}), h_{ij}^1(\mathrm{s}), \ldots, h_{ij}^D(\mathrm{s})]^{\mathrm{T}}$. Here, each $h_{ij}^d(\mathrm{s})$ is the count of the number of times the K-mer pair (m_i, m_j) is observed at distance d in s. Finally, the feature space transformation of sequence s is obtained by stacking together the distance histograms of all K-mer pairs over Σ.

$$\Phi(\mathrm{s}) = [\mathbf{h}_{11}^{\mathrm{T}}(\mathrm{s}), \mathbf{h}_{12}^{\mathrm{T}}(\mathrm{s}), \ldots, \mathbf{h}_{MM}^{\mathrm{T}}(\mathrm{s})]^{\mathrm{T}} \tag{6}$$

Then, N training samples are given as: $\mathbf{X} = [\Phi(\mathrm{s}_1), \ldots, \Phi(\mathrm{s}_N)]$ and the $N \times N$ kernel matrix is given by $\mathbf{K} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$.

2.3 Choosing an appropriate segment-size

While the user could choose a segment-size that is appropriate for a problem, there is a trade-off one should consider. On the one hand, the ODH kernel computation involving dot products between very high-dimensional feature vectors benefits from the sparsity of these feature vectors. But, with just 4 characters in the DNA alphabet, for very long segments the representation may not be sparse enough. On the other hand, having too many segments overall, influences the computation time spent performing clustering and subsequently applying the transformation per segment. Thus, there is a trade-off between the size of the segments and the total number of segments at the training stage.

In general, many long segments in total from all the sequences at the training stage could lead to a longer computation time for the (instance-wise) base kernel, but we note that this is done only once at the beginning.

2.4 Interpretation and Visualization of Features

In the following, we discuss how one can interpret and visualize the sequence features deemed important by *CoMIK* for a prediction problem.

2.4.1 Obtaining the SVM Weight Vector for *CoMIK*

In the MKL problem [1], the weight vector corresponding to a given sub-kernel K_m is given as in Eq. 7.

$$\mathrm{w}_m = \beta_m \sum_{i=1}^{N} \alpha_i y_i \Phi_m(X_i) \tag{7}$$

$$\Phi_m^{c_e}(X) = \frac{1}{B} \sum_{x \in X} \tilde{\kappa}(x, c_e)\phi_m(x) \tag{8}$$

Here β_m is the sub-kernel weight learnt by solving the MKL problem and each $\Phi_m(X_i)$ is the feature space representation of sequence X_i corresponding to sub-kernel K_m. And, for the conformally transformed multi-instance setting, this would mean $\Phi_m(X)$ is the bag-level, transformed ODH representation of the sequence corresponding to the cluster centre chosen when computing the sub-kernel K_m. Thus, $\Phi_m^{c_e}(X)$ can be represented mathematically as in Eq. 8 where $\phi_m(x)$ is the ODH representation of segment x (Eq. 6) belonging to bag X, $\tilde{\kappa}(x, c_e)$ is the Gaussian transformation (Eq. 3) and B is the feature space normalization factor. Following [20], B can either be $\sqrt{k(X,X)}$ or $||\sum_{x\in X} \tilde{\kappa}(x, c_e)\phi_m(x)||_2$ since our base kernel, the ODH kernel, is a dot product kernel (refer to Section 2.2.3). Thus, we have a bag-level representation of a sequence corresponding to all cluster centres which allows us to compute all the relevant weight vectors. These individual weight vectors can also be used to make fast predictions on test examples. For this, we only need the transformed ODH representations of the test examples corresponding to each kernel in the collection.

2.4.2 Visualizing Features from the *CoMIK* Weight Vector

Figure 2 shows two ways of visualizing the features deemed important by *CoMIK* in discerning the positive set of sequences from the negative set. The bottom-left panel in Figure 2 shows the distance-centric view, the 'Absolute Max Per Distance' (AMPD) visualization [14], and, the right panel, K-mer-centric view [11]. In the AMPD visualization, based on the ODH feature representation, each dimension of the weight vector corresponds to the histogram count of an oligomer pair lying at a given distance d (unit: basepairs) (refer Section 2.2.3, Eq. 6). From this weight vector, for each distance value, among all the K-mer pairs, we pick the pair that is assigned the most positive and most negative coefficient. A positive coefficient value means the feature (i.e., the d-separated K-mer pair, $d \in \{0, 1, \ldots, D\}$) is prominent among the positive set of examples, otherwise negative. This provides a distance-centric view of the important features. The K-mer-centric view [11] shows the role of each K-mer

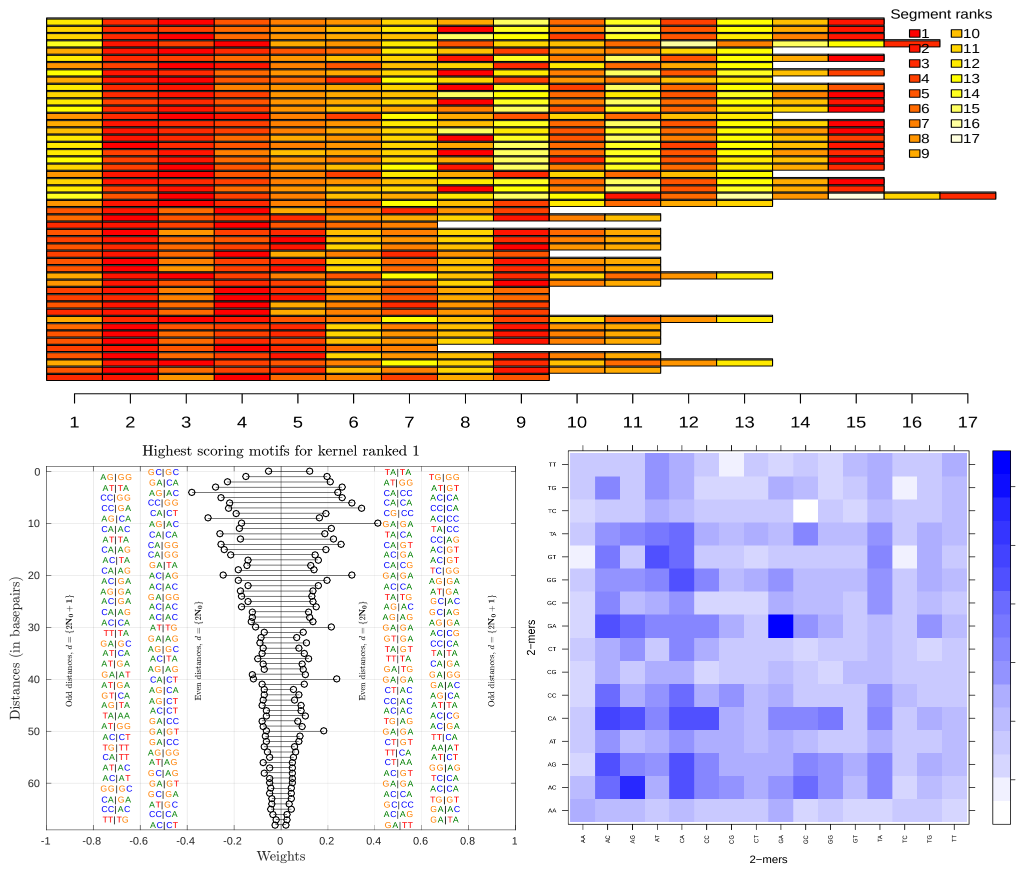


Figure 2 (top) Visualizing the weights assigned to segments of the variable-length sequences in simulated data set, distance-centric (bottom-left panel) and K-mer-centric visualizations (bottom-right panel) of features for the simulated data set. The left panel shows 2-mer pairs that were assigned the highest positive and negative weights at each distance value corresponding to a sub-kernel that was assigned the highest weight. For easy viewing, the K-mer-pairs at odd distances are placed on the outside and the even distances, inside. Horizontal axis: weights, vertical axis: distances between 2-mer pairs (in basepairs). Refer to Section 2.4.2 for details on the K-mer centric visualization.

pair towards prediction. Simply stated, the K-mer-centric view of the discriminant is a matrix which is obtained upon performing, for all K-mer pairs, an ℓ_2-norm of the relevant dimensions of the weight vector, corresponding to all distance values considered, with itself. Thus, a pair which holds high importance (i.e., it has large coefficients in the discriminant, positive or negative) will have higher absolute value in the matrix.

3 Materials

Simulated data set: We prepared a simulated data set of 1000 arbitrary-length sequences with a mix of many coupled and non-coupled motifs as explained below. Of these 1000 were three kinds of positive sequences totaling 500; the rest 500 comprised of two kinds of negative sequences.

Refer to Table 1 for the following: (a) 300 of the 500 positive sequences had motifs from set A planted in them (column marked '+'), all except those marked with `N` (e.g., `4N` and `5N` which are negative variants of the positive motifs `4P` and `5P`, respectively). (b) Another 100

Table 1 Motif sets planted in the simulated data set. The differences between the positive and the negative variants are underlined (e.g., `4P` and `4N`). '`-`' denotes a gap. Columns '+' and '−' give the number of positives and negatives respectively containing the corresponding set of motifs. Columns 'P' and 'N' give the #segment (non-shifted) in which the motif could lay (start positions).

Set	Motifs	+	−	P	N
A	`1.'GAGTTATACATGGTATAGACCACACTATTA'`	300	300	{1,2}	{2,3}
	`2.'AACATGGTCTAGACCATTTT'`			{3}	{1}
	`3.'CTAAACAGGGTCTATACCACACTATTA'`			{5}	{5}
	`4P.'AGGATATATATGTGCTCTTCAGATTTTCACCCTTAGCAAGAGCGAGG'`			{6}	–
	`4N.'ACCATATACATGTGCAGATCAGATTTTCACCCCGAGCAAGAGCGAGG'`			–	{6}
	`5P.'ACACAGCTACTACCACAGGGACAGACAGACAG'`			{4}	–
	`5N.'ATAGCGCTACTACCACACCCACAGACAGACAG'`			–	{1}
B	`1.'ACCATATACATGTGCAGATCAGATTTTCACCCCGAGCAAGAGCGAGG'`	100	200	{3}	{2,3}
	`2.'ATAGCGCTACTACCACACCCACAGACAGACAG'`			{2}	{1}
	`3P.'GACACATGTGCACATATGGTTTTCACCCCGATACATAGTGAGG'`			{4}	–
	`3N.'GACACATGTGCACATATG-TAGCGAGG'`			–	{3,4}
C	`'GA' repeated at every 10 nt in the sequence`	100	–	–	–

positive sequences had motifs from set B planted in them – `3P` and `3N` denoting variants as in (a). (c) Additional 100 positive sequences had the dinucleotide `'GA'` repeated at every 10nt throughout the sequences. For the 500 negative sequences, 300 contained all motifs from set A (1, 2, 3 and the negative variants) and the remaining 200, similarly, with motifs from set B. In all the sequences, each motif was planted at a randomly chosen start position inside a respective window. For *CoMIK*, it was then possible to determine the segment in which the different motifs could lay. Since we later discuss results with segment-size 70nt, columns 'P' and 'N' already give the 70nt-segment numbers (for non-shifted segments) for each motif. Length of sequences of type (a) and (b), either positive or negative, was in the range [300,500]nt, and [500,600]nt for type (c). All sequences were generated with uniform probabilities for `A`, `C`, `G` and `T` and the motifs had a 0.1 mutation probability. Maintaining equal proportions of the different kinds of positives and negatives, we held out 200 sequences as unseen test examples (100 positives and 100 negatives) and used the remaining 800 sequences for training.

Yeast: Lubliner et al. studied yeast core promoter sequences analyzing the effect of sequence variation in different core promoter regions [12]. Among other things, the authors showed that location, orientation, and flanking bases are important for TATA element function. We obtained a total of 316 118nt-long core promoter sequences ([-118,-1] relative to the TSS) for which the core promoter activity measurements were provided and followed the procedure in Figure 5 in [12] to classify them into two classes– sequences showing either low or high activity (low or high expression), giving 28 positive and 288 negative sequences.

5C: In a recent study, Nikumbh and Pfeifer [14] approached the problem of predicting the long-range interaction partners of a genomic locus (of interest) profiled in 5C experiments in cell lines GM12878, K562 and HeLa-S3 [18] using the DNA sequence at the interacting (positive class) and non-interacting loci (negative class) w.r.t. the locus of interest. For any 5C-profiled TSS-containing region, the distal loci that showed a significant interaction with it in all replicates were considered as positive and the ones that did not interact significantly

Table 2 Number of positive and negative sequences the 5C data set.

	GM12878	K562	HeLa-S3
#Positives	63	46	98
#Negatives	226	105	207

Table 3 Parameters and the range of values tested for the simulated, 5C and the yeast data set.

Parameters	Simulated data set	5C	Yeast
#Clusters	{2, 5, 7}	{5, 7, 10}	{2, 5, 7}
Segment size	{50, 70}	50	10
Sigma (σ) for Gaussian transformation	$10^{\{-1,\ldots,2\}}$	$10^{\{2,4,6\}}$	$10^{\{-1,\ldots,2\}}$
Oligomer length	{2, 3}	{2, 3}	{2, 3}
Maximum distance	{50, 70}	50	10
SVM-cost	$10^{\{-3,\ldots,3\}}$	$10^{\{-3,\ldots,6\}}$	$10^{\{-3,\ldots,3\}}$

in any replicate were considered negative. Thus, here the problem is approached as a single binary classification task. While the authors also consider a multitask setting [14], here, for the performance comparison, we performed experiments in the single task setting (see [14] for more details). We fetched the positive and negative set of sequences for one region (*region* 0) per cell line [14]. The number of positive and negative sequences for each of these are given in Table 2.

4 Experimental Setup

For each data set, we performed 5-fold nested cross-validation (CV) by splitting the data into 80%:20% for training and test, respectively. For each outer-fold, model selection was performed with a 5-fold inner CV loop on the training set with ℓ_1- and ℓ_2-norm MKL. We note that *CoMIK* accounts for any class imbalance by proportionately up-weighting the misclassification cost for the minority class as proposed in [7]. All parameters and the range of values tested for them are given in Table 3. Of these, #Clusters, σ and SVM-cost are optimized by cross-validation while other parameters, namely segment size, oligomer length and maximum distance, are assigned fixed values for each individual run. We used the same segment-size for the *shifted* and the *non-shifted* cases. The best performing set of parameter values obtained from the inner CV-folds was used to re-train the model using the complete training data and make predictions on the unseen test set of examples per outer CV-fold. We report the area under the receiver operating characteristic (ROC) curve (AUC) for predictions on this held-out test set averaged over the five outer folds.

We compare our performance on the simulated data set to that of KIRMES (Kernel-based Identification of Regulatory Modules in Euchromatic Sequences) [19]. KIRMES was shown to perform well on gene sets derived from microarray experiments for identifying loss or gain of gene function [19]. For a collection of positive and negative genomic sequences, given a set of motifs representing transcription factor binding sites and their match-positions (obtained by performing a motif-finding step *a priori*) in the sequences, KIRMES picks a *fixed-size* window around the best match-position of the motif in the sequence as a representative of the sequence for that motif. These selected, *fixed-size* portions from all sequences (thus, equal length) are used to compute a WDSC kernel (weighted degree kernel with shifts and conservation information; see [19] for complete details) corresponding to that motif. This procedure results in as many kernels as the number of motifs. The remaining parts of the sequences (those not selected for any motif) are *neglected*.

5 Results

In the following, with computational experiments on a simulated data set, and yeast and 5C data, we demonstrate how *CoMIK* can be used to uncover the features regarded important for classification together with their locations (at the segment level) in any candidate sequence.

Simulated data set: For this data set, while KIRMES achieves an AUC of 0.9432, *CoMIK* attains near-perfect classification, AUC 0.9960 ± 0.003. We surmise that the superior performance of *CoMIK* is due to the sequences containing the dinucleotide repeat motif '`GA`' (see Figure 1) which may not be captured at the motif-finding step and thus affect KIRMES' prediction.

We provide visualizations from the run that achieved the best performance with oligomer-length 3, segment-size 70nt, ℓ_1-norm MKL in Figure 2. The top panel visualizes the 70nt-long segments of 50 out of the 200 test sequences horizontally. For each sequence, the *non-shifted* segments are followed by its shifted segments. Per sequence, the higher-ranked segments would be the ones where the features are located. Figure 2, bottom-left panel, is a distance-centric visualization of the SVM weight vector and the bottom-right panel, the K-mer-centric view. While the K-mer-centric view clearly indicates `GA`'s important role, the distance-centric visualization shows that it could be periodic. Experiments using different segment-sizes can easily uncover the fact that they are spread throughout the sequences.

Yeast: *CoMIK* achieved an AUC of 0.9459 ± 0.029 on this data set with segment-size 10nt, oligomer-length 3 and ℓ_1-norm MKL. Furthermore, the most important features represent motifs known as important for classification. We visualize the 3-mer pairs deemed important by *CoMIK* for this classification in Figure 3, left panel. The right panel here visualizes the sequences and their ranked segments as a heatmap. The 316 sequences are arranged vertically from top to bottom, and their segments horizontally. For the 118nt-long sequences in this data set, the segment-size of 10nt lead to 12 *non-shifted* and 11 *shifted* segments, and are arranged in that order. Thus, the coordinates for the *non-shifted* and *shifted* segments in the sequence are as marked on the top of the heatmap.

We observe that segments 3 and 9, i.e., regions $[-98,-89]$ and $[-38,-29]$ happen to be ranked first consistently. Segments 15 and 21 are the best-ranked shifted segments also corresponding to the same genomic window. And, indeed, Lubliner et al. report that the main TSS lay at position -30 and that the regions $[-118,-99]$ and $[-98,-69]$ hold important features which upon mutations greatly reduced expression [12]. In the left panel, the top-ranked kernel shows TATA-like elements to be important for classification. Furthermore, among the features reported by other kernels in the collection (not shown), *CoMIK* rightly identifies `T/C`-rich K-mers to be enriched among the positive sequences as against `G/C`-rich K-mers which are also reported in Supplementary Figure 4 in [12].

5C data set: Performances of *CoMIK* on the three cell lines are given in Table 4. For comparison with the method by Nikumbh and Pfeifer [14] that represents the complete restriction fragment with its ODH representation, we directly report the performances with oligomer length 3 from Table 1 in [14]. We observe that in experiments with segment-size 50 using 3-mers, *CoMIK* already achieves as good or better performance. Furthermore, the

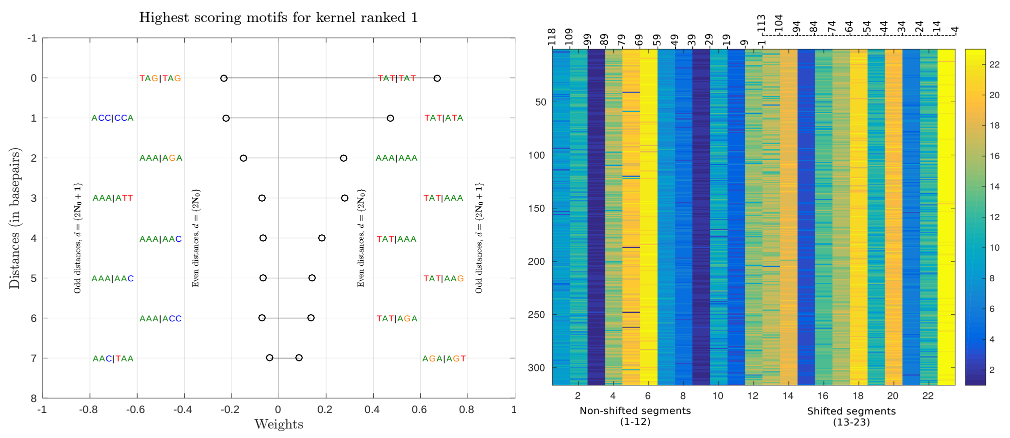


Figure 3 Distance-centric visualization of features (left) and visualization of weights assigned to segments per sequence for the yeast data set. As in Figure 2, the left panel shows 3-mer pairs that were assigned the highest positive and negative weights at each distance value corresponding to the sub-kernel with the highest weight among all sub-kernels in the collection. For easy viewing, the K-mer-pairs at odd distances are placed on the outside and the even distances, inside. Horizontal axis: weights, vertical axis: distances between 3-mer pairs (in basepairs). The right panel shows all sequences in the data set as segments: Segment rankings based on the weights assigned to the various segments are visualized as a heatmap. The rank to color mapping is as shown in the colorbar on the extreme right.

Table 4 5C data set results: Test AUC values (mean±s.d.) for *region* 0 [14] in three cell lines.

Method↓/Cell lines→	GM12878	K562	HeLa-S3
Nikumbh and Pfeifer [14]	0.7417 ± 0.059	0.8163 ± 0.071	0.6914 ± 0.058
CoMIK	0.7829 ± 0.063	0.7920 ± 0.084	0.6993 ± 0.012

additional ability of *CoMIK* to identify important portions in the individual sequences could give novel insights.

6 Discussion

We presented a multiple instance learning-based approach, called *CoMIK* ('Conformal Multi-Instance Kernels'), that can handle highly dispersed features in comparing variable-length sequences in a discriminative setting. We assessed the performance of *CoMIK* on three classification problems: a simulated data set and two real biological data sets including a 5C data set. Together with the visualizations, we demonstrated the efficacy of *CoMIK* in all these problems. As compared to KIRMES, where the classifier completely relies on the motif-finding step *a priori* for its input, *CoMIK*, by design, uses the complete sequence and is able to locate the portions deemed important for the prediction problem. This enables *CoMIK* to avert the risk of ignoring the low-affinity, weak binding sites in the sequences which can be missed by KIRMES. Technically, one could use the complete sequence with KIRMES provided the set of motifs considered are spread through-out the sequence, but that is again controlled by the motif-finding stage. *CoMIK* allows positional freedom in comparing sequences. For the 5C data set, Nikumbh and Pfeifer also used the ODH representation to compare the restriction fragments [14], but their approach does not give any information on the location of the features in the long restriction fragments. Our results on this data set showed that in this scenario looking closely at shorter segments rather than the complete restriction fragments can help attain better performance. Additionally, *CoMIK*'s ability to locate signal within the sequence could be useful in studying the so-called structural interactions between the intervening chromatin [18] of the long-range interacting loci.

We note that *CoMIK's* computation time is largely governed by the clustering step and the subsequent transformation of the segments – both performed at every CV iteration, and both of these are influenced by the choice of the segment-size. Our implementation exploits the sparsity of short individual segments, makes use of sparse representations and computations. Thus, while, in general, the segment-size only affects *CoMIK's* running time, for scenarios like the discussed yeast problem, shorter segments could be preferable. In the clustering step, the buckshot heuristic is incognizant of the imbalance prevalent in the data. This could be improved by using a stratified sample with buckshot clustering. We also note that for scenarios wherein positional information is important, kernels like the WDS [15] or the oligo kernel [13] could be more suitable as base kernels depending on the problem.

References

1 Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML'04, page 6, New York, NY, USA, 2004. ACM. `doi:10.1145/1015330.1015424`.

2 Matthew B. Blaschko and Thomas Hofmann. Conformal multi-instance kernels. In *NIPS 2006 Workshop on Learning to Compare Examples*, 2006.

3 Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT'92, pages 144–152, New York, NY, USA, 1992. ACM. `doi:10.1145/130385.130401`.

4 Jennifer E. F. Butler and James T. Kadonaga. The RNA polymerase II core promoter: a key component in the regulation of gene expression. *Genes & Development*, 16(20):2583–2592, 2002. `doi:10.1101/gad.1026202`.

5 Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'92, pages 318–329, New York, NY, USA, 1992. ACM. `doi:10.1145/133160.133214`.

6 Thomas G. Dietterich, Richard H. Lathrop, Tomas Lozano-Perez, and Arris Pharmaceutical. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.

7 Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence – Volume 2*, IJCAI'01, pages 973–978, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

8 Thomas Gärtner, Peter A. Flach, Adam Kowalczyk, and Alex J. Smola. Multi-instance kernels. In *Proc. 19th International Conf. on Machine Learning*, pages 179–186, Massachusetts, 2002. Morgan Kaufmann.

9 C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 7, pages 566–575, 2002.

10 Christina S. Leslie, Eleazar Eskin, Adiel Cohen, Jason Weston, and William Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004. `doi:10.1093/bioinformatics/btg431`.

11 Thomas Lingner and Peter Meinicke. Remote homology detection based on oligomer distances. *Bioinformatics*, 22(18):2224–2231, September 2006. doi:10.1093/bioinformatics/btl376.

12 Shai Lubliner, Ifat Regev, Maya Lotan-Pompan, Sarit Edelheit, Adina Weinberger, and Eran Segal. Core promoter sequence in yeast is a major determinant of expression level. *Genome research*, 25(7):1008–1017, 2015.

13 Peter Meinicke, Maike Tech, Burkhard Morgenstern, and Rainer Merkl. Oligo kernels for datamining on biological sequences: a case study on prokaryotic translation initiation sites. *BMC Bioinformatics*, 5(1):169, 2004. doi:10.1186/1471-2105-5-169.

14 Sarvesh Nikumbh and Nico Pfeifer. Genetic sequence-based prediction of long-range chromatin interactions suggests a potential role of short tandem repeat sequences in genome organization. *BMC Bioinformatics*, 18(1):218, 2017. doi:10.1186/s12859-017-1624-x.

15 G. Rätsch, S. Sonnenburg, and B. Schölkopf. RASE: recognition of alternatively spliced exons in C.elegans. *Bioinformatics*, 21(suppl 1):i369–i377, 2005. doi:10.1093/bioinformatics/bti1053.

16 Gunnar Rätsch and Sören Sonnenburg. Accurate splice site prediction for caenorhabditis elegans. In *Kernel Methods in Computational Biology*, MIT Press series on Computational Molecular Biology, pages 277–298. MIT Press, Cambridge, MA., 2004.

17 Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, July 2004. doi:10.1093/bioinformatics/bth141.

18 Amartya Sanyal, Bryan R. Lajoie, Gaurav Jain, and Job Dekker. The long-range interaction landscape of gene promoters. *Nature*, 489(7414):109–113, Sep 2012. doi:10.1038/nature11279.

19 Sebastian J. Schultheiss, Wolfgang Busch, Jan U. Lohmann, Oliver Kohlbacher, and Gunnar Rätsch. Kirmes: kernel-based identification of regulatory modules in euchromatic sequences. *Bioinformatics*, 25(16):2126–2133, 2009. doi:10.1093/bioinformatics/btp278.

20 John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

Forbidden Time Travel: Characterization of Time-Consistent Tree Reconciliation Maps[*]

Nikolai Nøjgaard[1], Manuela Geiß[2], Daniel Merkle[3], Peter F. Stadler[4], Nicolas Wieseke[5], and Marc Hellmuth[6]

1 Department of Mathematics and Computer Science, University of Greifswald, Greifswald, Germany; and
Department of Mathematics and Computer Science, University of Southern Denmark, Denmark
2 Bioinformatics Group, Department of Computer Science, and Interdisciplinary Center of Bioinformatics, University of Leipzig, Leipzig, Germany
3 Department of Mathematics and Computer Science, University of Southern Denmark, Denmark
4 Bioinformatics Group, Department of Computer Science, and Interdisciplinary Center of Bioinformatics, University of Leipzig, Leipzig, Germany; and
Max-Planck-Institute for Mathematics in the Sciences, Leipzig, Germany; and
Institute for Theoretical Chemistry, University of Vienna, Vienna, Austria; and
Santa Fe Institute, Santa Fe, NM, USA
5 Parallel Computing and Complex Systems Group, Department of Computer Science, Leipzig University, Leipzig, Germany
6 Department of Mathematics and Computer Science, University of Greifswald, Greifswald, Germany; and
Saarland University, Center for Bioinformatics, Saarbrücken, Germany

Abstract

Motivation: In the absence of horizontal gene transfer it is possible to reconstruct the history of gene families from empirically determined orthology relations, which are equivalent to *event-labeled* gene trees. Knowledge of the event labels considerably simplifies the problem of reconciling a gene tree T with a species trees S, relative to the reconciliation problem without prior knowledge of the event types. It is well-known that optimal reconciliations in the unlabeled case may violate time-consistency and thus are not biologically feasible. Here we investigate the mathematical structure of the event labeled reconciliation problem with horizontal transfer.

Results: We investigate the issue of time-consistency for the event-labeled version of the reconciliation problem, provide a convenient axiomatic framework, and derive a complete characterization of time-consistent reconciliations. This characterization depends on certain weak conditions on the event-labeled gene trees that reflect conditions under which evolutionary events are observable at least in principle. We give an $\mathcal{O}(|V(T)|\log(|V(S)|))$-time algorithm to decide whether a time-consistent reconciliation map exists. It does not require the construction of explicit timing maps, but relies entirely on the comparably easy task of checking whether a small auxiliary graph is acyclic.

Significance: The combinatorial characterization of time consistency and thus biologically feasible reconciliation is an important step towards the inference of gene family histories with horizontal transfer from orthology data, i.e., without presupposed gene and species trees. The fast algorithm to decide time consistency is useful in a broader context because it constitutes an attractive component for all tools that address tree reconciliation problems.

[*] Supported in part by the Danish Council for Independent Research, Natural Sciences, grants DFF-1323-00247 and DFF-7014-00041.

1998 ACM Subject Classification G.2.2 Graph Theory, G.2.3 Applications, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Tree Reconciliation, Horizontal Gene Transfer, Reconciliation Map, Time-Consistency, History of gene families

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.17

1 Introduction

Modern molecular biology describes the evolution of species in terms of the evolution of the genes that collectively form an organism's genome. In this picture, genes are viewed as atomic units whose evolutionary history *by definition* forms a tree. The phylogeny of species also forms a tree. This species tree is either interpreted as a consensus of the gene trees or it is inferred from other data. An interesting formal manner to define a species tree independent of genes and genetic data is discussed e.g. in [7]. In this contribution, we assume that gene and species trees are given independently of each other. The relationship between gene and species evolution is therefore given by a reconciliation map that describes how the gene tree is embedded in the species tree: after all, genes reside in organisms, and thus at each point in time can be assigned to a species.

From a formal point of view, a reconciliation map μ identifies vertices of a gene tree with vertices and edges in the species tree in such a way that (partial) ancestor relations given by the genes are preserved by μ. Vertices in the species tree correspond to speciation events. Since in this situation genes are faithfully transmitted from the parent species into both (all) daughter species, some of the vertices in the gene tree correspond to speciation events. Other important events considered here are gene duplications, in which two copies of a gene keep residing in the same species, and horizontal gene transfer events (HGT). Here, the original remains in the parental species, while the offspring copy "jumps" into a different branch of the species tree. It is customary to define pairwise relations between genes depending on the event type of their last common ancestor [8, 11, 13].

Most of the literature on this topic assumes that both the gene tree and the species tree are known. The aim is then to find a mapping of the gene tree T into the species tree S and, at least implicitly, an event-labeling on the vertices of the gene tree T. Here we take a different point of view and assume that T and the types of evolutionary events on T are known. This setting has ample practical relevance because event-labeled gene trees can be derived from the pairwise orthology relation [15, 13]. These relations in turn can be estimated directly from sequence data using a variety of algorithmic approaches that are based on the pairwise best match criterion and hence do not require any *a priori* knowledge of the topology of either the gene tree or the species tree, see e.g. [19, 2, 17, 1].

Genes that share a common origin (homologs) can be classified into orthologs, paralogs, and xenologs depending whether they originated by a speciation, duplication or horizontal gene transfer (HGT) event [8, 13]. Recent advances in mathematical phylogenetics [10, 11, 15] have shown that the knowledge of these event-relations (orthologs, paralogs and xenologs) suffices to construct event-labeled gene trees and, in some case, also a species tree.

Conceptually, both the gene tree and species tree are associated with a timing of each event. Reconciliation maps must preserve this timing information because there are *biologically infeasible* event labeled gene trees that cannot be reconciled with any species tree. In the absence of HGT, biologically feasibility can be characterized in terms of certain triples (rooted binary trees on three leaves) that are displayed by the gene trees [16]. In contrast, the timing information must be taken into account explicitly in the presence of HGT. In other words, there are gene trees with HGT that can be mapped to species trees only in such a way that some genes travels back in time.

There have been several attempts in the literature to handle this issue, see e.g. [6] for a review. In [18, 5] a *single* HGT adds timing constraints to a time map for a reconciliation to be found. Time-consistency is then subsequently defined based on the existence of a topological order of the digraph reflecting all the time constraints. In [20] NP-hardness was shown for finding a parsimonious time-consistent reconciliation based on a definition for time-consistency that essentially is based on considering *pairs* of HGTs. However, the latter definitions are explicitly designed for *binary* gene trees and do not apply to non-binary gene trees, which are used here to model incomplete knowledge of the exact gene phylogenies. Different algorithmic approaches for tackling time-consistency exist [6] such as the inclusion of time-zones known for specific evolutionary events. It is worth noting that *a posteriori* modifications of time-inconsistent solutions will in general violate parsimony [18]. So-far, no results have become available to determine the *existence* of time-consistent reconciliation maps given the (undated) species tree and the event-labeled gene tree.

Here, we introduce an axiomatic framework for time-consistent reconciliation maps and characterize for given event-labeled gene trees and species trees whether there exists a time-consistent reconciliation map. We provide an algorithm that constructs a time-consistent reconciliation map if one exists. The algorithms are implemented in `C++` using the boost graph library and are freely available at `https://github.com/Nojgaard/tc-recon`. In addition, the proofs and additional information on this paper are provided at this url.

2 Notation and Preliminaries

We consider *rooted trees* $T = (V, E)$ *(on* L_T*)* with root $\rho_T \in V$ and leaf set $L_T \subseteq V$. A vertex $v \in V$ is called a *descendant* of $u \in V$, $v \preceq_T u$, and u is an *ancestor* of v, $u \succeq_T v$, if u lies on the path from ρ_T to v. As usual, we write $v \prec_T u$ and $u \succ_T v$ to mean $v \preceq_T u$ and $u \neq v$. The partial order $\succeq_T$ is known as the *ancestor order* of T; the root is the unique maximal element w.r.t $\succeq_T$. If $u \preceq_T v$ or $v \preceq_T u$ then u and v are *comparable* and otherwise, *incomparable*. We consider edges of rooted trees to be directed away from the root, that is, the notation for edges (u, v) of a tree is chosen such that $u \succ_T v$. If (u, v) is an edge in T, then u is called *parent* of v and v *child* of u. It will be convenient for the discussion below to extend the ancestor relation $\preceq_T$ on V to the union of the edge and vertex sets of T. More precisely, for the edge $e = (u, v) \in E$ we put $x \prec_T e$ if and only if $x \preceq_T v$ and $e \prec_T x$ if and only if $u \preceq_T x$. For edges $e = (u, v)$ and $f = (a, b)$ in T we put $e \preceq_T f$ if and only if $v \preceq_T b$. For $x \in V$, we write $L_T(x) := \{y \in L_T \mid y \preceq_T x\}$ for the set of leaves in the subtree $T(x)$ of T rooted in x.

For a non-empty subset of leaves $A \subseteq L$, we define $\operatorname{lca}_T(A)$, or the *least common ancestor of* A, to be the unique $\preceq_T$-minimal vertex of T that is an ancestor of every vertex in A. In case $A = \{u, v\}$, we put $\operatorname{lca}_T(u, v) := \operatorname{lca}_T(\{u, v\})$. We have in particular $u = \operatorname{lca}_T(L_T(u))$ for all $u \in V$. We will also frequently use that for any two non-empty vertex sets A, B of a tree, it holds that $\operatorname{lca}(A \cup B) = \operatorname{lca}(\operatorname{lca}(A), \operatorname{lca}(B))$.

A *phylogenetic tree* is a rooted tree such that no interior vertex in $v \in V \setminus L_T$ has degree two, except possibly the root. If L_T corresponds to a *set of genes* $\mathbb{G}$ or *species* $\mathbb{S}$, we call a phylogenetic tree on L_T *gene tree* or *species tree*, respectively. In this contribution we will **not** restrict the gene or species trees to be binary, although this assumption is made implicitly or explicitly in much of the literature on the topic. The more general setting allows us to model incomplete knowledge of the exact gene or species phylogenies. Of course, all mathematical results proved here also hold for the special case of binary phylogenetic trees.

In our setting a gene tree $T = (V, E)$ on $\mathbb{G}$ is equipped with an *event-labeling* map $t : V \cup E \to I \cup \{0, 1\}$ with $I = \{\bullet, \square, \triangle, \odot\}$ that assigns to each interior vertex v of T a value $t(v) \in I$ indicating whether v is a speciation event ($\bullet$), duplication event ($\square$) or HGT event ($\triangle$). It is convenient to use the special label $\odot$ for the leaves x of T. Moreover, to each edge e a value $t(e) \in \{0, 1\}$ is added that indicates whether e is a *transfer edge* (1) or not (0). Note, only edges (x, y) for which $t(x) = \triangle$ might be labeled as transfer edge. We write $\mathcal{E} = \{e \in E \mid t(e) = 1\}$ for the set of transfer edges in T. We assume here that all edges labeled "0" transmit the genetic material vertically, that is, from an ancestral species to its descendants.

We remark that the restriction $t_{|V}$ of t to the vertex set V coincides with the "symbolic dating maps" introduced in [4]; these have a close relationship with cographs [10, 12, 14]. Furthermore, there is a map $\sigma : \mathbb{G} \to \mathbb{S}$ that assigns to each gene the species in which it resides. The set $\sigma(M)$, $M \subseteq \mathbb{G}$, is the set of species from which the genes M are taken. We write $(T; t, \sigma)$ for the gene tree $T = (V, E)$ with event-labeling t and corresponding map σ.

Removal of the transfer edges from $(T; t, \sigma)$ yields a forest $T_{\overline{\mathcal{E}}} := (V, E \setminus \mathcal{E})$ that inherits the ancestor order on its connected components, i.e., $\preceq_{T_{\overline{\mathcal{E}}}}$ iff $x \preceq_T y$ and x, y are in same subtree of $T_{\overline{\mathcal{E}}}$ [20]. Clearly $\preceq_{T_{\overline{\mathcal{E}}}}$ uniquely defines a root for each subtree and the set of descendant leaf nodes $L_{T_{\overline{\mathcal{E}}}}(x)$.

In order to account for duplication events that occurred before the first speciation event, we need to add an extra vertex and an extra edge "above" the last common ancestor of all species in the species tree $S = (V, E)$. Hence, we add an additional vertex to V (that is now the new root ρ_S of S) and the additional edge $(\rho_S, \mathrm{lca}_S(\mathbb{S}))$ to E. Strictly speaking S is not a phylogenetic tree in the usual sense, however, it will be convenient to work with these augmented trees. For simplicity, we omit drawing the augmenting edge $(\rho_S, \mathrm{lca}_S(\mathbb{S}))$ in our examples.

3 Observable Scenarios

The true history of a gene family, as it is considered here, is an arbitrary sequence of speciation, duplication, HGT, and gene loss events. The applications we envision for the theory developed, here, however assume that the gene tree and its event labels are inferred from (sequence) data, i.e., $(T; t, \sigma)$ is restricted to those labeled trees that can be constructed at least in principle from observable data. The issue here are gene losses that may completely eradicate the information on parts of the history. Specifically, we require that $(T; t, \sigma)$ satisfies the following three conditions:

(O1) Every internal vertex v has degree at least 3, except possibly the root which has degree at least 2.

(O2) Every HGT node has at least one transfer edge, $t(e) = 1$, and at least one non-transfer edge, $t(e) = 0$;

(O3) (a) If x is a speciation vertex, then there are at least two distinct children v, w of x such that the species V and W that contain v and w, resp., are incomparable in S.
(b) If (v, w) is a transfer edge in T, then the species V and W that contain v and w, resp., are incomparable in S.

Condition (O1) ensures that every event leaves a historical trace in the sense that there are at least two children that have survived in at least two of its subtrees. If this were not the case, no evidence would be left for all but one descendant tree, i.e., we would have no evidence that event v ever happened. We note that this condition was used e.g. in [16] for scenarios without HGT. Condition (O2) ensures that for an HGT event a historical trace remains of both the transferred and the non-transferred copy. If there is no transfer edge, we have no evidence to classify v as a HGT node. Conversely, if all edges were transfers, no evidence of the lineage of origin would be available and any reasonable inference of the gene tree from data would assume that the gene family was vertically transmitted in at least one of the lineages in which it is observed. In particular, Condition (O2) implies that for each internal vertex there is a path consisting entirely of non-transfer edges to some leaf. This excludes in particular scenarios in which a gene is transferred to a different "host" and later reverts back to descendants of the original lineage without any surviving offspring in the intermittent host lineage. Furthermore, a speciation vertex x cannot be observed from data if it does not "separate" lineages, that is, there are two leaf descendants of distinct children of x that are in distinct species. However, here we only assume to have the weaker Condition (O3.a) which ensures that any "observable" speciation vertex x separates at least locally two lineages. In other words, if all children of x would be contained in species that are comparable in S or, equivalently, in the same lineage of S, then there is no clear historical trace that justifies x to be a speciation vertex. In particular, most-likely there are two leaf descendants of distinct children of x that are in the same species even if only $T_{\overline{\mathcal{E}}}$ is considered. Hence, x would rather be classified as a duplication than as a speciation upon inference of the event labels from actual data. Analogously, if $(v, w) \in \mathcal{E}$ then v signifies the transfer event itself but w refers to the next (visible) event in the gene tree T. Given that (v, w) is a HGT-edge in the observable part, in a "true history" v is contained in a species V that transmits its genetic material (maybe along a path of transfers) to a contemporary species Z that is an ancestor of the species W containing w. Clearly, the latter allows to have $V \succeq_S W$ which happens if the path of transfers points back to the descendant lineage of V in S. In this case the transfer edge (v, w) must be placed in the species tree such that $\mu(v)$ and $\mu(w)$ are comparable in S. However, then there is no evidence that this transfer ever happened, and thus v would be rather classified as speciation or duplication vertex.

It can be shown that (O1), (O2) and (O3) imply Lemma 1 as well as two important properties (Σ1) and (Σ2) of event labeled species trees that play a crucial role for the results reported here.

▶ **Lemma 1.** *Let $\mathcal{T}_1, \ldots, \mathcal{T}_k$ be the connected components of $T_{\overline{\mathcal{E}}}$ with roots $\rho_1, \ldots, \rho_k$, respectively. If* (O2) *holds, then, $\{L_{T_{\overline{\mathcal{E}}}}(\rho_1), \ldots, L_{T_{\overline{\mathcal{E}}}}(\rho_k)\}$ forms a partition of $\mathbb{G}$.*

(Σ1) If $t(x) = \bullet$ then there are distinct children v, w of x in T such that $\sigma(L_{T_{\overline{\mathcal{E}}}}(v)) \cap \sigma(L_{T_{\overline{\mathcal{E}}}}(w)) = \emptyset$.

Intuitively, (Σ1) is true because within a component $T_{\overline{\mathcal{E}}}$ no genetic material is exchanged between non-comparable nodes. Thus, a gene separated in a speciation event necessarily ends up in distinct species in the absence of horizontal transfer. It is important to note that we do not require the converse: $\sigma(L_{T_{\overline{\mathcal{E}}}}(y)) \cap \sigma(L_{T_{\overline{\mathcal{E}}}}(y')) = \emptyset$ does **not** imply $t(\mathrm{lca}_T(L_{T_{\overline{\mathcal{E}}}}(y) \cup L_{T_{\overline{\mathcal{E}}}}(y')) = \bullet$, that is, the last common ancestor of two sets of genes from different species is not necessarily a speciation vertex.

Now consider a transfer edge $(v, w) \in \mathcal{E}$, i.e., $t(v) = \triangle$. Then $T_{\overline{\mathcal{E}}}(v)$ and $T_{\overline{\mathcal{E}}}(w)$ are subtrees of distinct connected components of $T_{\overline{\mathcal{E}}}$. Since HGT amounts to the transfer of genetic material *across* distinct species, the genes v and w must be contained in distinct species X and Y, respectively. Since no genetic material is transferred between contemporary species X' and Y' in $T_{\overline{\mathcal{E}}}$, where X' and Y' is a descendant of X and Y, respectively we derive

(Σ2) If $(v, w) \in \mathcal{E}$ then $\sigma(L_{T_{\overline{\mathcal{E}}}}(v)) \cap \sigma(L_{T_{\overline{\mathcal{E}}}}(w)) = \emptyset$.

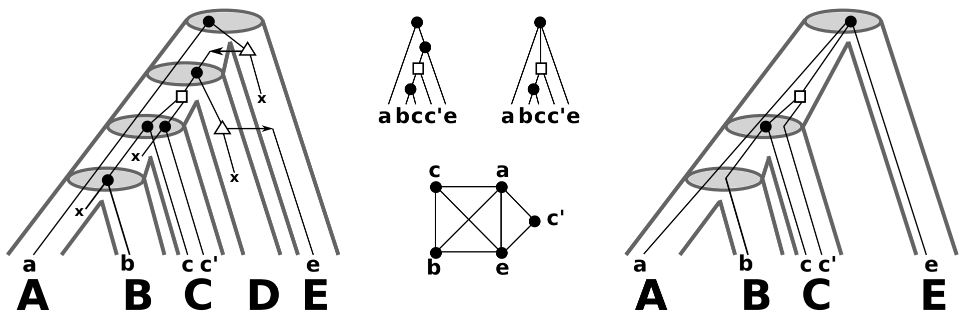


Figure 1 *Left:* A “true” evolutionary scenario for a gene tree with leaf set $\mathbb{G}$ evolving along the tube-like species trees is shown. The symbol “x” denotes losses. All speciations along the path from the root ρ_T to the leaf a are followed by losses and we omit drawing them.
Middle: The observable gene tree is shown in the upper-left. The orthology graph $G = (\mathbb{G}, E)$ (edges are placed between genes x, y for which $t(\text{lca}(x, y)) = \bullet$) is drawn in the lower part. This graph is a cograph and the corresponding *non-binary* gene tree T on $\mathbb{G}$ that can be constructed from such data is given in the upper-right part (cf. [10, 11, 13] for further details).
Right: Shown is species trees S on $\mathbb{S} = \sigma(\mathbb{G})$ with reconciled gene tree T. The reconciliation map μ for T and S is given implicitly by drawing the gene tree T within S. Note, this reconciliation is not consistent with DTL-scenarios [20, 3]. A DTL-scenario would require that the duplication vertex and the leaf a are incomparable in S. for further details.

From here on we simplify the notation a bit and write $\sigma_{T_{\overline{\mathcal{E}}}}(u) := \sigma(L_{T_{\overline{\mathcal{E}}}}(u))$. We are aware of the fact that condition (O3) cannot be checked directly for a given event-labeled gene tree. In contrast, (Σ1) and (Σ2) are easily determined. Hence, in the remainder of this paper we consider the more general case, that is, gene trees that satisfy (O1), (O2), (Σ1) and (Σ2).

4 Time-Consistent Reconciliation Maps

The problem of reconciliation between gene trees and species tree is formalized in terms of so-called DTL-scenarios in the literature [20, 3]. This framework, however, usually assumes that the event labels t on T are unknown, while a species tree S is given. The “usual” DTL axioms, furthermore, explicitly refer to binary, fully resolved gene and species trees. We therefore use a different axiom set here that is a natural generalization of the framework introduced in [16] for the HGT-free case:

▶ **Definition 2.** Let $T = (V, E)$ and $S = (W, F)$ be phylogenetic trees on $\mathbb{G}$ and $\mathbb{S}$, resp., $\sigma : \mathbb{G} \to \mathbb{S}$ the assignment of genes to species and $t : V \cup E \to \{\bullet, \square, \triangle, \odot\} \cup \{0, 1\}$ an event labeling on T. A map $\mu : V \to W \cup F$ is a *reconciliation map* if for all $v \in V$ it holds that:

(M1) *Leaf Constraint.* If $t(v) = \odot$, then $\mu(v) = \sigma(v)$.

(M2) *Event Constraint.*

- **(i)** If $t(v) = \bullet$, then $\mu(v) = \text{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(v))$.
- **(ii)** If $t(v) \in \{\square, \triangle\}$, then $\mu(v) \in F$.
- **(iii)** If $t(v) = \triangle$ and $(v, w) \in \mathcal{E}$, then $\mu(v)$ and $\mu(w)$ are incomparable in S.

(M3) *Ancestor Constraint.*

Suppose $v, w \in V$ with $v \prec_{T_{\overline{\mathcal{E}}}} w$.

- **(i)** If $t(v), t(w) \in \{\square, \triangle\}$, then $\mu(v) \preceq_S \mu(w)$,
- **(ii)** otherwise, i.e., at least one of $t(v)$ and $t(w)$ is a speciation $\bullet$, $\mu(v) \prec_S \mu(w)$.

We say that S is a *species tree for* $(T; t, \sigma)$ if a reconciliation map $\mu : V \to W \cup F$ exists.

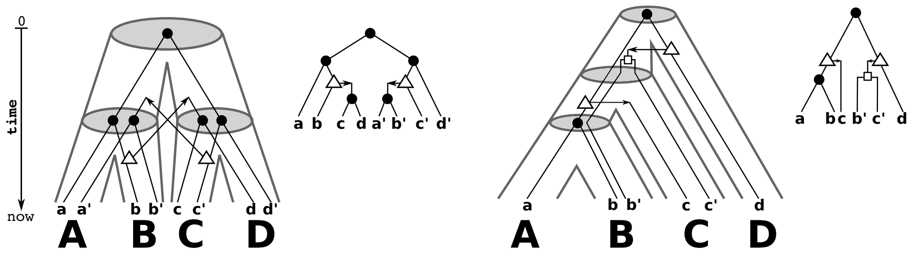


Figure 2 Shown are two (tube-like) species trees with reconciled gene trees. The reconciliation map μ for T and S is given implicitly by drawing the gene tree (upper right to the respective species tree) within the species tree. In the left example, the map μ is unique. However, μ is not time-consistent and thus, there is no time consistent reconciliation for T and S. In the example on the right hand side, μ is time-consistent.

It can be shown that the DTL axioms and the notation used here as in Definition 2 are equivalent in the case of binary trees. In Figure 1 an example of a biologically plausible reconciliation of non-binary trees that is valid w.r.t. Definition 2 is shown, however, it does not satisfy the conditions of a DTL-scenario.

Condition (M1) ensures that each leaf of T, i.e., an extant gene in $\mathbb{G}$, is mapped to the species in which it resides. Conditions (M2.i) and (M2.ii) ensure that each inner vertex of T is either mapped to a vertex or an edge in S such that a vertex of T is mapped to an interior vertex of S if and only if it is a speciation vertex. Condition (M2.i) might seem overly restrictive, an issue to which we will return below. Condition (M2.iii) satisfies condition (O3) and maps the vertices of a transfer edge in a way that they are incomparable in the species tree, since a HGT occurs between distinct (co-existing) species. It becomes void in the absence of HGT; thus Definition 2 reduces to the definition of reconciliation maps given in [16] for the HGT-free case. Importantly, condition (M3) refers only to the connected components of $T_{\overline{\mathcal{E}}}$ since comparability w.r.t. $\prec_{T_{\overline{\mathcal{E}}}}$ implies that the path between x and y in T does not contain transfer edges. It ensures that the ancestor order $\preceq_T$ of T is preserved along all paths that do not contain transfer edges.

We will make use of the following bound that effectively restricts how close to the leafs the image of a vertex in the gene tree can be located.

▶ **Lemma 3.** *If $\mu : (T; t, \sigma) \to S$ satisfies (M1) and (M3), then $\mu(u) \succeq_S \operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u))$ for any $u \in V(T)$.*

Proof. If u is a leaf, then by Condition (M1) $\mu(u) = \sigma(u)$ and we are done. Thus, let u be an interior vertex. By Condition (M3), $z \preceq_S \mu(u)$ for all $z \in \sigma_{T_{\overline{\mathcal{E}}}}(u)$. Hence, if $\mu(u) \prec_S \operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u))$ or if $\mu(u)$ and $\operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u)))$ are incomparable in S, then there is a $z \in \sigma_{T_{\overline{\mathcal{E}}}}(u)$ such that z and $\mu(u)$ are incomparable; contradicting (M3). ◀

Condition (M2.i) implies in particular the weaker property "(M2.i') if $t(v) = \bullet$ then $\mu(v) \in W$". In the light of Lemma 3, $\mu(v) = \operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(v))$ is the lowest possible choice for the image of a speciation vertex. Clearly, this restricts the possibly exponentially many reconciliation maps for which $\mu(v) \succ_S \operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(v))$ for speciation vertices v is allowed to only those that satisfy (M2.i). However, the latter is justified by the observation that if v is a speciation vertex with children u, w, then there is only one unique piece of information given by the gene tree to place $\mu(v)$, that is, the unique vertex x in S with children y, z such that $\sigma_{T_{\overline{\mathcal{E}}}}(u) \subseteq L_S(y)$ and $\sigma_{T_{\overline{\mathcal{E}}}}(w) \subseteq L_S(z)$. The latter arguments easily generalizes to the case that v has more than two children in T. Moreover, any *observable* speciation node $v' \succ_T v$ closer to the root

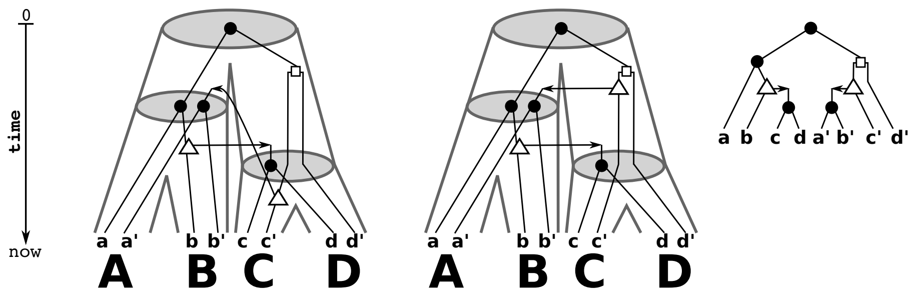


Figure 3 Shown are a gene tree $(T; t, \sigma)$ (right) and two identical (tube-like) species trees S (left and middle). There are two possible reconciliation maps for T and S that are given implicitly by drawing T within the species tree S. These two reconciliation maps differ only in the choice of placing the HGT-event either on the edge $(\text{lca}_S(C,D), C)$ or on the edge $(\text{lca}_S(\{A,B,C,D\}), \text{lca}_S(C,D))$. In the first case, it is easy to see that μ would not be time-consistent, i.e., there are no time maps τ_T and τ_S that satisfy (C1) and (C2). The reconciliation map μ shown in the middle is time-consistent.

than v must be mapped to a node ancestral to $\mu(v)$ due to (M3.ii). Therefore, we require $\mu(v) = x = \text{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(v))$ here.

If S is a species tree for the gene tree (T, t, σ) then there is no freedom in the construction of a reconciliation map μ on the set $\{x \in V(T) \mid t(x) \in \{\bullet, \odot\}\}$. The duplication and HGT vertices of T, however, can be placed differently. As a consequence there is a possibly exponentially large set of reconciliation maps from (T, t, σ) to S.

From a biological point of view, however, the notion of reconciliation used so far is too weak. In the absence of HGT, subtrees evolve independently and hence, the linear order of points along each path from root to leaf is consistent with a global time axis. This is no longer true in the presence of HGT events, because HGT events imply additional time-consistency conditions. These stem from the fact that the appearance of the HGT copy in a distant subtree of S is concurrent with the HGT event. To investigate this issue in detail, we introduce time maps and the notion of time-consistency, see Figures 2 – 4 for illustrative examples.

▶ **Definition 4** (Time Map)**.** The map $\tau_T : V(T) \to \mathbb{R}$ is a time map for the rooted tree T if $x \prec_T y$ implies $\tau_T(x) > \tau_T(y)$ for all $x, y \in V(T)$.

▶ **Definition 5.** A reconciliation map μ from $(T; t, \sigma)$ to S is *time-consistent* if there are time maps τ_T for T and τ_S for S for all $u \in V(T)$ satisfying the following conditions:
(C1) If $t(u) \in \{\bullet, \odot\}$, then $\tau_T(u) = \tau_S(\mu(u))$.
(C2) If $t(u) \in \{\square, \triangle\}$ and, thus $\mu(u) = (x, y) \in E(S)$, then $\tau_S(y) > \tau_T(u) > \tau_S(x)$.

Condition (C1) is used to identify the time-points of speciation vertices and leaves u in the gene tree with the time-points of their respective images $\mu(u)$ in the species trees. In particular, all genes u that reside in the same species must be assigned the same time point $\tau_T(u) = \tau_S(\sigma(u))$. Analogously, all speciation vertices in T that are mapped to the same speciation in S are assigned matching time stamps, i.e., if $t(u) = t(v) = \bullet$ and $\mu(u) = \mu(v)$ then $\tau_T(u) = \tau_T(v) = \tau_S(\mu(u))$.

To understand the intuition behind (C2) consider a duplication or HGT vertex u. By construction of μ it is mapped to an edge of S, i.e., $\mu(u) = (x, y)$ in S. The time point of u must thus lie between time points of x and y. Now suppose $(u, v) \in \mathcal{E}$ is a transfer edge. By construction, u signifies the transfer event itself. The node v, however, refers to the next (visible) event in the gene tree. Thus $\tau_T(u) < \tau_T(v)$. In particular, $\tau_T(v)$ must not be

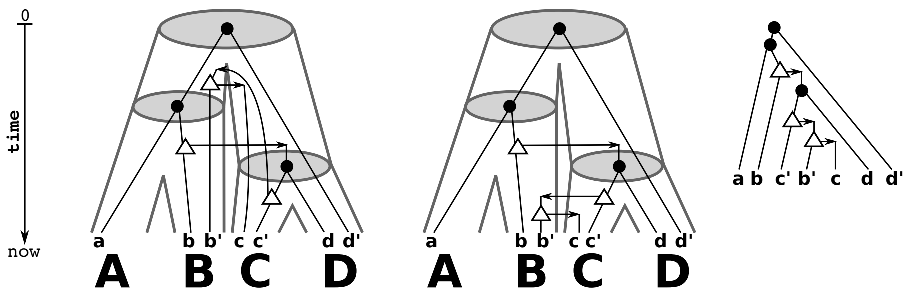


Figure 4 Shown are a gene tree $(T;t,\sigma)$ (right) and two identical (tube-like) species trees S (left and middle). There are two possible reconciliation maps for T and S that are given implicitly by drawing T within the species tree S. The left reconciliation maps each gene tree vertex as high as possible into the species tree. However, in this case only the middle reconciliation map is time-consistent.

misinterpreted as the time of introducing the HGT-duplicate into the new lineage. While this time of course exists (and in our model coincides with the timing of the transfer event) it is not marked by a visible event in the new lineage, and hence there is no corresponding node in the gene tree T.

W.l.o.g. we fix the time axis so that $\tau_T(\rho_T) = 0$ and $\tau_S(\rho_S) = -1$. Thus, $\tau_S(\rho_S) < \tau_T(\rho_T) < \tau_T(u)$ for all $u \in V(T) \setminus \{\rho_T\}$.

Clearly, a necessary condition to have biologically feasible gene trees is the existence of a reconciliation map μ. However, not all reconciliation maps are time-consistent, see Fig. 2.

▶ **Definition 6.** An event-labeled gene tree $(T;t,\sigma)$ is *biologically feasible* if there exists a time-consistent reconciliation map from $(T;t,\sigma)$ to some species tree S.

As a main result of this contribution, we provide simple conditions that characterize (the existence of) time-consistent reconciliation maps and thus, provides a first step towards the characterization of biologically feasible gene trees.

▶ **Theorem 7.** *Let μ be a reconciliation map from $(T;t,\sigma)$ to S. There is a* time-consistent *reconciliation map from $(T;t,\sigma)$ to S if and only if there are two time-maps τ_T and τ_S for T and S, respectively, such that the following conditions are satisfied for all $x \in V(S)$:*
(D1) *If $\mu(u) = x$, for some $u \in V(T)$ then $\tau_T(u) = \tau_S(x)$.*
(D2) *If $x \preceq_S \operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u))$ for some $u \in V(T)$ with $t(u) \in \{\square, \triangle\}$, then $\tau_S(x) > \tau_T(u)$.*
(D3) *If $\operatorname{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u) \cup \sigma_{T_{\overline{\mathcal{E}}}}(v)) \preceq_S x$ for some $(u,v) \in \mathcal{E}$, then $\tau_T(u) > \tau_S(x)$.*

From the algorithmic point of view it is desirable to design methods that allow to check whether a reconciliation map is time-consistent. Moreover, given a gene tree T and species tree S we wish to decide whether there exists a time-consistent reconciliation map μ, and if so, we should be able to construct μ.

To this end, observe that any constraints given by Definition 4, Theorem 7 (D2)–(D3), and Definition 5 (C2) can be expressed as a total order on $V(S) \cup V(T)$, while the constraints (C1) and (D1) together suggest that we can treat the preimage of any vertex in the species tree as a "single vertex". In fact we can create an auxiliary graph in order to answer questions that are concerned with time-consistent reconciliation maps.

▶ **Definition 8.** Let μ be a reconciliation map from $(T;t,\sigma)$ to S. The *auxiliary graph* A is defined as a directed graph with a vertex set $V(A) = V(S) \cup V(T)$ and an edge-set $E(A)$ that is constructed as follows:

Algorithm 1 Check existence and construct time-consistent reconciliation map

```
Precondition: $S = (W, F)$ is a species tree for $T = (V, E)$.
1: $\ell \leftarrow$ ComputeLcaSigma($(T; t, \sigma), S$)
2: $\mu(u) \leftarrow \emptyset$ for all $u \in V$                    ▷ "$\emptyset$" means uninitialized
3: for all $u \in V$ do
4:     if $t(u) \in \{\bullet, \odot\}$ then $\mu(u) \leftarrow \ell(u)$
5:     else $\mu(u) \leftarrow (p(\ell(u)), \ell(u))$          ▷ $p(\ell(u))$ denotes the parent of $\ell(u)$
6: Compute the auxiliary graph $A_2$
7: if $A_2$ contains a cycle then return "No time-consistent reconciliation map exists."
8: Let $\tau : V(A_2) \to \mathbb{R}$ such that if $(x, y) \in E(A_2)$ then $\tau(x) < \tau(y)$
9:                       ▷ W.l.o.g. we can assume that $\tau(x) \neq \tau(y)$ for all $x, y \in V(A_2)$
10: $\tau_S \leftarrow$ A time map such that $\tau_S(x) = \tau(x)$ for all $x \in W$
11: $\tau_T \leftarrow$ A time map such that $\tau_T(u) = \tau(\mu(u))$ if $t(u) \in \{\bullet, \odot\}$, otherwise $\tau_T(u) = \tau(u)$
    for all $u \in V$.
12: for $u \in V$ where $t(u) \in \{\square, \triangle\}$ do
13:     while it does not hold that $\tau_S(x) < \tau_T(u) < \tau_S(y)$ for $(x, y) = \mu(u)$ do
14:         $\mu(u) \leftarrow (p(x), x)$
15: return $\mu$
```

(A1) For each $(u, v) \in E(T)$ we have $(u', v') \in E(A)$, where

$$u' = \begin{cases} \mu(u) & \text{if } t(u) \in \{\odot, \bullet\} \\ u & \text{otherwise} \end{cases}, \quad v' = \begin{cases} \mu(v) & \text{if } t(v) \in \{\odot, \bullet\} \\ v & \text{otherwise} \end{cases},$$

(A2) For each $(x, y) \in E(S)$ we have $(x, y) \in E(A)$.
(A3) For each $u \in V(T)$ with $t(u) \in \{\square, \triangle\}$ we have $(u, lca_S(\sigma_{T_{\overline{\mathcal{E}}}}(u))) \in E(A)$.
(A4) For each $(u, v) \in \mathcal{E}$ we have $(lca_S(\sigma_{T_{\overline{\mathcal{E}}}}(u) \cup \sigma_{T_{\overline{\mathcal{E}}}}(v)), u) \in E(A)$
(A5) For each $u \in V(T)$ with $t(u) \in \{\triangle, \square\}$ and $\mu(u) = (x, y) \in E(S)$ we have $(x, u) \in E(A)$ and $(u, y) \in E(A)$.

We define A_1 and A_2 as the subgraphs of A that contain only the edges defined by (A1), (A2), (A5) and (A1), (A2), (A3), (A4), respectively.

We note that the edge sets defined by conditions (A1) through (A5) are not necessarily disjoint. The mapping of vertices in T to edges in S is considered only in condition (A5). The following two theorems are the key results of this contribution.

▶ **Theorem 9.** *Let μ be a reconciliation map from $(T; t, \sigma)$ to S. The map μ is time-consistent if and only if the auxiliary graph A_1 is a directed acyclic graph (DAG).*

▶ **Theorem 10.** *Assume there is a reconciliation map μ from $(T; t, \sigma)$ to S. There is a time-consistent reconciliation map, possibly different from μ, from $(T; t, \sigma)$ to S if and only if the auxiliary graph A_2 (defined on μ) is a DAG.*

Naturally, Theorems 9 or 10 can be used to devise algorithms for deciding time-consistency. To this end, the efficient computation of $\mathrm{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u))$ for all $u \in V(T)$ is necessary. This can be achieved with Algorithm 2 in $O(|V(T)| \log(|V(S)|))$. More precisely, we have the following statement.

▶ **Lemma 11.** *For a given gene tree $(T = (V, E); t, \sigma)$ and a species tree $S = (W, F)$, Algorithm 2 correctly computes $\ell(u) = \mathrm{lca}_S(\sigma_{T_{\overline{\mathcal{E}}}}(u))$ for all $u \in V(T)$ in $O(|V| \log(|W|))$ time.*

Let S be a species tree for $(T; t, \sigma)$, that is, there is a valid reconciliation between the two trees. Algorithm 1 describes a method to construct a time-consistent reconciliation map for $(T; t, \sigma)$ and S, if one exists, else "No time-consistent reconciliation map exists" is returned. First, an arbitrary reconciliation map μ that satisfies the condition of Def. 2 is computed. Second, Theorem 10 is utilized and it is checked whether the auxiliary graph A_2 is not a DAG in which case no time-consistent map μ exists for $(T; t, \sigma)$ and S. Finally, if A_2 is a DAG, then we continue to adjust μ to become time-consistent.

▶ **Theorem 12.** *Let $S = (W, F)$ be species tree for the gene tree $(T = (V, E); t, \sigma)$. Algorithm 1 correctly determines whether there is a time-consistent reconciliation map μ and in the positive case, returns such a μ in $O(|V| \log(|W|))$ time.*

5 Outlook and Summary

We have characterized here whether a given event-labeled gene tree $(T; t, \sigma)$ and species tree S can be reconciled in a time-consistent manner in terms of two auxiliary graphs A_1 and A_2 that must be DAGs. These are defined in terms of given reconciliation maps. This condition yields an $O(|V| \log(|W|))$-time algorithm to check whether a given reconciliation map μ is time-consistent, and an algorithm with the same time complexity for the construction of a time-consistent reconciliation maps, provided one exists.

Our results depend on three conditions on the event-labeled gene trees that are motivated by the fact that event-labels can be assigned to internal vertices of gene trees only if there is observable information on the event. The question which event-labeled gene trees are actually observable given an arbitrary, true evolutionary scenario deserves further investigation in future work. Here we have used conditions that arguable are satisfied when gene trees are inferred using sequence comparison and synteny information. A more formal theory of observability is still missing, however.

Our results provide an efficient way of deciding whether a *given* pair of gene and species tree can be time-consistently reconciled. There are, however, in general exponentially many putative species trees. This begs the question whether there is *at least one* species tree S for a gene tree and if so, how to construct S. "Informative triples" extracted from the gene tree answer this question in the absence of HGT [16]. It is plausible that this idea can be generalized to our current setting to provide at least a partial characterization [9].

Acknowledgment. We thank the organizers of the 32nd TBI Winterseminar 2017 in Bled (Slovenia), where the authors participated, met and basically drafted the main ideas of this paper, while drinking a cold and tasty red Union, or was it a green Laško?

References

1 A. M. Altenhoff, B. Boeckmann, S. Capella-Gutierrez, D. A. Dalquen, T. DeLuca, K. Forslund, J. Huerta-Cepas, B. Linard, C. Pereira, L. P. Pryszcz, F. Schreiber, A. S. da Silva, D. Szklarczyk, C. M. Train, P. Bork, O. Lecompte, C. von Mering, I. Xenarios, K. Sjölander, L. J. Jensen, M. J. Martin, M. Muffato, T. Gabaldón, S. E. Lewis, P. D. Thomas, E. Sonnhammer, and C. Dessimoz. Standardized benchmarking in the quest for orthologs. *Nature Methods*, 13:425–430, 2016.

2 A. M. Altenhoff and C. Dessimoz. Phylogenetic and functional assessment of orthologs inference projects and methods. *PLoS Comput Biol.*, 5:e1000262, 2009.

3 M. S. Bansal, E. J. Alm, and M. Kellis. Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. *Bioinformatics*, 28(12):i283–i291, 2012.

4 S. Böcker and A. W. M. Dress. Recovering symbolically dated, rooted trees from symbolic ultrametrics. *Adv. Math.*, 138:105–125, 1998.

5 M. A. Charleston. Jungles: a new solution to the host/parasite phylogeny reconciliation problem. *Math Biosci.*, 149(2):191–223, 1998.

6 J.-P. Doyon, V. Ranwez, V. Daubin, and V. Berry. Models, algorithms and programs for phylogeny reconciliation. *Briefings in Bioinformatics*, 12(5):392, 2011.

7 A. Dress, V. Moulton, M. Steel, and T. Wu. Species, clusters and the 'tree of life': A graph-theoretic perspective. *J. Theor. Biol.*, 265:535–542, 2010.

8 W. M. Fitch. Homology: a personal view on some of the problems. *Trends Genet.*, 16:227–231, 2000.

9 M. Hellmuth. Biologically feasible gene trees, reconciliation maps and informative triples, 2017. (submitted) arXiv:1701.07689.

10 M. Hellmuth, M. Hernandez-Rosales, K. T. Huber, V. Moulton, P. F. Stadler, and N. Wieseke. Orthology relations, symbolic ultrametrics, and cographs. *J. Math. Biology*, 66(1-2):399–420, 2013.

11 M. Hellmuth, P. F. Stadler, and N. Wieseke. The mathematics of xenology: Di-cographs, symbolic ultrametrics, 2-structures and tree- representable systems of binary relations. *Journal of Mathematical Biology*, 2016. DOI: 10.1007/s00285-016-1084-3.

12 M. Hellmuth and N. Wieseke. On symbolic ultrametrics, cotree representations, and cograph edge decompositions and partitions. In Dachuan et al., editor, *Proceedings COCOON 2015*, pages 609–623, Cham, 2015. Springer International Publishing.

13 M. Hellmuth and N. Wieseke. From sequence data including orthologs, paralogs, and xenologs to gene and species trees. In Pierre Pontarotti, editor, *Evolutionary Biology: Convergent Evolution, Evolution of Complex Traits, Concepts and Methods*, pages 373–392, Cham, 2016. Springer.

14 M. Hellmuth and N. Wieseke. On tree representations of relations and graphs: Symbolic ultrametrics and cograph edge decompositions. *J. Comb. Opt.*, 2017. `doi:DOI10.1007/s10878-017-0111-7`.

15 M. Hellmuth, N. Wieseke, M. Lechner, H.-P. Lenhof, M. Middendorf, and P. F. Stadler. Phylogenomics with paralogs. *Proceedings of the National Academy of Sciences*, 112(7):2058–2063, 2015. `doi:10.1073/pnas.1412770112`.

16 M. Hernandez-Rosales, M. Hellmuth, N. Wieseke, K. T. Huber, V. Moulton, and P. F. Stadler. From event-labeled gene trees to species trees. *BMC Bioinformatics*, 13(Suppl 19):S6, 2012.

17 M. Lechner, M. Hernandez-Rosales, D. Doerr, N. Wieseke, A. Thévenin, J. Stoye, R. K. Hartmann, S. J. Prohaska, and P. F. Stadler. Orthology detection combining clustering and synteny for very large datasets. *PLoS ONE*, 9(8):e105015, 08 2014.

18 D. Merkle and M. Middendorf. Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information. *Theory in Biosciences*, 4:277–299, 2005.

19 A. C. J. Roth, G. H. Gonnet, and C. Dessimoz. Algorithm of OMA for large-scale orthology inference. *BMC Bioinformatics*, 9:518, 2008.

20 A. Tofigh, M. Hallett, and J. Lagergren. Simultaneous identification of duplications and lateral gene transfers. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(2):517–535, 2011.

Rainbowfish: A Succinct Colored de Bruijn Graph Representation[*]

Fatemeh Almodaresi[1], Prashant Pandey[2], and Rob Patro[3]

1 **Stony Brook University, Stony Brook, NY, USA**
`falmodaresit@cs.stonybrook.edu`
2 **Stony Brook University, Stony Brook, NY, USA**
`ppandey@cs.stonybrook.edu`
3 **Stony Brook University, Stony Brook, NY, USA**
`rob.patro@cs.stonybrook.edu`

Abstract

The colored de Bruijn graph – a variant of the de Bruijn graph which associates each edge (i.e., k-mer) with some set of colors – is an increasingly important combinatorial structure in computational biology. Iqbal et al. demonstrated the utility of this structure for representing and assembling a collection (population) of genomes, and showed how it can be used to accurately detect genetic variants. Muggli et al. introduced VARI, a representation of the colored de Bruijn graph that adopts the BOSS representation for the de Bruijn graph topology and achieves considerable savings in space over `Cortex`, albeit with some sacrifice in speed. The memory-efficient representation of VARI allows the colored de Bruijn graph to be constructed and analyzed for large datasets, beyond what is possible with `Cortex`.

In this paper, we introduce Rainbowfish, a succinct representation of the color information of the colored de Bruijn graph that reduces the space usage even further. Our representation also uses BOSS to represent the de Bruijn graph, but decomposes the color sets based on an equivalence relation and exploits the inherent skewness in the distribution of these color sets. The Rainbowfish representation is compressed based on the 0th-order entropy of the color sets, which can lead to a significant reduction in the space required to store the relevant information for each edge. In practice, Rainbowfish achieves up to a 20× improvement in space over VARI. Rainbowfish is written in `C++11` and is available at `https://github.com/COMBINE-lab/rainbowfish`.

1998 ACM Subject Classification E.1 Data Structures, E.2 Data Storage and Representations, E.4 Coding and Information Theory

Keywords and phrases de Bruijn graph, succinct data structures, rank and select operation, colored de Bruijn graph

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.18

1 Introduction and Related Work

This paper proposes a new representation of the colored de Bruijn graph. The colored de Bruijn graph is a variant of the de Bruijn graph where each edge (i.e., k-mer) is associated with some set of colors. Here, each color is used to encode the source of the corresponding k-mers (e.g., different source genomes, transcriptomes, sequenced samples, etc.). From this perspective, it is a flexible and powerful combinatorial structure for representing a collection of sequences while maintaining the identity of each. This structure gained popularity in the

[*] We gratefully acknowledge support from NSF grant BBSRC-NSF/BIO-1564917.

work of Iqbal et al. [12], which demonstrated the utility of the colored de Bruijn graph for representing and assembling a collection (population) of genomes, and for detecting both simple and complex genetic variants with high accuracy. Analysis of the colored de Bruijn graph exhibits particular promise for analyzing complex population-level variation, since topological structures (e.g., bubbles) can be associated with variation in the underlying sub-populations. The representation adopted by Iqbal, as implemented in the tool `Cortex`, is optimized for speed, and so requires a considerable amount of memory to represent both the topology of the de Bruijn graph and the colors associated with each edge.

The memory usage of the colored de Bruijn graph representation adopted in `Cortex` precludes this approach from being adopted when the underlying genomes and color sets become too large. In order to overcome such limitations, Muggli et al. [16] introduced the VARI representation of the colored de Bruijn graph. This approach sacrifices some of the speed of the `Cortex` representation for a considerable reduction in the required space. VARI achieves this space savings in two ways. First, rather than using a hash-table-based representation of the de Bruijn graph topology, it adopts the highly-efficient BOSS representation. The BOSS [1] representation (named based on the initials of the authors) makes use of the FM index [7] to encode the topology of the de Bruijn graph. BOSS uses $4N + o(N)$ bits to represent a de Bruijn graph with N edges (empirically, this often works out to be as few as 4-6 bits per edge).

VARI couples the BOSS representation of the de Bruijn graph topology with a compressed representation of the color information. By its nature, BOSS assigns to every de Bruijn graph edge a distinct rank in the range $[0, N)$. So, VARI represents the color information as a $N \times C$ bit matrix where C is the number of input colors. Conceptually, each of the N rows of this matrix is simply a bit vector that encodes which of the C colors label the corresponding edge. To reduce the space required to store this color information, VARI concatenates these rows into a single vector over $N \times C$ coordinates and stores them in an Elias-Fano [5, 6] encoded bit vector, allowing for a (sometimes substantial) reduction in the size while still enabling efficient point queries (i.e., is a particular edge labeled with a given color?). Muggli et al. [16] demonstrate that the VARI representation can be built on data sets consisting of large numbers of k-mers, large input color sets, or both. Specifically, the space efficiency of VARI makes it possible to build and query the colored de Bruijn graph on datasets that are orders of magnitude larger than what is possible with `Cortex`. This is an exciting development that opens up this methodology for increasingly large-scale analysis.

Though VARI achieves a substantial improvement in space over `Cortex`, there is still a considerable amount of redundancy present in its representation. Both of these systems represent the color set corresponding to each k-mer independently of other k-mers. Hence a considerable amount of redundant information can be present when the color set for each k-mer is represented independently. In fact, some existing colored de Bruijn Graph representations, like the Bloom Filter Trie [11] exploit this redundancy to compress shared color information, and share certain ideas and motivation with the representation proposed in this paper. However, many of the possible subsets of colors do not occur in practice, and there is an inherent (often extreme) skewness in the distribution of the color sets that do appear. It becomes even more important to exploit this skewness for large metagenomic datasets because the space usage of VARI for these datasets can become impractical.

In this paper, we introduce a succinct representation, called Rainbowfish, of the color sets associated to each edge in the de Bruijn graph. We also adopt the BOSS representation of the de Bruijn graph topology, and focus, specifically, on how to concisely represent the color information. Rainbowfish's colored de Bruijn graph representation is entropy compressed

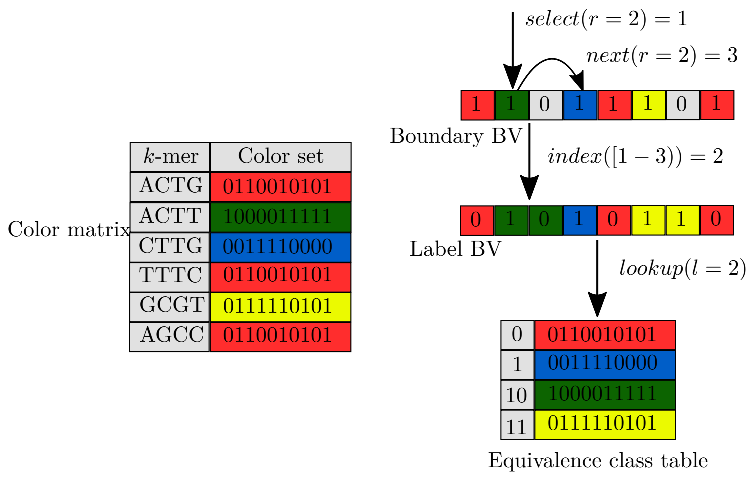


Figure 1 The representation of color information in Rainbowfish. The "Color Matrix" at the top represents 6 distinct 4-mers, each assigned a color set. 3 of these 4-mers (ACTG, TTTC, AGCC) have the same color class, labeled 0, and the other 3 (CTTG, ACTT, and GCGT) each have color classes labeled 1, 10, and 11 respectively. To retrieve the color set for a k-mer, we first perform select on the boundary bit vector (BBV) using rank r of the corresponding edge (k-mer). This returns the label's starting position, i. We then look for the next set bit BBV to find the label's ending position, j. Then, we fetch the label at indices i to j in label bit vector (LBV). Finally, we lookup the label l in the equivalence class table (ECT) and return the color class corresponding to the label. A detailed explanation of the data structure and its construction is given in Section 3.1.

and exploits the high skewness present in the distribution of color sets. By exploiting a more efficient decomposition of the set of present colors (i.e., in terms of equivalence classes), we achieve a considerable reduction over the space required by VARI (up to 20× depending on the dataset), while still retaining efficient (i.e., constant time) queries.

2 Background and definitions

Rainbowfish is a succinct representation of the color information, and uses rank and select operations to lookup the color class corresponding to k-mers in the de Bruijn graph. Here, we briefly recapitulate the definition of a succinct data structure and the rank and select operations.

A ***succinct data structure*** consumes an amount of space that is close to the information-theoretic optimum. More precisely, if Z denotes the information-theoretic optimal space usage for a given data structure, then a succinct data structure uses $Z + o(Z)$ space [14].

rank and ***select*** [13] are operations that are commonly used for navigating within succinct data structures. For a bit vector $B[0, \dots, n-1]$, $\textsc{rank}(j)$ returns the number of 1s in the prefix $B[0, \dots, j]$ of B. $\textsc{select}(r)$ returns the position of the rth 1, that is, the smallest index j such that $\textsc{rank}(j) = r$. For example, for the 12-bit vector $B[0, \dots, 11] =$100101001010, $\textsc{rank}(5) = 3$, because there are three bits set to one in the 6-bit prefix $B[0, \dots, 5]$ of B, and $\textsc{select}(4) = 8$, because $B[8]$ is the fourth 1 in the bit vector.

3 Method

In this section we first describe the design of Rainbowfish. We then analyze the space usage and provide a lower bound for the representation of sets of colors given a ranking of de Bruijn graph edges. Finally, we discuss the Rainbowfish implementation.

3.1 Design

Rainbowfish's compact representation of color information is based on two particular observations. First, it is often the case that many of the k-mers in a colored de Bruijn graph share the same set of colors. More formally, we define an equivalence relation $\sim$ over the set of k-mers in the de Bruijn graph. Let $\mathtt{Col}(\cdot)$ denote the function that maps each k-mer to its corresponding set of colors. We say that two k-mers are color-equivalent (i.e., $k_1 \sim k_2$) if and only if $\mathtt{Col}(k_1) = \mathtt{Col}(k_2)$. We will refer to the set of colors shared by the k-mers related by $\sim$ as a ***color class***. If C, the number of input colors, is large, it is often the case that the number of distinct color classes is far less than the number of possible color classes (which is bounded above by $\min(N, 2^C)$).

Second, it is often the case that the frequency distribution of color classes is far from uniform. Hence, it will often be useful to record a frequently occurring color class using a short description (i.e., a small number of bits) while reserving larger descriptions for less frequent color classes.

The design of Rainbowfish is motivated by the above observations. Instead of storing the color set for each k-mer separately, Rainbowfish stores each distinct color class only once and assigns to each distinct class a label (which, practically, is much smaller than the unary encoding of the color class itself). It then stores, for each k-mer, the label of the color class to which it belongs.

The approach we use to assign variable-length labels to color classes is similar in spirit to the construction of a Huffman code, where the message is a string of color class symbols. However, we do not build a prefix code, and instead opt to store an additional bit vector to allow the efficient selection of an arbitrary label from the list. We generate the labels according to the following procedure. We first sort, in descending order, all the color classes based on their frequency (i.e., the number of k-mers in this color equivalence class). We then assign labels to each color class starting from the class with the largest cardinality, so that the color class represented by the most frequent label will have the shortest label length etc.

The color class representation in Rainbowfish has three components. Rainbowfish stores the mappings between labels and color classes in an ***equivalence class table (ECT)***. As labels are assigned sequentially, this is simply an array of bit vectors encoding the corresponding color sets. Apart from the equivalence class table, Rainbowfish maintains two bit vectors, a ***boundary bit vector (BBV)*** and a ***label bit vector(LBV)***.

All color classes are stored in the equivalence class table (with their corresponding labels implicitly being their position). However, we now need to store a mapping from k-mers to the variable-length labels. Rainbowfish stores variable-length labels corresponding to each k-mer in the label bit vector. The labels are stored in the order in which k-mers are stored in the de Bruijn graph representation. Specifically, the k-mers are stored in the rank order induced by BOSS. However, since these labels are variable-length, we can not directly read the label corresponding to the k-mer of a specific rank, since we do not know where such a label begins or how long it is.

To address this, Rainbowfish maintains another bit vector – the boundary bit vector (BBV) – to mark the boundary of each variable-length label in LBV. The BBV is the same size as the LBV and has a bit set to 1 at each index where a new label starts in the LBV. Thus, the starting position for the label corresponding to the rth k-mer can be obtained by issuing a SELECT(r) query on BBV, and the length of this label can be obtained by simply scanning BBV until we encounter the next set bit.

Figure 1 shows how the color classes are represented in Rainbowfish. To perform a query for the color class corresponding to a k-mer in the colored de Bruijn graph, we first get the rank r of the k-mer in the de Bruijn graph. We then perform a select operation using r on BBV. The result of the select operation i is the start index of the label of the color class in LBV to which the k-mer belongs. To find the length of the label we determine the index i' of the next bit set in BBV using the TZCNT instruction. TZCNT returns the number of trailing zeros in its argument. If B is a 12-bit vector such that $B[0, 11] =$`110010100000` then $\text{TZCNT}(B) = 5$. Using i and i' we retrieve the label from LBV, and using the label we lookup the corresponding color class in ECT. We also note that, as we never have $> 2^{64}$ distinct k-mers in practice, and number of distinct labels is at max equal to the number of distinct k-mers (when each k-mer has a unique label), then we never have $> 2^{64}$ labels. Hence, we can always represent a label using a single machine word. Consequently, we will always reach the next set bit in the LBV after scanning at most a single machine word when starting from current label. This ensures we need only issue a single TZCNT instruction per label decoding call.

3.2 Space analysis

The color class representation in Rainbowfish is entropy compressed, i.e., the space is bounded by the entropy ($H(X_c)$) of the color class distribution. For a dataset in which number of k-mers belonging to each distinct color class are similar, the entropy of the color class distribution will be high. On the other hand, if most of the k-mers in a dataset belong to a small number of distinct color classes, the entropy of the color class distribution will be low.

▶ **Lemma 1.** *The size of each color class label is bounded by $\log_2 M$ bits, where M is the total number of distinct color classes. For a dataset with N distinct k-mers coming from C input samples (i.e., colors), we have that $M \leq \min(N, 2^C)$.*

▶ **Theorem 2.** *Given an ordering of edges (or k-mers) in a de Bruijn graph, the space needed by Rainbowfish to represent a set of colors attached to each edge is $O(MC + NH(X_c))$ bits, where M is the number of distinct color classes, C is the number of colors, N is the number of distinct k-mers, and $H(X_c) = -\sum_{i=1}^{M} P(x_i) \log P(x_i)$ is the entropy (i.e., order-0 or Shannon's entropy) over random variable X_c, which distributed according to the frequency distribution of the color classes.*

Proof. The space needed by Rainbowfish can be analyzed as follows. There are three bit vectors in Rainbowfish, the equivalence class table, label bit vector, and boundary bit vector. To store an equivalence class table containing M distinct color classes each having C colors we need MC bits. To store a label bit vector (as stated in Theorem 1), for N k-mers, where each label corresponds to one of the M distinct color classes, takes $N \log_2 M$ bits. However, as explained in Section 3.1, in Rainbowfish we assign (optimal) variable-length labels based on the frequency of color classes. Therefore, the space needed to store the label bit vector is dependent on the 0th-order entropy of the color class variable, $H(X_c)$, and the size of the label bit vector is upper bounded by $N \log_2 M$. The boundary bit vector has the same number of bits as the label bit vector. ◀

3.3 Lower bound for color representation

We now provide a lower bound to store a color class representation for a set of edges in a colored de Bruijn graph. In the color class representation, the equivalence class table takes MC bits to store M bit vectors each having C bits, which is optimal. The other two bit vectors, the boundary and label bit vector, map k-mers given an ordering in the de Bruijn graph to their corresponding color classes. The theorem below gives the lower bound to store such a mapping.

▶ **Theorem 3.** *The lower bound to represent a mapping from an ordered list of k-mers in a de Bruijn graph to a set of color classes is* $\log_2(M^{N-M} \cdot M!)$ *bits, where M is the number of distinct color classes, N is the number of edges, and for a dataset with N distinct k-mers coming from C input samples (i.e., colors), we have that $M \leq \min(N, 2^C)$.*

Proof. We can analyze the lower bound using a counting argument. We count the number of ways to map a set of M distinct color classes to a set of N edges. The space required to store the color class representation should be less than or equal to the space required to store these mappings.

Edges can be mapped to color classes using a surjective (onto) function. Thus, we wish to count the total number of surjections from M color classes to N edges. Rather than counting this number exactly, we instead provide a lower bound. First, we must ensure that each of the M color classes maps to at least one edge – so, we select a set of M edges and label each with a distinct color class. There are $M!$ ways to assign M color classes to a set of M edges. We will then allow the remaining $N - M$ edges to be colored in any possible manner. We can assign M colors to $N - M$ edges (the remaining number) in M^{N-M} ways. Therefore, the total number of different mappings is bounded below by $M^{N-M} \cdot M!$. To be able to represent each such mapping, and distinguish it from the others, we need at least $\log_2(M^{N-M} \cdot M!)$ bits. ◀

The lower bound can be expanded using Sterling's approximation as

$$(N - M)\log_2 M + M\log_2 M - 0.44M + O(\log_2 M),$$

which, ignoring the additive term $O(\log_2 M)$, is greater or equal to $N\log_2 M - 0.44M$. Given the range of M (i.e., $1 \leq M \leq N$), $N\log_2 M$ always dominates the lower bound.

Now, we show that the space needed by Rainbowfish to store the variable-length labels assigned to color classes is equal to the lower bound. As explained in Theorem 1, the upper bound to store any label is $\log_2 M$ bits, and for N edges, it is given by $N\log_2 M$ bits. Rainbowfish also stores a boundary bit vector which has the same number of bits as the label bit vector. Therefore, the space required to store the label mappings is strictly $\leq 2N\log_2 M$. Note that the extra overhead to store the metadata to perform a select operation in constant time on the boundary bit vector is bounded by $o(N)$, where N is the numbers of bits in the bit vector [9].

However, Rainbowfish's representation of color classes is entropy compressed (see Section 3.1) and the space required depends on the entropy of the color class distribution. For a highly skewed distribution, the entropy is low and the space required to store labels is much smaller than $N\log_2 M$ bits. On the other hand, when the distribution is near-uniform, i.e., the entropy is high, Rainbowfish makes all labels to be $\log_2 M$ bits and dispenses with BBV. Therefore, the space required by Rainbowfish is always smaller than or equal to the lower bound.

3.4 Implementation

Considerations due to the underlying de Bruijn graph representation. We recall here that we make use of the BOSS representation of the underlying de Bruijn graph topology. To build the BOSS representation, k-mer counting is first performed using KMC2 [3], canonicalizing k-mers during counting. Though the BOSS representation inserts both forward and reverse

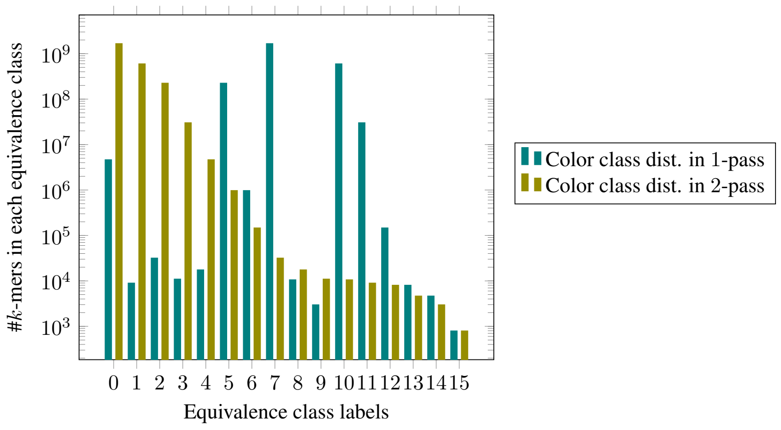


Figure 2 Distribution of k-mer frequencies across equivalence class labels in Rainbowfish after 1-pass and 2-pass algorithm on plant dataset Table 1. The 2-pass algorithm assigns the smallest label to color class with maximum number of k-mers. The distribution in 2-pass algorithm is monotonically decreasing.

complement k-mers into the graph, it associates only a single color vector with this pair. Moreover, BOSS creates "dummy" edges (real k-mers prepended or appended with \$) to allow encoding k-mers that appear near terminal nodes in the de Bruijn graph. In the colored de Bruijn graph these dummy edges are assigned the empty color set. All of this information is encoded by both VARI and Rainbowfish. However, as we discuss in more detail in Section 5, the Rainbowfish representation can work with any de Bruijn graph representation that can assign distinct ranks to each k-mer in the de Bruijn graph. Thus, we would expect this encoding scheme to work well with, e.g., a de Bruijn graph representation based on minimum perfect hashing of the k-mers [4].

Storing bit vectors. In Rainbowfish, we use bit vector implementations from the SDSL library [8] to store the three bit vectors from Figure 1. We use the *rrr_vector* implementation from SDSL to store the equivalence class table and boundary bit vector, and the *bit_vector* implementation from SDSL to store the label bit vector.

The *rrr_vector* of SDSL is an implementation of RRR encoding [20]. RRR encoding is an entropy compressed encoding and also supports constant time rank and select operations on the compressed bit vector. The space reduction depends on the entropy of the bit vector. For high entropy bit vectors, the compression is not noticeable and in fact "negative" in some cases because of the extra metadata overhead to support rank and select operations.

The equivalence class table and boundary bit vector often have fairly low entropy, and can be compressed efficiently using RRR encoding. However, the label bit vector often has high entropy, and compressing it using RRR encoding is not effective. In our representation, the average order-0 entropy of the label bit vector for four different datasets is 0.94. This is a quite high, and hence we did not see any reduction in the space using RRR encoding. However, for the other two bit vectors, the order-0 entropy is lower (e.g., for boundary bit vector the average entropy over same four datasets is 0.56) and, in practice, we achieve a considerable space reduction using RRR encoding.

Construction. We use a 2-pass algorithm to construct the three bit vectors. In the first pass, we read the color matrix, compute the distinct color classes, and count the frequency

Table 1 The number of edges (include k-mers and dummy edges in the BOSS representation), samples and color classes for different datasets used in the experiments. $k = 32$ unless otherwise specified. *# of edges excluding dummies.

Datasets	# of edges	# of colors (samples)	# of distinct color classes
E. coli 10	28,273,951	10	479
E. coli 1000	157,737,064	1000	2,669,157
E. coli 5598	435,705,390	5598	7,000,715
E. coli 1000 (k=63)	258,893,268	1000	2,530,253
Plant	2,520,140,426	4	16
Beef safety	97,096,576,010*	87	623,022,532
Human transcriptome	159,441,804*	95,146	340,762

of each class. Once we have the frequency information, we sort color classes in descending order based on their frequency. We then assign labels to color classes starting from zero. In the second pass, we read the uncompressed color matrix again, and add the label of each k-mer to the label bit vector. While building the label bit vector, we also build the boundary bit vector by storing a 1 at every index where a new label starts in the label bit vector. The labels are stored in the same order as the k-mers in the BOSS representation.

To reduce the space required for the labeling even further, we implemented our label encoding in the following way. Every time that the label size increases from x bits to $x + 1$ bits, we restart the counter of that label in label bit vector to 0. For example, we store 0 and 1 for labels 0 and 1 respectively, then we store 00, 01, 10 and 11 for labels 2, 3, 4 and 5 respectively. For label value 6 we again restart the counter to 0 and store 000 to represent 6 in the label bit vector, etc. Later, when we want to retrieve the actual value of a label, we first recover the stored label l' from the label bit vector and then calculate the actual label l using the equation $l = l' + 2^d - 2$ where d is length of label l in bits.

As explained in Section 3.2, the 2-pass algorithm minimizes the space used to represent color class labels by sorting the classes based on their frequencies and assigning labels to color classes to minimize the length of the resulting code path, similar to Huffman coding. However, one could also imagine assigning labels to color classes as we see them in the order k-mers appear in the BOSS representation. This way, we can construct all three tables in a single pass (i.e., a 1-pass algorithm).

However, as shown in Figure 2, this 1-pass algorithm can end up assigning long labels to frequent k-mers, and hence produce poor (i.e., large) encodings. However, the 2-pass algorithm always assigns labels according to the corresponding frequency distribution of the color classes. Sometimes, the 1-pass algorithm does well, but we chose to adopt the 2-pass algorithm in Rainbowfish.

4 Evaluation

In this section we evaluate Rainbowfish, and compare it to VARI [15], a state-of-the-art colored de Bruijn graph representation. We evaluate both systems in terms of space and running time. We address the following questions about the performance of Rainbowfish: How does Rainbowfish compare to VARI in terms of the space required to represent color information?; How does Rainbowfish compare to VARI in terms of the construction time?; How does Rainbowfish compare to VARI in terms of typical queries (e.g., in bubble calling)? We are particularly concerned with ensuring that Rainbowfish produces small encodings of the color information and remains practically efficient to query.

Table 2 Construction and bubble calling time for Rainbowfish and VARI for different datasets.

Datasets	Construction Time (secs)		Bubble Calling Time (secs)	
	VARI	Rainbowfish	VARI	Rainbowfish
E. coli 10	44	31	344	366
E. coli 1000	340	270	2,610	2,356
E. coli 5598	3,141	4,021	8,796	8,201
Plant	108	339	47,040	48,537
Beef safety	15,378	30,478	NA	NA
Human transcriptome	13,961	30,804	NA	NA

4.1 Experimental setup

To answer the above questions, we perform two different benchmarks. First, we evaluate the time taken to construct the color class representation. The construction time is the time taken to construct the color class representation from a list of color classes stored in the order of the edges in the de Bruijn graph (this is the same input used by VARI). During construction, we adopt a two-pass algorithm. In the first pass, we use a sparse hash-table to determine the distinct color classes and the cardinality of each such class.

We note that the space taken in this first pass is within a small constant factor of the final space required by the final ECT table itself, since we need only store each color class once in the hash table (as a key), and store the associated count (a machine word) as the value. Thus, the memory required by this first pass is almost always a small fraction of the total memory usage of the construction algorithm.

Given this information, we know exactly the number of bits that will be required to store the label and boundary vectors. In the second pass, we fill in both the label and boundary vectors and then save all three structures to file. As with most succinct representations, the space required for our data structure in memory and on disk is almost the same (as the two-pass algorithm allows us to allocate only the space we need for our final representation). The construction time recorded here does not include (for either Rainbowfish or VARI) the time taken to build the de Bruijn graph and color list corresponding to edges in the de Bruijn graph (since this is the same for both methods).

We also report the space needed by both Rainbowfish and VARI to store the color class representation on disk. We do not include the space needed to represent the actual de Bruijn graph in our space comparisons because both Rainbowfish and VARI use BOSS to store the actual de Bruijn graph, and the BOSS representation itself tends to take less space than the color information.

Second, we evaluate the time taken to perform the bubble calling benchmark as described in [16], using both the VARI and Rainbowfish representations. Finding bubbles in a colored de Bruijn graph enables one to detect regions in the de Bruijn graph where different samples (i.e., colors) diverge from each other. As originally suggested by Iqbal et al. [12], such algorithms can form the basis for analyzing certain types of genetic variants in populations of genomes. We note that we adopt the exact bubble calling algorithm implemented in VARI, and the only variable being altered in our bubble-calling benchmark is the data structure being used to determine the set of colors present for each k-mer. Since VARI and Rainbowfish are both built upon the BOSS representation, which is based on the edge-centric view of de Bruijn graph, they consider k-mers as edges in the de Bruijn graph, meaning that each edge is associated with a k-mer, and its corresponding rank and color set. Briefly, the bubble calling algorithm takes as input a pair c_1, c_2 of colors and traverses edges in the de Bruijn graph to find bubbles in which the edges in one sub-path are colored with c_1 and the edges in the other sub-path are colored with c_2 (see [16] for further details).

For all experiments in this paper, unless otherwise noted, we consider the k-mer size to be 32 to match the parameters adopted by Muggli et al. [16]. We carry out these benchmarks on a number of datasets as described in Section 4.2. The time reported for construction and bubble calling are averaged over two runs, and the time is measured as the wall-clock time using the `/usr/bin/time` executable. All experiments were performed on an Intel(R) Xeon(R) CPU (E5-2699 v4 @2.20GHz with 44 cores and 56MB L3 cache) with 512GB RAM and a 4TB TOSHIBA MG03ACA4 ATA HDD running ubuntu 16.10, and were carried out using a single thread. We note that, while the construction of the color set representation in Rainbowfish (and VARI) are serial operations, queries are trivially parallelizable, as each label can be queried and decoded independently.

4.2 Data

We run our benchmarks on the datasets mentioned in Table 1. The first three datasets, *E. coli*, Plant, and Beef safety are slight variants of those used for evaluation in VARI [16]. Each of these data sets exhibits different characteristics in terms of the number of k-mers, the number of input samples (i.e., colors) and the homogeneity of the underlying samples (i.e., how different are the de Bruijn graph for each of the individual samples). The first dataset consists of the assemblies of 5,598 different strains of *E. coli* obtained from GenBank [18]. Here, each “color” represents a specific *E. coli* assembly. Since these assemblies are from different strains of the same species, they exhibit a small degree of heterogeneity. In other words, a large fraction of the union de Bruijn graph is expected to occur in all samples.

To evaluate the scalability of Rainbowfish when primarily changing the underlying number of input colors, we have evaluated three variants of the *E. coli* dataset. These consist of a dataset containing only 10 different strains, another containing 1,000 different strains and the final containing all 5,598 strains. We also performed experiments with k-mer size to be 63 for *E. coli* 1000 dataset to evaluate the space usage for higher k-mer sizes.

The second dataset (i.e., Plant) consists of the genome assemblies of four different plant species. Hence, this dataset contains only four colors, but has more than ≈ 2 billion distinct k-mers. The plant species considered are, *A. thaliana* [1] [22], Corn[2] [21], Rice[3] [23], and Tomato[4] [2]. These genomes exhibit considerable diversity and heterogeneity. Given the diverse regions in the colored de Bruijn graph, this dataset is a good candidate for the bubble calling benchmark. Further, Muggli et al. [16] found that this was the only of the three original datasets on which they were able to construct the original `Cortex` representation of the colored de Bruijn graph. They validated `Cortex` produces the same bubble calls as VARI [16] (which, of course, produces the same bubble calls as Rainbowfish). For more detailed analysis of `Cortex`'s construction and processing time and space on this dataset, please refer to [16].

The third dataset, Beef safety, is considerably different from the prior data. Instead of the input samples consisting of assembled genomes, they consist of 87 metagenomic samples

[1] `ftp://ftp.ensemblgenomes.org/pub/plants/release-34/fasta/arabidopsis_thaliana/dna/Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.gz`
[2] `ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/005/005/GCF_000005005.1_B73_RefGen_v3/GCF_000005005.1_B73_RefGen_v3_genomic.fna.gz`
[3] `http://rice.plantbiology.msu.edu/pub/data/Eukaryotic_Projects/o_sativa/annotation_dbs/pseudomolecules/version_7.0/all.dir/all.con`
[4] `ftp://ftp.solgenomics.net/tomato_genome/assembly/build_2.50/SL2.50ch00.fa.tar.gz`

sequenced from cattle in the commercial process of beef production [17]. Hence, this dataset yields a considerably larger and more complex de Bruijn graph since it is built upon many un-assembled (and non-error-corrected) reads. Thus, the de Bruijn graph will encode portions of the relevant metagenomes as well as the effects of sequencing errors. This dataset also has many more k-mers than the others, ≈ 97 billion. It exhibits a large degree of heterogeneity and an intermediate number of input colors (87).

In addition to the three datasets used in the VARI paper, we also consider building the colored de Bruijn graph on the human transcriptome[5] (Gencode v26 protein coding transcripts) [10]. Here, we consider each transcript as an individual sample (i.e., a distinct input color). This data consists of $\approx 95,000$ colors, but only ≈ 159 million k-mers. Hence, this dataset will give an idea about how the representations will perform when the number of colors becomes very large (though the number of distinct color classes remains orders of magnitude smaller than the number of k-mers). Further, we note that this dataset highlights some of the similarities between the color class encoding adopted by Rainbowfish and the k-mer-based equivalence class decomposition adopted by certain transcript quantification methods (e.g. [19]).

4.3 Performance

Table 2 shows the time taken by Rainbowfish and VARI to construct the color class representation for different datasets. Rainbowfish uses a 2-pass algorithm to construct the color class representation, and hence the construction time is dominated by the steps to read the color list file twice. For small datasets like *E. coli* 10 and *E. coli* 1,000, the input file size is small and does not affect the overall construction time compared to VARI. However, for large datasets like Plant and Beef safety, the time to read the color file twice dominates the construction time and Rainbowfish is $1.9\times$–$3\times$ slower. We note that this time can be considerably reduced by avoiding the uncompressed color matrix representation currently used upstream of Rainbowfish and VARI, and integrating determination and encoding of the color classes into the de Bruijn graph construction directly. However, this is outside the scope of the current paper.

Space

Table 3 shows the space usage of Rainbowfish and VARI for the different datasets we consider. Among these data, there are a range of characteristics in terms of the number of k-mers, the number of colors, and the complexity and heterogeneity of the de Bruijn graph. We find that, for all datasets, Rainbowfish requires less space to store the color information than VARI. The magnitude of the improvement depends on the number of distinct equivalence classes and their distribution, but is as large as $\sim 20\times$. We see the same trend with higher values of k-mer sizes.

In particular, Rainbowfish's space usage is particularly impressive for datasets with a large number of input colors but a relatively small number of distinct k-mers. In this case, we usually find that the number of distinct color classes is very small compared to the universe of possibilities, and so each label can be encoded in much fewer than C bits. However, the space VARI consumes depends greatly on the sparsity of the color matrix. The color matrix itself grows rapidly as the number of k-mers and colors increases, but VARI's compression

[5] `ftp://ftp.sanger.ac.uk/pub/gencode/Gencode_human/release_26/gencode.v26.pc_transcripts.fa.gz`

Table 3 The space required by Rainbowfish and VARI to store the color class representation for different datasets. The first column shows space required for the uncompressed color matrix ($N \times C$ bits). All space is reported in MB. $k = 32$ unless otherwise specified.

Datasets	uncompressed color matrix	VARI	Rainbowfish
E. coli 10	34	58	20
E. coli 1000	18,804	8,848	475
E. coli 5598	290,761	58,718	2,938
E. coli 1000 (k=63)	185,669	8,872	637
Plant	1,202	1,603	497
Beef safety	1,007,009	210,998	144,564
Human transcriptome	1,808,435	841	817

mechanism (Elias-Fano encoding) is very effective if the color matrix is sparse (e.g., each k-mer is labeled with only a small subset of colors). This is exactly the case for the Human transcriptome, where the color matrix has an entropy of ~ 0.0004 (compared to *E. coli* 5,598 and *E. coli* 1,000 with entropies of ~ 0.16 and ~ 0.34 respectively). Thus, in the *E. coli* dataset, VARI can save space up to a factor of ~ 5 compared with the uncompressed representation, while in the Human transcriptome it can save a factor of $\sim 2,150$ because of the low entropy of the color matrix. Rainbowfish does well in all experiments, even when the number of input colors is small (e.g., in the Plant dataset). Rainbowfish achieves the most impressive compression when the color class distribution has low entropy and the number of color classes is small relative to the upper bound. In such cases, the entropy compressed representation of Rainbowfish is able to represent a large fraction of all labels using a very small number of bits.

Bubble calling

Table 2 shows the time taken by Rainbowfish and VARI to perform the bubble calling benchmark on different datasets. We run the bubble calling benchmark on the *E. coli* and Plant datasets (as in the VARI paper). We note that the current bubble calling algorithm is too slow to run on the Beef safety data set (the time in [16] was estimated at $> 3,000$ hours). It is possible, however, that optimizations to the underlying algorithm might lift this restriction. We also did not perform bubble calling on the human transcriptome dataset as here, we were unable, given the resources on our server, to even run the de Bruijn graph construction to completion. Specifically, due to the large amount of external memory that VARI uses to build the uncompressed color matrix and the de Bruijn graph on these larger (either in terms of the number of k-mers, the number of colors, or both) datasets (on order of Terabytes), we exhausted the available disk space. For these datasets, to approximate the relevant sizes and construction times, we produced a uncompressed color matrix that lists the colors for each k-mer and its reverse complement, and we use this to build both the VARI and Rainbowfish color representations. While very similar to the full color matrix that VARI would produce, this file is slightly different in that it does not include entries for dummy edges (a detail of the BOSS representation), and the order of the color matrix rows can be different from what will appear in the BOSS representation. However, we still believe these numbers, provided in Table 1, give a reasonable approximation of how the respective methods would perform were we able to construct the de Bruijn graph completely.

For bubble calling, both representations require a very similar amount of time. This is likely due, in part, to the fact that navigating the BOSS representation of the de Bruijn graph may be the performance bottleneck in the bubble calling algorithm. Thus, both VARI and Rainbowfish provide sufficiently fast access to the color sets for each edge that they do not represent bottlenecks in this regard.

5 Conclusion and Future Work

In this paper, we propose an entropy-compressed, succinct data structure to store the color information of a colored de Bruijn graph. To represent the topology of the de Bruijn graph itself, we adopt the BOSS [1] representation. However, we note that, for our representation of the color sets, we only require that the underlying de Bruijn graph representation is able to associate a unique rank between 0 and $N-1$ with each edge. Hence, it is possible to use the Rainbowfish representation with other representations of the de Bruijn graph topology (e.g., those based on minimal perfect hashing).

We demonstrate that the inherent skewness in the distribution of color classes can be exploited to reduce the size of the color information. This allows Rainbowfish to represent the colored de Bruijn graph, even for large datasets with many colors, in a reasonably small space. In fact, for representing the color information itself, we show that Rainbowfish is succinct, and hence requires only $Z + o(z)$ bits where Z is the number of bits required by an information-theoretically optimal representation. Moreover, it may be possible for the color information stored in the equivalence class table to be further compressed to reduce the space. For example, one could imagine an encoding of color sets that takes advantage of their shared subsets, e.g., storing the shared prefixes of membership vectors only once.

While we have described here a system for efficiently representing the color information in a colored de Bruijn graph, our encoding scheme can be generalized to store any type of attribute attached to the edges. For example, one could use the same (or a related) scheme to encode information like the k-mer count or set of positions associated with a given edge. Moreover, it will be interesting to explore how multiple attributes could be efficiently stored simultaneously, and how potential correlations between these attributes might be exploited. For example, there may be natural extensions of similar coding schemes to the *compacted* de Bruijn graph, where one might also be able to take advantage of the coherence in annotation (i.e., color or count information) shared among the constiuent k-mers of a contig, allowing one to store only the information where these annotations change during traversal.

Finally, in our current implementation, the input to the system is a color matrix file generated by VARI. This implementation requires first building the uncompressed color matrix, and then permuting the rows of this matrix along with the edges of the de Bruijn graph during the BOSS construction procedure. This process can require a large amount of space, as the uncompressed color matrix can become extremely large (on the order of Terabytes for some of the datasets we considered here). Consequently, in most cases, the construction algorithm must resort to making extensive use of external memory (i.e., disk), which increases building time and consumes a large amount of disk space. However, we note that the Rainbowfish representation can be built without direct access to the uncompressed color matrix.

Specifically, the current VARI algorithm uses a mergesort-like approach to construct the uncompressed color matrix, where the k-mers in each sample are sorted lexicographically (independently), and the rows of the color matrix are constructed one by one by asking for each k-mer, in lexicographic order, which samples contain it. The working memory of this approach is very small compared to the size of the full color matrix itself. One could imagine using the same merge-based scheme to construct the Rainbowfish representation directly. In the first pass, the distinct color classes and a counter for each would be stored, resulting in a small, sparse hash table rather than a large, uncompressed color matrix. In the second pass, one would simply associate the relevant labels, rather than uncompressed color vectors, with each edge. This would vastly reduce the time and space required to construct the colored de Bruijn graph.

Thus, in the future, we are interested in both incorporating the Rainbowfish representation more tightly inside the existing VARI codebase, as well as pairing the Rainbowfish representation with other compatible representations of the de Bruijn graph topology.

Acknowledgments. We also thank Michael Bender and Robert Johnson for fruitful conversations and important insight when performing this research. We would also like to thank Rayan Chikhi for suggesting using auxiliary, rank/index-based storage for maintaining and accessing annotations for each edge in a de Bruijn graph, and for useful feedback on an early version of this manuscript. Finally, we would like to thank the anonymous reviewers for constructive feedback.

References

1 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *Proceedings of the International Workshop on Algorithms in Bioinformatics*, pages 225–235. Springer, 2012.

2 Mathilde Causse, Nelly Desplat, Laura Pascual, Marie-Christine Le Paslier, Christopher Sauvage, Guillaume Bauchet, Aurélie Bérard, Rémi Bounon, Maria Tchoumakov, Dominique Brunel, et al. Whole genome resequencing in tomato reveals variation associated with introgression and breeding events. *BMC genomics*, 14(1):791, 2013.

3 Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. KMC 2: Fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015.

4 Erwan Drezen, Guillaume Rizk, Rayan Chikhi, Charles Deltel, Claire Lemaitre, Pierre Peterlongo, and Dominique Lavenier. Gatb: Genome assembly & analysis tool box. *Bioinformatics*, 30(20):2959–2961, 2014.

5 Peter Elias. Efficient storage and retrieval by content and address of static files. *Journal of the ACM (JACM)*, 21(2):246–260, 1974.

6 Robert Mario Fano. *On the number of bits required to implement an associative memory.* Massachusetts Institute of Technology, Project MAC, 1971.

7 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

8 Simon Gog. Succinct data structure library. `https://github.com/simongog/sdsl-lite`, 2017. [online; accessed 01-Feb-2017].

9 Rodrigo González, Szymon Grabowski, Veli Mäkinen, and Gonzalo Navarro. Practical implementation of rank and select queries. In *Poster Proceedings Volume of 4th Workshop on Efficient and Experimental Algorithms (WEA)*, pages 27–38, 2005.

10 J. Harrow, A. Frankish, J.M. Gonzalez, E. Tapanari, M. Diekhans, F. Kokocinski, B.L. Aken, D. Barrell, A. Zadissa, S. Searle, I. Barnes, A. Bignell, V. Boychenko, T. Hunt, M. Kay, G. Mukherjee, J. Rajan, G. Despacio-Reyes, G. Saunders, C. Steward, R. Harte, M. Lin, C. Howald, A. Tanzer, T. Derrien, J. Chrast, N. Walters, S. Balasubramanian, B. Pei, M. Tress, J. M. Rodriguez, I. Ezkurdia, J. van Baren, M. Brent, D. Haussler, M. Kellis, A. Valencia, A. Reymond, M. Gerstein, R. Guigo, and T. J. Hubbard. GENCODE: The reference human genome annotation for the ENCODE project. *Genome Research*, 22(9):1760–1774, sep 2012. `doi:10.1101/gr.135350.111`.

11 Guillaume Holley, Roland Wittler, and Jens Stoye. Bloom filter trie: an alignment-free and reference-free data structure for pan-genome storage. *Algorithms Mol. Biol.*, 11:3, 2016.

12 Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.

13 Guy Jacobson. Space-efficient static trees and graphs. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 549–554. IEEE, 1989.

14 Guy Joseph Jacobson. *Succinct Static Data Structures.* phdthesis, Carnegie Mellon University, 1988. AAI8918056.

15 Martin D. Muggli. Vari. `https://github.com/cosmo-team/cosmo/tree/VARI`, 02 2017. Viewed Feb 3, 2017.

16 Martin D. Muggli, Alexander Bowe, Noelle R. Noyes, Paul Morley, Keith Belk, Robert Raymond, Travis Gagie, Simon J. Puglisi, and Christina Boucher. Succinct Colored de Bruijn Graphs. *Bioinformatics*, 2017.

17 Noelle R. Noyes, Xiang Yang, Lyndsey M. Linke, Roberta J. Magnuson, Adam Dettenwanger, Shaun Cook, Ifigenia Geornaras, Dale E. Woerner, Sheryl P. Gow, Tim A. McAllister, et al. Resistome diversity in cattle and the environment decreases during beef production. *ELife*, 5:e13195, 2016.

18 Nuala A. O'Leary, Mathew W. Wright, J. Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, et al. Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, pages D733–D745, 2015. `doi:10.1093/nar/gkv1189`.

19 Rob Patro, Stephen M. Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature biotechnology*, 32(5):462–464, 2014.

20 Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 233–242. Society for Industrial and Applied Mathematics, 2002.

21 Patrick S. Schnable, Doreen Ware, Robert S. Fulton, Joshua C. Stein, Fusheng Wei, Shiran Pasternak, Chengzhi Liang, Jianwei Zhang, Lucinda Fulton, Tina A. Graves, et al. The b73 maize genome: complexity, diversity, and dynamics. *science*, 326(5956):1112–1115, 2009.

22 David Swarbreck, Christopher Wilks, Philippe Lamesch, Tanya Z. Berardini, Margarita Garcia-Hernandez, Hartmut Foerster, Donghui Li, Tom Meyer, Robert Muller, Larry Ploetz, et al. The arabidopsis information resource (tair): gene structure and function annotation. *Nucleic acids research*, 36(suppl 1):D1009–D1014, 2008.

23 Tsuyoshi Tanaka, Baltazar A. Antonio, Shoshi Kikuchi, Takashi Matsumoto, Yoshiaki Nagamura, Hisataka Numa, Hiroaki Sakai, Jianzhong Wu, Takeshi Itoh, Takuji Sasaki, et al. The rice annotation project database (rap-db): 2008 update. *Nucleic Acids Research*, 36(Supp 1):D1028–D1033, 2008.

ThIEF: Finding Genome-wide Trajectories of Epigenetics Marks*

Anton Polishko[1], **Md. Abid Hasan**[2], **Weihua Pan**[3], **Evelien M. Bunnik**[4], **Karine Le Roch**[5], **and Stefano Lonardi**[6]

1 **Department of Computer Science, University of California, Riverside, CA, USA**

2 **Department of Computer Science, University of California, Riverside, CA, USA**

3 **Department of Computer Science, University of California, Riverside, CA, USA**

4 **University of Texas Health Science Center, San Antonio, TX, USA**

5 **Department of Cell Biology, University of California, Riverside CA, USA**

6 **Department of Computer Science, University of California, Riverside, CA, USA**

Abstract

We address the problem of comparing multiple genome-wide maps representing nucleosome positions or specific histone marks. These maps can originate from the comparative analysis of ChIP-Seq/MNase-Seq/FAIRE-Seq data for different cell types/tissues or multiple time points. The input to the problem is a set of maps, each of which is a list of genomics locations for nucleosomes or histone marks. The output is an alignment of nucleosomes/histone marks across time points (that we call *trajectories*), allowing small movements and gaps in some of the maps. We present a tool called ThIEF (TrackIng of Epigenetic Features) that can efficiently compute these trajectories. ThIEF comes into two "flavors": ThIEF:Iterative finds the trajectories progressively using bipartite matching, while ThIEF:LP solves a k-partite matching problem on a hyper graph using linear programming. ThIEF:LP is guaranteed to find the optimal solution, but it is slower than ThIEF:Iterative. We demonstrate the utility of ThIEF by providing an example of applications on the analysis of temporal nucleosome maps for the human malaria parasite. As a surprisingly remarkable result, we show that the output of ThIEF can be used to produce a supervised classifier that can accurately predict the position of stable nucleosomes (i.e., nucleosomes present in all time points) and unstable nucleosomes (i.e., present in at most half of the time points) from the primary DNA sequence. To the best of our knowledge, this is the first result on the prediction of the dynamics of nucleosomes solely based on their DNA binding preference. Software is available at `https://github.com/ucrbioinfo/ThIEF`

1998 ACM Subject Classification G.2.1 Combinatorics, I.1.2 Algorithms, J.3 Life and Medical Sciences

Keywords and phrases Nucleosomes, Histone Marks, Histone Tail Modifications, Epigenetics, Genomics

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.19

* This work was partially supported by NSF IIS-1302134 to SL and KLR and NSF III-1526742 to SL.

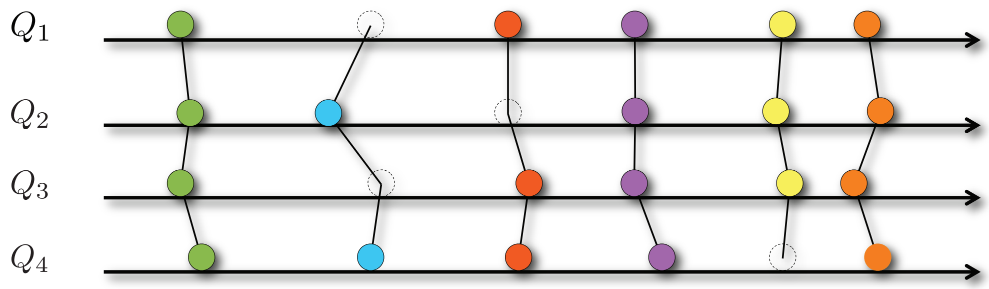


Figure 1 An illustration of the problem of aligning epigenetic features on four maps Q_1, Q_2, Q_3, Q_4; the input is a set of locations for the feature of interest (e.g., CHiP-seq peaks for nucleosome or specific histone marks) illustrated as circles here; the output is an assignment of features to trajectories, in a way that most parsimoniously explain the data; dotted circles indicate gaps or "missing features"

1 Introduction

Advancements in high-throughput DNA sequencing technology has enabled life scientists to carry out increasingly large-scale experiments. In genomics and epigenetics, it is relatively common to run multiple genome-wide experiments: for example, one can use ChIP-Seq/MNase-Seq/FAIRE-Seq/NOMe-Seq to obtain nucleosome levels or specific histone tail post-translational modifications (e.g., methylation, acetylation, phosphorylation, ubiquitination) at different cell types/tissues (which allows to understand cell type-specific marks), or at different time points for a particular cell process cycle (which allows to explore the dynamics of epigenetic marks). The ENCODE project [8], modENCODE [10, 41], phychENCODE [1], and the NIH Roadmap Epigenomics Project [2] are notable examples of these efforts. Similar comparative analysis of epigenetic marks could be carried out for a set of closely related organisms (e.g., human-chimp) if the correspondences between the genomes are known (e.g., using liftover by [20]).

Epigenetic maps are usually obtained by (i) sequencing a DNA sample enriched for a feature of interest, (ii) mapping short reads to the reference genome and (iii) running a peak-calling algorithm (e.g., MACS/MACS2 [50], NOrMAL [37], Puffin [36]). In this paper we focus on the fundamental question on how to compare multiple genome-wide epigenetic maps, when features are expected to shift or be missing. Our model allows the possibility of the nucleosome of physically sliding along the genome or compensate for the noise in the peak detection process. In the example in Figure 1, the objective is to align four maps. Circles represent features to align; dashed circle mark the gap; trajectories are indicated with solid lines, which represent the most "likely" explanation of the data.

We propose to compare epigenetic maps by aligning them in a similar way we align DNA sequences. We arrange multiple nucleosome/feature maps on top of each other, with the objective to build a *trajectory* for each individual nucleosomes/feature across time points or cell types, in a way that a total cost (i.e., total "traveled distance" in the case of nucleosomes) is minimized. We call such a set of trajectories an *alignment*: similarly to multiple sequence alignment we allow "insertion" or "deletions" of nucleosomes/features at specific time points or cell types. For instance, Figure 2 illustrates the output of THIEF in the IGV browser for a region of chromosome 10 of *P. falciparum*. THIEF aligned eight nucleosome maps over multiple time points, where each trajectory is assigned a unique ID, using alternating colors. Observe that in some trajectories (e.g., the one with ID 10_4546), nucleosomes are stable, while in others (e.g., the one with ID 10_4547) nucleosomes are evicted then bind later to

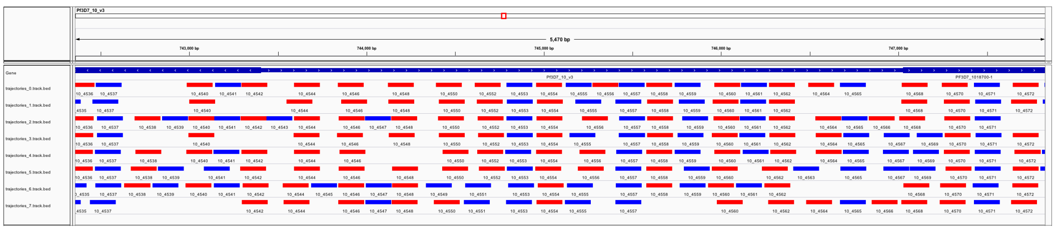


Figure 2 IGV snapshot of nucleosomes (red and blue rectangles – 147bps long) in region [742,356-747,829] of chromosome 10 of *P. falciparum* at eight time points; ThIEF assigns nucleosomes to trajectories which can be identified by an integer ID (zoom in to see them), and displayed in alternating colors

the same location. We should note that it not easy to tag individual nucleosomes and image them under microscopes (see, e.g., [47]), so there is no way with current technology to obtain the "true" trajectories of nucleosomes genome-wide.

2 Problem definition

We define a *epigenetic map* $Q = \{f_1, \ldots, f_n\}$ as a set of n epigenetic features f_i, where a feature could be a nucleosome or a specific histone mark. Each feature $f \in Q$ is described by vector $f = (\mu, a_1, \ldots, a_l)$ where μ is the genomic coordinate of f in the genome (e.g., chromosome number and position in the chromosome) and each a_j, $j = 1, \ldots, l$ is an *attribute* of that feature (e.g., confidence score of a nucleosome, strength of a histone mark, p-value, peak width, "fuzziness" of a nucleosome, etc.). For convenience, we assume that features in Q are ordered by their position μ.

Given k epigenetic maps $Q_1, Q_2, \ldots, Q_k$, the goal is to align them so that the total alignment cost is minimized. For instance when $k = 2$, we either match feature $f \in Q_1$ to a feature $g \in Q_2$ so that we minimize a cost $\Delta(f, g)$ (e.g., some distance between the vectors $f = (\mu^f, a_1^f, \ldots, a_l^f)$ and $g = (\mu^g, a_1^g, \ldots, a_l^g)$) or report that feature $f \in Q_1$ has no match in Q_2 (insertion/deletion). In this latter case we will say that we have an alignment *gap*. For k maps, the cost function is $\Delta(q_1, q_2, \ldots, q_k)$ where q_i is a feature in the i-th map. Note that $\Delta(q_1, q_2, \ldots, q_k)$ is supposed to be defined for all combinations of $q_1, q_2, \ldots, q_k$.

While our framework allows features with $l \geq 0$ attributes, in the rest of the paper we only use the genomic coordinate (i.e., $l = 0$). In that case $\Delta(q_1, q_2, \ldots, q_k)$ is simply the *absolute deviation*, i.e., the sum of absolute distance between the coordinate of each feature and the mean of those k coordinates. As it is done in sequence alignment, we will not allow swaps in the order of genomic features. In other words, we do not allow trajectories to cross each other.

3 Previous work

The comparative analysis of a set of genome-wide epigenetic maps can be challenging, in particular when the number of maps is large. Some bioinformatic tools are available to help users make sense of the data. For instance, BEDTools allows users to compares multiple maps by determining which peaks are overlapping, non-overlapping, or have a minimal fraction of overlap [40]; Galaxy [12, 3, 11] offers a set tools for working with genomic intervals under the "Operate on Genomic Intervals" section; ChromHMM can be also used to summarize multiple epigenetic maps. In ChromHMM, a multivariate hidden Markov Model allows users to determine *chromatin states* (e.g., active enhancer, heterochromatin, repressed PolyComb, etc.) from multiple histone marks, DNA methylation, DNase I hypersensitive sites, and gene expression [9]. Similarly, Segway uses a dynamic Bayesian network model to identify chromatin patterns associated with transcription start sites, gene ends, enhancers, transcriptional regulator CTCF-binding regions and repressed regions [14, 15]. Recently, EpiCSeg was introduced to address shortcomings of ChromHMM and Segway [30]. In terms of motivations, DGW is the closest work to ours. DGW uses dynamic time warping to align the profiles of ChIP-seq enrichment and to cluster them into groups which share similar shapes [28].

The problem of aligning genomic features from multiple maps is similar to the *multiple sequence alignment* (MSA) problem (see, e.g., [6]). MSA is a central problem in bioinformatics with a wide range of applications (see, e.g., [42, 39, 23, 31]). A very large corpus of literature has been published on MSA and its applications. A direct solution for MSA uses dynamic programming to identify the globally optimal alignment [44, 32]. The majority of efficient methods employ sophisticated heuristics, since the global optimization problem of aligning long sequences is computationally costly (the problem is *NP*-complete [7, 49, 18]). Heuristic methods include progressive alignment construction [13, 46, 45, 25], iterative methods, hidden Markov models, genetic algorithms [34, 33], simulated annealing [21, 16], among others. The major difference between MSA on DNA/proteins and the problem discussed here is that instead of a substitution matrix, we use a cost function Δ defined between all possible combinations of features in the k maps.

The problem of aligning genomic features is also related to the *multi-target tracking* problem or *data association* problem. These latter problems have been known for decades and are extensively studied. These problems arise when there is a need to track features on video sequences, radar scans, etc. The main computational challenge is to overcome scalability when dealing with a large number of snapshots and a relatively small number of objects to track. In our case, we are interested in dealing with a relatively small number of maps and a large number of objects.

4 Methods

We first describe the naïve (greedy) approach and explain its limitations, then we provide two improved algorithms. The first builds iterative approximations of the optimal alignment, the second computes the solution via integer linear programming [4]. To make the second approach feasible to real-world-sized data we applied branch-and-bound technique to reduce the size of the problem. Then, we relax the problem to a linear program, which we solve using the off-the-shelf solver GLPK [29].

4.1 Naïve (greedy) approach

In the naïve algorithm, we consider each feature (at location μ) in the first map and check whether we can match it to features on the other maps such that all these features are located within a window of predefined length w, centered at μ. If a map does not have a feature in the window $[\mu - w/2, \mu + w/2]$, we introduce a gap. When a feature is assigned to a trajectory, we mark it so that it cannot be used for other trajectories. After considering all the features in the first map, we check whether the next map contains any feature that was not marked; if any of the feature is still unmatched the same sliding window approach is employed. We repeat this process until all the features are marked.

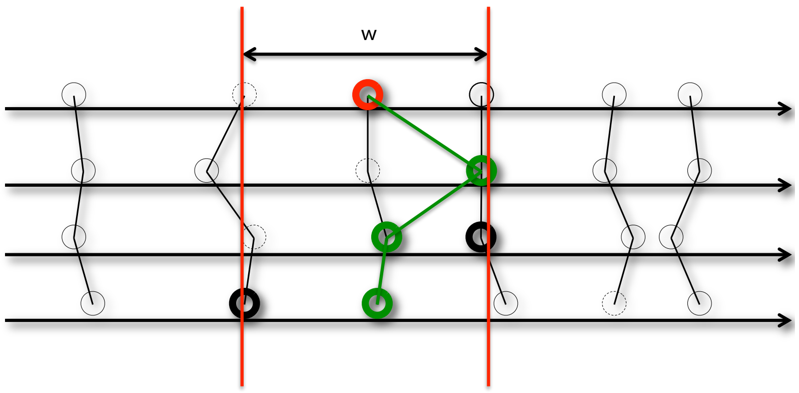


Figure 3 The naïve (greedy) method can create sub-optimal alignments (see text).

The naïve algorithm is fast, but it does not guarantee to produce an optimal solution. First, it relies on the assumption that the alignment score of three or more features is decomposable into the combination of pairwise alignments. Second, the algorithm can make wrong "choices" depending on window size w. Figure 3 illustrates an example of what could go wrong. Circles represent features to align, where a solid circle means that a feature is present, dashed circle means that a feature is missing. The black lines represent the trajectories that we are expected to recover. When the naïve algorithm considers the "red" feature, it will to match it with features marked with thick-line circles. The resulting alignment is shown in green, which is sub-optimal globally.

4.2 ThIEF:Iterative

ThIEF:Iterative iteratively constructs alignments by processing input maps pairwise. The algorithm starts by aligning the first two maps by solving the *weighted bipartite matching* problem. Then, ThIEF:Iterative aligns the resulting alignment between the first two maps to the third map. At each iteration the algorithm solves a bipartite matching problem between the current alignment and the next epigenetic map.

Each node in the bipartite graph is assigned a genomic location. When a genomic feature f is mapped to a node $v_f \in V$, then its location is μ_f. Since the algorithm needs to align alignment to maps, we use the average of the coordinates of the features that belong to an alignment as the "location" of that alignments. In this way, nodes corresponding to partial alignments will have a location attribute and the cost function can be evaluated. Each edge (w, v) is assigned weight $|\mu_w - \mu_v|$.

The bipartite graph must have an equal number of vertices in each feature map (partition), so for every vertex in one partition we introduce a "dummy" vertex in the other one. The presence of dummy nodes also naturally allows gaps: when a feature is matched against a dummy node we are implicitly introducing a gap. The cost of edges connecting "dummy" node to features is the gap penalty δ, and edges "dummy"-"dummy" are not allowed. To solve the $n - to - n$ assignment problem we use the *Hungarian algorithm* [17], which has $\Theta(n^3)$ time complexity.

The total running time of this approach is $\Theta(km^3)$, where k is the number of maps and m is an upper bound on the number of alignments. In the worst case, if we align maps such that features on each map are aligned with corresponding gaps then we could have $m \in O(2^{k-1}n)$. The other disadvantage of this approach is that it does not guarantee optimality, unless the cost function Δ is decomposable into sum of pairwise costs. In general, the order in which maps are processed can give rise to different (sub-optimal) solutions.

4.3 ThIEF:LP

ThIEF:LP casts the alignment problem as a *weighted k-partite matching* problem. Our problem is slightly more general than the k-partite matching problem because we need to deal with gaps. First ThIEF:LP builds a hyper-graph $H = (V, E)$, where each vertex $v \in V$ represent a genomic feature, and an hyper-edge $e \in E$ connects a subset of vertices (i.e., $e \subseteq V$) representing a possible alignment. By construction, the graph is k-partite: each hyper-edge contains at most one vertex from each partition (feature map). Hyper-edges can skip a partition to model gaps in the alignment.

We build the graph H iteratively. First, we create edges between the nodes in the first two maps (according to the criteria below), then extend them as hyper-edges to the other maps. Given a current (hyper)edge $e \in E$, in order to limit the size of H we extend it to a vertex v in the new partition only if (i) v has a position within $[-\delta, +\delta]$ of the position of the hyper-edge (computed as the average of the position of the nodes that belong to e) and (ii) it does not cross other hyper-edges. Once H is built, we solve the weighted k-partite matching via an integer linear program (ILP) [4], as follows:

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i=1}^{|E|} w_i x_i & \\
\text{subject to} & \displaystyle\sum_{i \in S_j} x_i = 1 & j = 1, \ldots, |V| \\
 & x_i \in \{0, 1\} & i = 1, \ldots, |E|
\end{array}
$$

where binary variable x_i is associated to hyper-edge $e_i \in E$, S_j is the set of hyper-edges incident to node $v_j \in V$, and weight w_i is the absolute deviation of hyper-edge e_i, i.e., the sum of absolute distance between the coordinate of each feature in e_i and their mean.

The integer program is relaxed to a linear program by allowing each variable to take values in the interval $[0, 1]$. The linear program is solved using the off-the-shelf solver GLPK [29].

5 Experimental results on synthetic data

A synthetic dataset can be described by five parameters, namely (i) the number k of maps, (ii) the number n of features, (iii) the minimum distance d between features, (iv) the probability p of a gap, and (v) the allowed “movement” σ. We generate first a “master” map Q where the features are placed at random locations so that the average distance between adjacent features is uniformly distributed in $[d, 2d]$. We generate the k maps from the initial map Q, as follows. We shift the location of each feature $f \in Q$ by a random quantity drawn from Gaussian distribution with parameters $(0, \sigma)$, then assign f it to a map at the perturbed location. Finally, we replace a feature with a gap with a small probability p. Observe that this procedure might produce alignments that do not satisfy the assumption that trajectories cannot cross.

5.1 Performance analysis

We analyzed the performance of ThIEF:LP and ThIEF:Iterative against the naïve (greedy) approach on several synthetic datasets. We generated a large number of datasets

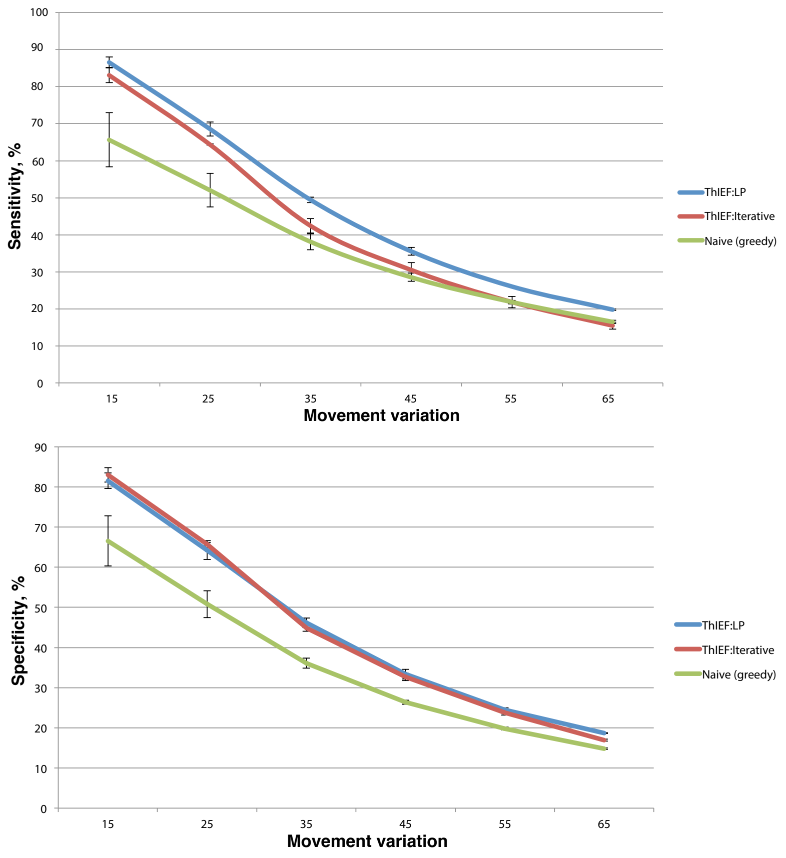


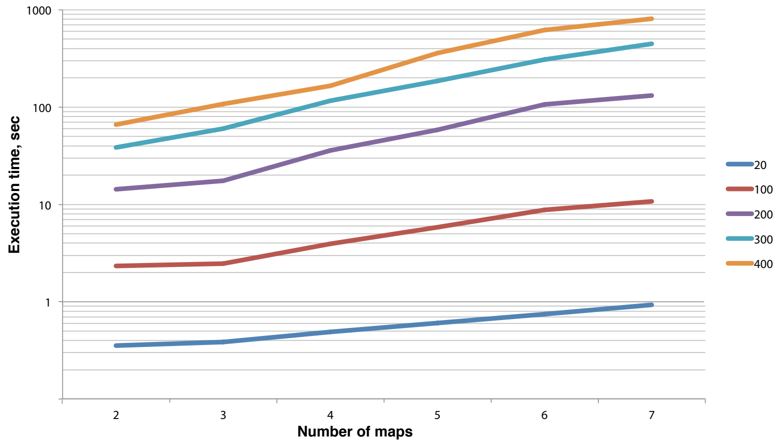
Figure 4 Sensitivity (TOP) and specificity (BOTTOM) of ThIEF:LP and ThIEF:Iterative, and the naïve for several choices of parameter $\sigma = [15, 65]$.

consisting of $k = 3$ maps and $n = 1000$ features, using different values for minimum distance d between features, gap probability p and movement variation σ.

We compared the alignments produced by these tools against the "ground-truth" and measured sensitivity and specificity. Figure 4-LEFT shows the average sensitivity as a function of parameter σ. Observe that ThIEF:LP outperforms the other approaches and ThIEF:Iterative's performance is better then naïve approach (as expected). Also observe that as σ increases, the performance decreases: this can be explained by the fact that with more variability in the location of the features it becomes more likely to have crossing of trajectories, which none of the tools is designed to capture. Specificity analysis shows a similar behavior (see Figure 4-RIGHT). Here ThIEF:LP and ThIEF:Iterative have almost identical performance, which is better than the naïve.

5.2 Execution time

To study the speed of the two algorithms we measured the total execution time on a variety of input datasets. ThIEF:Iterative's time complexity is $\Theta(km^3)$, where k is the number of maps and m is an upper bound on the total number of alignments. The functional dependency between m and n is data-dependent, specifically on how many new alignments are introduced



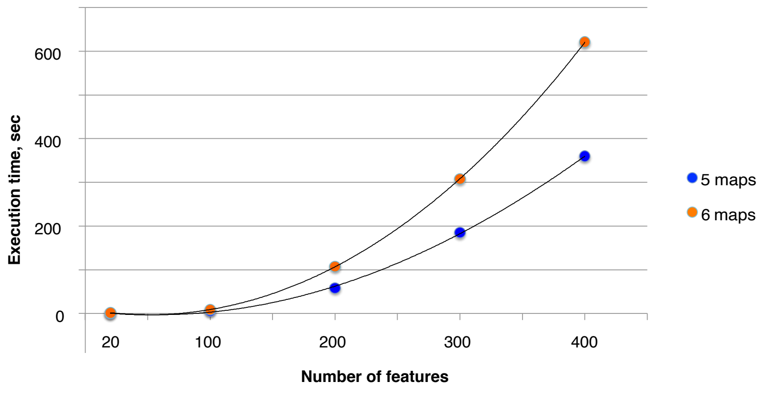


Figure 5 Execution time of THIEF:ITERATIVE. (LEFT) As a function of the number $k = 2, \dots, 7$ of maps to align, for different choices of the number n of features (20, 100, 200, 300, 400); (RIGHT) As a function of the number of features (for $k = 5, 6$ maps).

at every iteration (instead of extending existing ones). The worst case is $m \in O(2^{k-1}n)$. As a result, the total worst-case time-complexity could be as bad as $O(2^k n^3)$. Figure 5-LEFT shows the experimental dependency between the execution time of THIEF:ITERATIVE and the number of maps k. Observe that since the Y axis is log-scale, these experiments confirm that the actual running time is exponential. Figure 5-RIGHT shows the dependency between the execution time of THIEF:ITERATIVE and the number of features n, for $k = 5$ and $k = 6$ maps. The solid lines are cubic functions of n fitted to the data. These experiments confirm the cubic dependency on number of features in the input.

THIEF:LP's running time is dominated by the cost of solving a linear program, which in the case of the simplex algorithm, has exponential worst-case running time (although in practice, for the large majority of the instances the time-complexity of simplex is polynomial). The size of the linear program depends on the size of the hyper-graph: in our implementation, the size of the graph can be exponential in k. Figure 6-LEFT shows the dependency between execution time and the number of maps k. Each curve shows a linear trend (note that Y axis is in log-scale). The different shape of the blue curve (twenty features per map) could be explained by the fact that with small inputs the I/O overhead of transferring the data to the GLPK solver dominates the execution time. When we consider the blue curve for at

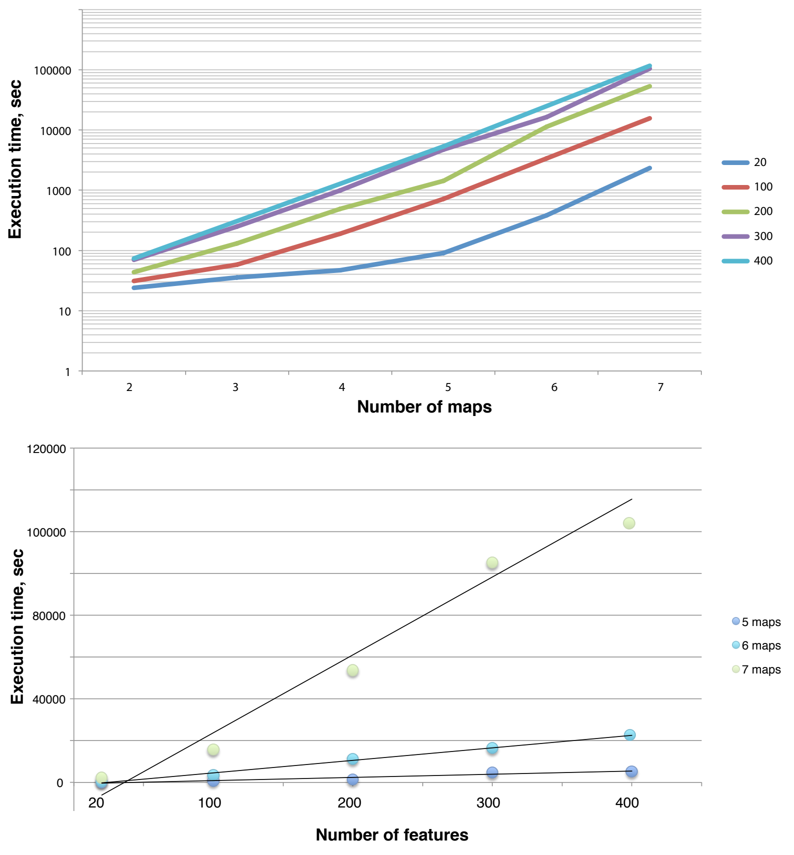


Figure 6 Execution time of THIEF:LP. (TOP) As a function of the number $k = 2, \ldots, 7$ of maps to align, for different choices of the number n of features (20, 100, 200, 300, 400); (BOTTOM) As a function of the number of features (for $k = 5, 6, 7$ maps).

least five maps the size of the hyper-graph becomes big enough so that solving the linear program dominate the execution time. These experiments support the claim of exponential complexity on the number of maps. Figure 6-RIGHT shows a linear dependency between the execution time and the number of features to track, as expected from the theoretical analysis.

6 Experimental results on real data

As mentioned previously, current imaging technology does not allow one to track nucleosomes or specific histone marks genome-wide over time. Since real data has no "ground-truth" about trajectories that would allows us to objectively evaluate our tool, nor we have found other tools that solve the same problem to which we could compare THIEF, we decided to demonstrate the utility of THIEF by developing a supervised classifier that can accurately predict the position of stable and unstable nucleosomes from the primary DNA sequence. We define a nucleosome to be *stable* when it appears in approximately the same position in the genome at all time points of an experiment. We define a nucleosome to be *unstable* when it appears in approximately the same position in at most half of the time points of an experiment. Nucleosomes in the second category are expected to be the most informative for investigating how chromatin structure affects gene expression. While there is significant amount of work on predicting the binding of nucleosomes from the DNA primary sequence (e.g., [22, 51, 26, 52, 27, 48, 35, 43]), we are not aware of any work that has addressed the problem of predicting whether a nucleosome is expected to be stable or unstable from the DNA sequence.

6.1 Detecting stable and unstable nucleosomes

As an example of application of ThIEF, we analyzed eight nucleosome maps from a study on the human malaria parasite [19]. In this study, synchronized *P. falciparum* parasites were collected at eight different stages of intra-erythrocytic development with five hour intervals (T5-T40). The eight samples were digested to enrich for nucleosome-bound DNA using the MNase protocol, then the digested samples were paired-end sequenced on an Illumina sequencing instrument. Raw reads provided by [19] were mapped to *P. falciparum* 3D7 genome v13.0 (available from `www.plasmoDB.org`) using Bowtie2 [24] allowing a maximum of one mismatch per read. Reads that mapped to multiple locations in the genome, reads that were PCR duplicates, and reads that mapped to ribosomal RNA or transfer RNA were discarded. The final datasets contained about 409 M mapped paired-end reads, with an average read length of 100 bp.

We used PuFFIN [36] to generate nucleosomes positions (for all eight time points) from the aligned reads. Then, we used ThIEF:LP to produce nucleosome trajectories by solving 14 independent optimization problems (one for each chromosome of *P. falciparum*). PuFFIN detected a total of 770,238 nucleosomes, with an average of 96,280 nucleosomes per time point. Nucleosome positions were consistent with the one reported in [19].

ThIEF:LP reported a total of 141,363 trajectories (average of 10,097 trajectories for chromosome). A trajectory generated by ThIEF:LP was considered *stable* if it contained nucleosomes at each time point (i.e., no missing nucleosomes). Trajectories with no more than four nucleosomes (out of eight) were considered as *unstable* (i.e., at least three missing nucleosomes). According to this definition we detected 43,122 stable trajectories (about 31% of the total) and 50,783 unstable trajectories (about 36% of the total). Trajectories that were not stable or unstable were not used in the experiments (about 33% of the total). Chromosomes by chromosomes, the percentage of stable trajectories ranged from 33.23% to 35.39% of the total number of trajectories. The percentage of unstable trajectories for all chromosomes ranged from 33.5% to 40.4%. The distribution of stable and unstable trajectories along the *P. falciparum* chromosomes is shown in Figure 7. Observe that there are regions devoid of stable nucleosomes, and regions very enriched for unstable nucleosomes (e.g., the telomere of chromosome 13).

6.2 A classifier for stable and unstable nucleosomes binding sites

First, we extracted the binding sites from the genome of *P. falciparum* for each trajectory. For each trajectory t labeled stable or unstable, we computed the average position of the nucleosomes in t, then selected 147 bp centered at that position from the malaria genome. Recall that a nucleosome consists of approximately 146-147 bp of DNA wrapped in superhelical turns around a histone octamer complex. Using this procedure we produced a training set composed of 43,122 147bp-long sequences representing stable nucleosomes, and 50,783 147bp-long sequences representing unstable nucleosomes.

We chose a Support Vector Machine (SVM) with radial basis function (RBF) kernel as our binary classifier. We explored many features to use in the classification, including k-mer

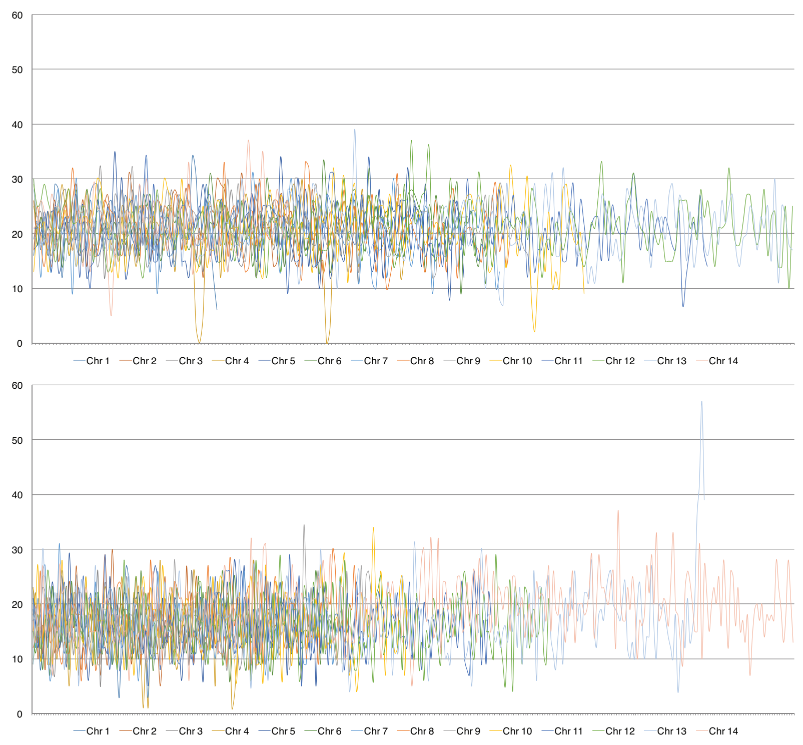


Figure 7 Distribution of stable (TOP) and unstable (BOTTOM) nucleosome trajectories along the 14 human malaria chromosomes (counts in a sliding window of 50Kbp).

distributions for $2 \leq k \leq 5$ and other physico-chemical properties of DNA (e.g., purine-pyrimidine, amino-keto, strong-weak H-bond). By running an extensive set of cross-validation experiments we determined that the five features described next were the most informative.

The first four features were obtained from the 3-mer distribution. For each sequence we collected its 3-mer frequencies, then computed the eigenvalues of four 4×4 matrices corresponding to 3-mers that have a middle nucleotide being `A`, `C`, `G`, and `T`, respectively. We represented each matrix with its leading eigenvalue, for a total of four features. The fifth feature was the DNA stability ΔG which is expressed by in terms of free energies of di-nucleotide stacks ΔG_{KL}, as described in [38]. The stability ΔG is measured by

$$\begin{aligned}\Delta G &= \frac{x_{GC}^2}{4}\left[\sum_{\mathcal{A}} \Delta G_{KL} + \sum_{\mathcal{B}} \Delta G_{KL} - \sum_{\mathcal{C}} \Delta G_{KL}\right] \\ &+ \frac{x_{GC}}{2}\left[\frac{1}{2}\sum_{\mathcal{C}} \Delta G_{KL} - \sum_{\mathcal{B}} \Delta G_{KL}\right] + \frac{1}{4}\sum_{\mathcal{B}} \Delta G_{KL} \qquad (1)\end{aligned}$$

where $\mathcal{A} = \{GG, CC, GC, CG\}$, $\mathcal{B} = \{AA, TT, AT, TA\}$ and $\mathcal{C} = \{GA, AG, CT, TC, GT, TG, CA, AC\}$; ΔG_{KL} is the standard melting free energy parameter where di-nucleotide stacks are calculated from stacking free energy parameters ΔG^{ST}_{KL}. Table 1 lists the value of ΔG^{ST}_{KL} and ΔG_{KL}, obtained from [38]. We z-normalized the feature vectors before training the SVM.

Table 2 shows the classification results (precision, accuracy, recall, and area under the ROC curve) when training the SVM on the DNA binding sites of stable/unstable trajectories

Table 1 (LEFT) Stacking free energy parameters ΔG^{ST}_{KL}, (RIGHT) Standard melting free energy parameters ΔG_{KL}.

KL	A	T	G	C
A	-1.11	-1.34	-1.06	-1.81
T	-0.19	-1.11	-0.55	-1.43
G	-1.43	-1.81	-1.44	-2.17
C	-0.55	-1.06	-0.91	-1.44

KL	A	T	G	C
A	-1.04	-1.27	-1.29	-2.04
T	-0.12	-1.04	-0.78	-1.66
G	-1.66	-2.04	-1.97	-2.70
C	-0.78	-1.29	-1.44	-1.97

Table 2 Classification results on the stable/unstable dataset for *P. falciparum*, by training the SVM on the odd-numbered chromosomes, and testing on the even-numbered chromosomes; AUC is the area under the ROC curve.

γ	C	*precision*	*accuracy*	*recall*	*F-score*	AUC
0.5	2	91.32%	79.60%	70.50%	0.795687	0.905418
0.5	8	92.48%	79.18%	68.64%	0.787988	0.908831
2	0.125	92.40%	79.97%	70.23%	0.798028	0.91509
2	0.5	92.59%	79.37%	68.91%	0.79015	0.913866
2	2	93.23%	79.28%	68.19%	0.787704	0.91498
8	0.03125	95.17%	80.32%	68.55%	0.797005	0.924962
8	0.125	94.08%	79.33%	67.58%	0.786582	0.921428
8	0.5	93.95%	79.14%	67.32%	0.784348	0.919447
32	0.007812	96.31%	79.79%	66.70%	0.788176	0.926078
32	0.03125	96.10%	79.39%	66.11%	0.7833	0.929091
32	0.125	94.65%	79.27%	67.00%	0.784592	0.92311
128	0.125	94.72%	79.23%	66.88%	0.78399	0.923386
512	0.125	94.71%	79.03%	66.51%	0.781401	0.923546
2048	0.03125	95.26%	79.04%	66.10%	0.780456	0.926556
8192	0.03125	94.99%	79.07%	66.36%	0.781336	0.925219

for even-numbered chromosomes of *P. falciparum*, and testing it on the odd-numbered chromosomes, for several choices of the penalty parameter C and kernel parameter γ in libSVM [5]. If TP, FP, TN and FN are true positive, false positive, true negative, and false negative, respectively, *precision* is defined as $TP/(TP+FP)$, *accuracy* is $(TP+TN)/(TP+TN+FP+FN)$, *recall* is $TP/(TP+FN)$, and the F-Score (also known as the F_1-score) is $(2\ precision\ \ recall)/(precision\ +\ recall)$. Observe that the SVM classifier makes a prediction for 66%-70% of tested nucleosome binding sites, and the precision of the prediction is very high. In 91%-95% of the predictions, the classifier can correctly determine solely from the DNA sequence whether the nucleosome will be stable or unstable. This is a surprisingly remarkable result.

7 Discussion and conclusion

We described the general problem of aligning multiple genome-wide epigenetic maps. We proposed two novel algorithms for this problem, namely ThIEF:LP and ThIEF:Iterative. The former finds a global optimal solution by constructing a hyper-graph representing the problem and solves it via linear programming. The latter reconstructs the final alignments by computing pair-wise alignments using the Hungarian algorithm. We determined that ThIEF:LP has slightly better sensitivity than ThIEF:Iterative, however the latter approach is more suitable for aligning a large number of maps. Both tools perform significantly better than the naïve (greedy) approach.

We have also demonstrated how ThIEF can be used in downstream analyses. In our example we used ThIEF to generate nucleosome trajectories for a time-course dataset on *P. falciparum*. The output of ThIEF can be used directly, and for the first time, to label nucleosome trajectories as stable or unstable. As proof of utility, we used the nucleosome-bound DNA sequence (labeled stable or unstable) to train a SVM classifier. Surprisingly, the classifier was able to predict with high accuracy and precision whether a particular 147 bp-long sequence is likely to contain a stable or unstable nucleosome. To the best of our knowledge, this is the first result on the prediction of the dynamics of nucleosomes solely based on their DNA binding preference.

Acknowledgements. The authors thank Christian Shelton (UCR) and Suhn Kyong Rhie (USC) for helpful discussions on this manuscript, and Xueqing (Maggie) Lu (UCR) for providing the nucleosome datasets.

References

1 Schahram Akbarian, Chunyu Liu, et al. The PsychENCODE project. *Nature Publishing Group*, 2015.

2 Bradley E. Bernstein, John A. Stamatoyannopoulos, Joseph F. Costello, Bing Ren, Aleksandar Milosavljevic, Alexander Meissner, Manolis Kellis, Marco A. Marra, Arthur L. Beaudet, Joseph R. Ecker, Peggy J. Farnham, Martin Hirst, Eric S. Lander, Tarjei S. Mikkelsen, and James A. Thomson. The NIH roadmap epigenomics mapping consortium. *Nat Biotech*, 28(10):1045–1048, 10 2010.

3 Daniel Blankenberg, Gregory Von Kuster, Nathaniel Coraor, Guruprasad Ananda, Ross Lazarus, Mary Mangan, Anton Nekrutenko, and James Taylor. Galaxy: A web-based genome analysis tool for experimentalists. *Current protocols in molecular biology*, pages 19–10, 2010.

4 Yuk Hei Chan and Lap Chi Lau. On linear and semidefinite programming relaxations for hypergraph matching. *Mathematical programming*, 135(1-2):123–148, 2012.

5 Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

6 Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–7, January 2004.

7 Isaac Elias. Settling the intractability of multiple alignment. *Journal of Computational Biology*, 13(7):1323–1339, 2006.

8 ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 2012.

9 Jason Ernst and Manolis Kellis. ChromHMM: automating chromatin-state discovery and characterization. *Nature Methods*, 2012.

10 Mark B. Gerstein, Zhi John Lu, et al. Integrative analysis of the *Caenorhabditis elegans* genome by the modENCODE project. *Science*, 2010.

11 Belinda Giardine, Cathy Riemer, Ross C. Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, Webb C. Miller, W. James Kent, and Anton Nekrutenko. Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10):1451–1455, 2005.

12 Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

13 Desmond Higgins and Paul Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

14 Michael M. Hoffman, Orion J. Buske, Jie Wang, Zhiping Weng, Jeff A. Bilmes, and William Stafford Noble. Unsupervised pattern discovery in human chromatin structure through genomic segmentation. *Nature Methods*, 9:473–476, 2012.

15 Michael M. Hoffman, Jason Ernst, Steven P. Wilder, Anshul Kundaje, Robert S. Harris, Max Libbrecht, Belinda Giardine, Paul M. Ellenbogen, Jeffrey A. Bilmes, Ewan Birney, Ross C. Hardison, Ian Dunham, Manolis Kellis, and William Stafford Noble. Integrative annotation of chromatin elements from ENCODE data. *Nucleic Acids Research*, 41(2):827–841, 2013.

16 Masato Ishikawa, Tomoyuki Toya, Masaki Hoshida, Katsumi Nitta, Atushi Ogiwara, and Minoru Kanehisa. Multiple sequence alignment by parallel simulated annealing. *Comput. Appl. Biosci.*, 9(3):267–273, 1993.

17 Roy Jonker and Ton Volgenant. Improving the Hungarian assignment algorithm. *Operations Research Letters*, 5(4):171–175, 1986.

18 W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615–623, 2001.

19 Philip Reiner Kensche, Wieteke Anna Maria Hoeijmakers, Christa Geeke Toenhake, Maaike Bras, Lia Chappell, Matthew Berriman, and Richárd Bártfai. The nucleosome landscape of plasmodium falciparum reveals chromatin architecture and dynamics of regulatory sequences. *Nucleic Acids Research*, 44(5):2110–2124, 2016.

20 W. James Kent, Robert Baertsch, Angie Hinrichs, Webb Miller, and David Haussler. Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proc. Natl. Acad. Sci. U. S. A.*, 100(20):11484–11489, 30 September 2003.

21 Jin Kim, Sakti Pramanik, and Moon Chung. Multiple sequence alignment using simulated annealing. *Comput. Appl. Biosci.*, 10(4):419–426, 1994.

22 R. D. Kornberg and L. Stryer. Statistical distributions of nucleosomes: nonrandom locations by a stochastic mechanism. *Nucleic Acids Research*, 16(14A):6677–6690, 07 1988.

23 Ekaterina Kotelnikova, Vsevolod Makeev, and Mikhail Gelfand. Evolution of transcription factor DNA binding sites. *Gene*, 347(2):255–263, 2005.

24 Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with bowtie 2. *Nat Meth*, 9(4):357–359, 04 2012.

25 M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins. Clustal W and Clustal X version 2.0. *Bioinformatics*, 23(21):2947–2948, 2007.

26 Elisa Leimgruber, Queralt Seguin-Estevez, Isabelle Dunand-Sauthier, Natalia Rybtsova, Christoph D. Schmid, Giovanna Ambrosini, Philipp Bucher, and Walter Reith. Nucleosome eviction from MHC class II promoters controls positioning of the transcription start site. *Nucleic Acids Res*, 37(8):2514–2528, May 2009.

27 Hongde Liu, Xueye Duan, Shuangxin Yu, and Xiao Sun. Analysis of nucleosome positioning determined by DNA helix curvature in the human genome. *BMC Genomics*, 12:72, Jan 2011.

28 Saulius Lukauskas, Roberto Visintainer, Guido Sanguinetti, and Gabriele B. Schweikert. DGW: an exploratory data analysis tool for clustering and visualisation of epigenomic marks. *BMC Bioinformatics*, 17(16):447, 2016.

29 Andrew Makhorin. GLPK (GNU linear programming kit), 2008.

30 Alessandro Mammana and Ho-Ryun Chung. Chromatin segmentation based on a probabilistic model for read counts explains a large portion of the epigenome. *Genome Biology*, 16(1):151, 2015.

31 Alan Moses, Derek Chiang, Daniel Pollard, Venky Iyer, and Michael Eisen. MONKEY: identifying conserved transcription-factor binding sites in multiple alignments using a binding site-specific evolutionary model. *Genome biology*, 5(12):R98, 2004.

32 S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, Mar 1970.

33 C. Notredame and D. G. Higgins. SAGA: Sequence alignment by genetic algorithm. *Nucleic Acids Res.*, 24(8):1515–1524, 1996.

34 C. Notredame, E. A. O'Brien, and D. G. Higgins. RAGA: RNA sequence alignment by genetic algorithm. *Nucleic acids research*, 25(22):4570–4580, 1997.

35 Heather E. Peckham, Robert E. Thurman, Yutao Fu, John A. Stamatoyannopoulos, William Stafford Noble, Kevin Struhl, and Zhiping Weng. Nucleosome positioning signals in genomic DNA. *Genome Res*, 17(8):1170–1177, Aug 2007.

36 A. Polishko, E. M. Bunnik, K. G. Le Roch, and S. Lonardi. PuFFIN: A parameter-free method to build nucleosome maps from paired-end reads. *BMC Bioinformatics*, 15(Suppl 9):S11, 2014.

37 Anton Polishko, Nadia Ponts, Karine G Le Roch, and Stefano Lonardi. NORMAL: accurate nucleosome positioning using a modified gaussian mixture model. *Bioinformatics (Oxford, England)*, 28(12):i242–9, June 2012.

38 Ekaterina Protozanova, Peter Yakovchuk, and Maxim D. Frank-Kamenetskii. Stacked-unstacked equilibrium at the nick site of DNA. *J Mol Biol*, 342(3):775–785, Sep 2004.

39 Rainer Pudimat, Ernst-Günter Schukat-Talamazzini, and Rolf Backofen. A multiple-feature framework for modelling and predicting transcription factor binding sites. *Bioinformatics*, 21(14):3082–3088, 2005.

40 Aaron R. Quinlan and Ira M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

41 S. Roy, J. Ernst, P. V. Kharchenko, and P. Kheradpour. Identification of functional elements and regulatory circuits by Drosophila modENCODE. *Science*, 2010.

42 Rafik A. Salama and Dov J. Stekel. A non-independent energy-based multiple sequence alignment improves prediction of transcription factor binding sites. *Bioinformatics*, 29(21):2699–2704, 2013.

43 Eran Segal, Yvonne Fondufe-Mittendorf, Lingyi Chen, AnnChristine Thastrom, Yair Field, Irene K. Moore, Ji-Ping Z. Wang, and Jonathan Widom. A genomic code for nucleosome positioning. *Nature*, 442(7104):772–778, 08 2006.

44 T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, Mar 1981.

45 Julie Thompson, Toby Gibson, and Des Higgins. Multiple sequence alignment using ClustalW and ClustalX. *Current protocols in bioinformatics*, Chapter 2, 2002.

46 Julie Thompson, Desmond Higgins, and Toby Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

47 Mari-Liis Visnapuu and Eric C Greene. Single-molecule imaging of DNA curtains reveals intrinsic energy landscapes for nucleosome deposition. *Nat Struct Mol Biol*, 16(10):1056–1062, 10 2009.

48 Jia Wang, Shuai Liu, and Weina Fu. Nucleosome positioning with set of key positions and nucleosome affinity. *Open Biomed Eng J*, 8:166–170, 2014.

49 L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

50 Yong Zhang, Tao Liu, Clifford A. Meyer, Jérôme Eeckhoute, David S. Johnson, Bradley E. Bernstein, Chad Nusbaum, Richard M. Myers, Myles Brown, Wei Li, and X. Shirley Liu. Model-based analysis of ChIP-Seq (MACS). *Genome Biology*, 9(9):R137, 2008.

51 Yong Zhang, Zarmik Moqtaderi, Barbara P. Rattner, Ghia Euskirchen, Michael Snyder, James T. Kadonaga, X. Shirley Liu, and Kevin Struhl. Intrinsic histone-DNA interactions are not the major determinant of nucleosome positions in vivo. *Nat Struct Mol Biol*, 16(8):847–852, Aug 2009.

52 Xiujuan Zhao, Zhiyong Pei, Jia Liu, Sheng Qin, and Lu Cai. Prediction of nucleosome DNA formation potential and nucleosome positioning using increment of diversity combined with quadratic discriminant analysis. *Chromosome Res*, 18(7):777–785, Nov 2010.

Byte-Aligned Pattern Matching in Encoded Genomic Sequences

Petr Procházka[1] **and Jan Holub**[2]

1 **Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic**
`Petr.Prochazka@fit.cvut.cz`

2 **Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Prague, Czech Republic**

Abstract

In this article, we propose a novel pattern matching algorithm, called `BAPM`, that performs searching in the encoded genomic sequences. The algorithm works at the level of single bytes and it achieves sublinear performance on average. The preprocessing phase of the algorithm is linear with respect to the size of the searched pattern m. A simple $\mathcal{O}(m)$-space data structure is used to store all factors (with a defined length) of the searched pattern. These factors are later searched during the searching phase which ensures sublinear time on average. Our algorithm significantly overcomes the state-of-the-art pattern matching algorithms in the locate time on middle and long patterns. Furthermore, it is able to cooperate very easily with the block q-gram inverted index. The block q-gram inverted index together with our pattern matching algorithm achieve superior results in terms of locate time to the current index data structures for less frequent patterns. We present experimental results using real genomic data. These results prove efficiency of our algorithm.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases genomic sequences, pattern matching, q-gram inverted index

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.20

1 Introduction

DNA sequencing is nowadays the integral part of several disciplines like personalized medicine, biology, biotechnology, or forensic biology. The demand for cheap sequencing induced the evolution of High-Throughput Sequencing (HTS) technologies that can sequence large stretches of DNA in a massively parallel fashion and that produce millions of DNA sequences simultaneously. The public sources report the necessary time per one run in the order of hours and the cost per one million bases lower than 0.02 USD[1]. General availability of the sequencing causes producing large volumes of genomic data that needs to be stored effectively in the form allowing extremely fast searching.

DNA molecule can be mapped one-to-one to a sequence of letters which implies that it can be processed as a text string. The string matching problem is crucial task since early beginnings of the text processing. The task is very simple – to find all occurrences of a given pattern P in a large text T. However, this task is performed very frequently and over large volumes of data (text T) which implies that very fast algorithms are necessary. To accelerate

[1] `https://www.genome.gov/sequencingcostsdata/`

the string matching, the algorithm can preprocess either the pattern or the text or both. The pattern preprocessing is relevant only for the given pattern and therefore it is included in the search process. The text preprocessing (which includes especially the indexing methods) is universal for all possible patterns, however it usually requires some extra space to store an auxiliary data structure.

Knuth-Morris-Pratt (KMP) [11] is one of the most famous pattern matching algorithms and the first one ensuring the worst-case time linear with the length of the text T. Boyer-Moore (BM) [3] family algorithms represent backward pattern matching approach. BM algorithm allows skipping of some characters which leads to lower than linear average time. There exist also other variations of this algorithm given by Horspool [10] or Sunday [20]. Suffix automaton (often called DAWG – Deterministic Acyclic Word Graph) is the essence of another algorithm achieving sublinear average time BDM (Backward DAWG Match) [4]. The suffix automaton of the reversed pattern performs backward searching for the pattern. The byproduct of the search is always the longest prefix of the pattern occurring at that position in the text which ensures safe shifting for BDM. Another approach is to use non-deterministic instead of deterministic automata for searching in the text. So-called *bit-parallelism* [5, 2] proved to be a very simple way how to simulate the non-deterministic automaton. It exploits the parallelism provided by bitwise operations in terms of one computer word. It can accelerate the operations up to a factor w, where w is the number of bits in the computer word. Bit-parallelism is particularly efficient for the patterns with size lower than the size of the computer word $m \leq w$. Navarro et al. applied the bit-parallelism to simulate the suffix automaton and they proposed BNDM algorithm [17] that achieved 20%-25% improvement in search time in comparison to its deterministic version BDM. Later, Durian et al. [22] proposed an efficiency improvement of BNDM and Shift-Or algorithm residing in processing q-grams of the input symbols. BSDM [7] is relatively recent algorithm using suffix automaton searching for a factor with no repetitions of a condensed pattern. BSDM proved to be very fast especially for middle-sized and longer patterns. Very recently, the algorithms (e.g., [6, 21]) exploiting SIMD (Single Instruction, Multiple Data) instructions of modern CPUs appeared. EPSM [6] tabulates all the factors (of a given length) of the searched pattern. The factors are easy to access using hash table whereas the hash function is provided by CRC SIMD specialized instruction. The algorithm performs searching for any of the factors (in the filtering phase) and any of the hits must be confirmed by direct check at the corresponding position in the text.

We propose a novel pattern matching algorithm, called `BAPM` (Byte-Aligned Pattern Matching). Our algorithm is optimized for searching in the encoded genomic sequences only. It exploits the low alphabet size of the genomic sequences which implies a possibility to tabulate all factors (of a given length) of the pattern achieving reasonable memory consumption. Furthermore, a simple encoding scheme for genomic sequences allows to process the factors as a sequence of one or more bytes. The searching phase of `BAPM` resides in shifting over the encoded DNA sequence, reading a sequence of one or more bytes and comparing its value with the tabulated factors of the searched pattern. When a factor is found the potential occurrence of the searched pattern must be still confirmed by direct comparison of the text to the pattern. `BAPM` works at the level of bytes all the time. Only the very last step confirming a potential occurrence applies bitwise operations. This leads to high efficiency in searching for middle-sized and long patterns.

Preprocessing of the input text is another way how to speed up searching in the text. Suffix trees [23] are one of the fundamental index structures. Suffix array [14] is another basic index structure that significantly improves demanding space requirements of the suffix trees.

Other indexes like FM-index [8] or CSA (Compressed Suffix Array) [9] further improved the space requirements of the index data structure to be the same or lower than the size of the input text. A separate branch of research focused on indexing text files with natural language content. In this field, so-called inverted index [15] is considered as de-facto standard. However, the inverted index proved its efficiency also when performed on other kind of data [18]. We supplemented our `BAPM` with a simple block q-gram inverted index and experimentally compared its locate speed with state-of-the-art index data structures.

The rest of the paper is organized as follows. We give definitions of some basic notions in Section 2. The Section 3 is dedicated to definition and detailed description of `BAPM` algorithm and necessary data structures. The Section 4 summarizes experimental results performed on real genomic data. We give the conclusion and some ideas for future work in Section 5.

2 Basic Notions

Let $x = x_1x_2..x_n$ be a *string* composed of single symbols x_i of a finite ordered alphabet Σ. The length of the string x is $n = |x|$. The size of the alphabet Σ is $\sigma = |\Sigma| = \mathcal{O}(1)$. The start position i and the length j define so-called *factor* (or *substring*) denoted by $x_{i,j} = x_i..x_{i+j-1}$. A factor with $i = 0$ is called *prefix* and a factor with $i + j - 1 = n$ is called *suffix* of the string x. We denote by ε so-called empty string of length 0. The problem of string pattern matching is to find all occurrences of a pattern $P = p_1p_2..p_m$ in a text $T = t_1t_2..t_n$ where both strings are composed of symbols from the same alphabet Σ and $m \ll n$. Particularly, we can distinguish two tasks: (i) *count* when number of occurrences of P in T is reported and (ii) *locate* when exact positions of the occurrences of P in T are reported.

Pattern substitution method [13] is a compression method when q-grams of symbols of the input text T (i.e., Σ^q) are substituted with an assigned byte value b where $b \in \{0, 1, \ldots, 255\}$. The pattern matching on the compressed (encoded) text means to find all occurrences of the compressed pattern P_C in the compressed text T_C (both defined over the alphabet of byte values $b \in \{0, 1, \ldots, 255\}$).

Traditional *inverted index* consists of two major components: a vocabulary storing all distinct words occurring in the text T and a set of *posting lists* storing positions of all occurrences of a given word in the text T. The vocabulary of a q-gram inverted index [18] is composed of all possible q-grams of the alphabet Σ, i.e., Σ^q. For the purpose of *block indexing* we split the indexed text into single blocks of a defined fixed size. The posting lists of a *block inverted index* then store addresses of the blocks covering the exact positions of occurrences. The exact positions are determined in the next step when a standard pattern matching method is performed in terms of the preselected blocks.

In later description of the algorithm, we use C-like syntax for bitwise operations. Particularly, we use $|$ for bitwise-or, & for bitwise-and, $\ll$ for shift-left operation and $\gg$ for shift-right operation.

3 Byte-Aligned Pattern Matching

Byte-Aligned Pattern Matching algorithm (`BAPM`) is optimized for searching in the encoded genomic sequences. It assumes the input alphabet $\Sigma = \{A, C, G, T\}$ and a simple substitution encoding defined as $f : \Sigma^4 \mapsto B$ where $B = \{0, 1, \ldots, 255\}$ and $b \in B$ represents a byte value that is composed as a concatenation of bit couples given by the single symbols of the 4-gram $s \in \Sigma^4$ ($A \to 00$, $C \to 01$, $G \to 10$, $T \to 11$). The algorithm detects single occurrences within two steps. In the first step, the algorithm performs searching for all factors of a defined fixed

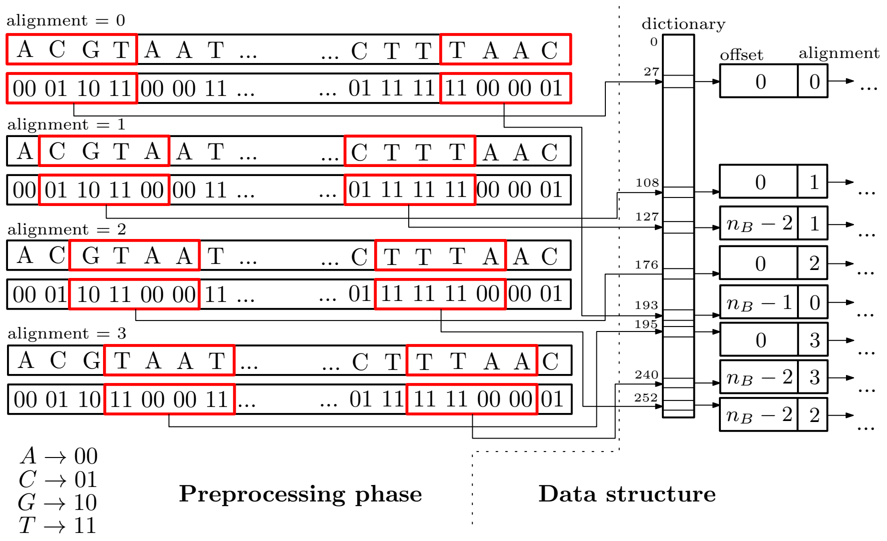


Figure 1 BAPM: Preprocessing phase. The length of the encoded pattern P_C is n_B bytes.

length that must be a multiple of 4 (in terms of the input alphabet Σ). This ensures that each encoded factor is represented as a sequence of one or more bytes. `BAPM` tabulates all possible factors of the encoded pattern during the preprocessing phase. It is reasonable to require the set of all factors to fit into the memory cache. For this reason, the acceptable lengths of the factors are 4 and 8 bases/symbols (in terms of the input alphabet Σ) which implies the length of one or two bytes, respectively for the encoded factors. The first step of the searching is only the filtering of possible occurrences. A potential occurrence must be always confirmed using direct comparison of the encoded pattern P_C with the encoded text T_C at a given position i. We have implemented two versions of `BAMP` tabulating 4-gram factors (`BAPM4`) and 8-gram factors (`BAPM8`), respectively. We explain all the principles of the algorithm using the version with 4-gram factors. However, the same principles are valid for the version with 8-gram factors as well. From now on, `BAPM` reports to 4-gram version of the algorithm if not stated other way.

Figure 1 depicts a simple data structure used to store the tabulated encoded factors of the pattern and it demonstrates also `BAPM` preprocessing phase when this data structure is filled. The dictionary data structure is depicted in the right part of the figure and its main part is an array with 256 entries (corresponding to 256 different byte values). Every entry can contain a pointer to a list which stores all occurrences of the factor (corresponding to the entry) in terms of the pattern. Each element of the list is a couple (offset, alignment). The offset o represents a byte position of the factor in the encoded pattern and it is easy deducible from its starting position i in the raw pattern $o = \lfloor \frac{i-1}{4} \rfloor$. The alignment a represents a position of the factor in terms of the byte and it can be computed as $a = (i-1) \mod 4$. Suppose constant size of the computer word. Then, it is obvious that the dictionary requires $\mathcal{O}(m)$ space where m is a size of the raw pattern. The space of the array is constant and the lists contain together $m-3$ elements, each of them consuming $\mathcal{O}(1)$ space. The offset o requires $\mathcal{O}(\log \frac{m}{4})$, however, we suppose it can be encoded within a single computer word in all practical cases.

The left part of Figure 1 describes single steps of the preprocessing phase of the algorithm. For every possible alignment $a \in \{0, 1, 2, 3\}$, consecutive byte values of the shifted encoded pattern P_C are processed. The value of the byte determines its position in the dictionary.

For every byte, its offset and alignment are stored to the corresponding list pointed from the dictionary. The remaining bases at the end of the shifted pattern that do not compose a complete byte are omitted (e.g., the suffix AAC for alignment = 1 in Figure 1).

Suppose the length of the raw pattern $m = 128$ bases which implies 32 bytes for the encoded pattern P_C. `BAPM4` needs to store $128 - 3 = 125$ factors (elements of the lists). Suppose a simple byte code used to store the offset o and the alignment a for every factor. Only two bytes are consumed for every pair (o, a) and still all the information is encoded at the level of bytes. Thus, for `BAPM4` the total space is $(128 - 3) \times 2 + 256 = 506$ bytes, plus some overhead needed to implement the lists. Still, the data structure easily fits into 2 kiB of memory and it can be kept in the top level of the computer cache. For `BAPM8`, we can estimate the needed space as $(128 - 7) \times 2 + 256 \times 256 = 65\,778$ bytes, plus the overhead for the lists. This is still acceptable space ensuring storing the data structure in the fast levels of the computer memory.

Algorithm 1 describes preprocessing and searching phase of `BAPM4`. The function ENCODE is called in the preprocessing phase. The function performs the simple substitution encoding described above. Its parameters are: the text to be encoded; the starting index for encoding; and the number of bases/symbols that need to be encoded. The function returns desired encoded factor of the text. The function BUILDDICTIONARY is responsible for constituting dictionary D and storing the shifted versions of the pattern in the array B. The **`while`** cycle (line 4) iterates over all possible alignments $a \in \{0, 1, 2, 3\}$. For every alignment, the number of bases/symbols that constitute the longest byte sequence starting at i is computed (line 5) and the corresponding encoded pattern is obtained (line 6). The encoded pattern is stored for the given alignment (line 7) and later is used for direct comparison of bytes (the encoded text with the encoded pattern). Next **`while`** cycle (line 9) iterates over all bytes of the encoded aligned pattern and it ensures storing the couples (offset, alignment) to their corresponding lists (line 12).

The function BUILDMASK is another part of the preprocessing. It generates all necessary masks possibly needed in the last step of the comparison (a prefix and/or a suffix of the encoded pattern with the corresponding part of the encoded text). Since the prefix and the suffix are smaller than one byte the masks are necessary to minimize the bitwise operations and therefore also the needed time. The function stores the masks in single variables. The variable *pref* stores a prefix of the encoded pattern (smaller than one byte) for all possible alignments $a \in \{0, 1, 2, 3\}$. The variable *suf* stores a suffix of the encoded pattern (smaller than one byte) for all possible alignments $a \in \{0, 1, 2, 3\}$. The examples of the stored prefixes and suffixes can be seen in Figure 1 as the symbols preceding/following the red rectangles. Similarly, the variables *prefMask* and *sufMask* store the masks (used for bitwise-and operation with the corresponding byte in the encoded text) needed to compare a prefix or the suffix, respectively of the encoded pattern. The **`while`** cycle (line 22) iterates over only three possible alignments. The *pref* value is stored for the alignments $a \in \{1, 2, 3\}$ (starting from the value 3). The prefix for the alignment $a = 0$ is an empty string ε and therefore it is not stored. The pointer to the array of the suffix values *suf* is shifted by the value *la* and it starts from the position $(la + i) \mod 4$. In every step of the cycle, the algorithm: (i) stores the corresponding prefix value to *pref* array and the corresponding prefix mask to *prefMask* array; (ii) stores the corresponding suffix value to *suf* array and the corresponding suffix mask to *sufMask* array; (iii) shifts auxiliary variables *prefM* and *prefB* two bits right; (iv) shifts auxiliary variables *sufM* and *sufB* two bits left.

The function SEARCH represents the main function of `BAPM`. After the preprocessing of the searched pattern (lines 31 and 32) the algorithm states a safe shift as the number of

Algorithm 1 BAPM4 preprocessing and searching phase

```
 1: function buildDictionary($P$, $m$)
 2:     $D \leftarrow \emptyset$; $B \leftarrow \emptyset$;
 3:     $i \leftarrow 0$;
 4:     while $i \leq 3$ do
 5:         $b \leftarrow \lfloor (m - i)/4 \rfloor \times 4$;
 6:         $E \leftarrow$ encode($P, i+1, b$);
 7:         $B_i \leftarrow E$;
 8:         $j \leftarrow 1$;
 9:         while $j \leq \lfloor (m - i)/4 \rfloor$ do
10:             if $D_{E_j} = \emptyset$ then
11:                 $D_{E_j} \leftarrow$ create a new list storing offsets and alignments;
12:             add a couple of offset and alignment $(j-1, i)$ to the list $D_{E_j}$;
13:             $j \leftarrow j + 1$;
14:         $i \leftarrow i + 1$;

15: function buildMask($P$, $m$)
16:     $\mathit{pref} \leftarrow \emptyset$; $\mathit{prefMask} \leftarrow \emptyset$; $\mathit{prefM} \leftarrow$ 0x3f;
17:     $\mathit{suf} \leftarrow \emptyset$; $\mathit{sufMask} \leftarrow \emptyset$; $\mathit{sufM} \leftarrow$ 0xfc;
18:     $\mathit{prefB} \leftarrow$ encode($P, 1, 4$) $\gg 2$;
19:     $\mathit{sufB} \leftarrow$ encode($P, m-4, 4$) $\ll 2$;
20:     $\mathit{la} \leftarrow m \bmod 4$;
21:     $i \leftarrow 1$;
22:     while $i \leq 3$ do
23:         $\mathit{prefMask}_{4-i} \leftarrow \mathit{prefM}$;
24:         $\mathit{sufMask}_{(\mathit{la}+i) \bmod 4} \leftarrow \mathit{sufM}$;
25:         $\mathit{pref}_{4-i} \leftarrow \mathit{prefB}$;
26:         $\mathit{suf}_{(\mathit{la}+i) \bmod 4} \leftarrow \mathit{sufB}$;
27:         $\mathit{prefM} \leftarrow \mathit{prefM} \gg 2$; $\mathit{prefB} \leftarrow \mathit{prefB} \gg 2$;
28:         $\mathit{sufM} \leftarrow \mathit{sufM} \ll 2$; $\mathit{sufB} \leftarrow \mathit{sufB} \ll 2$;
29:         $i \leftarrow i + 1$;

30: function search($T$, $n$, $P$, $m$)
31:     buildDictionary($P, m$);
32:     buildMask($P, m$);
33:     $\mathit{shift} \leftarrow \lfloor m/4 \rfloor - 1$;
34:     $i \leftarrow \mathit{shift}$;
35:     while $i \leq n$ do
36:         if $D_{T_i} \neq \emptyset$ then
37:             for each couple of offset and alignment $(o, a) \in D_{T_i}$ do
38:                 $r \leftarrow$ compare all bytes starting from $T_{i-o}$ with $B_a$;
39:                 if $r = 0$ & $a \neq 0$ then
40:                     $r \leftarrow$ compare ($T_{i-o-1}$ & $\mathit{prefMask}_a$) with $\mathit{pref}_a$;
41:                 if $r = 0$ & $a \neq \mathit{la}$ then
42:                     $r \leftarrow$ compare ($T_{i-o+\mathit{shift}}$ & $\mathit{sufMask}_a$) with $\mathit{suf}_a$;
43:                 if $r = 0$ then
44:                     report an occurrence at position $4 \times (i - o) - a + 1$;
45:         $i \leftarrow i + \mathit{shift}$;
```

whole bytes of the encoded pattern minus one (line 33). The `while` cycle (line 35) traverses the encoded text T of length n. It always reads a byte value T_i and the corresponding entry in the dictionary D_{T_i} is checked (line 36). If the dictionary entry D_{T_i} is empty the algorithm shifts (line 45) and it continues at the next position. Otherwise, the algorithm has to traverse over all couples (o, a) stored in the corresponding list and perform three-level comparison for every couple. The first level is comparison of the bytes in the encoded text (starting at the position given by the offset o) with the bytes of the encoded pattern B_a according to the shift/alignment a (see line 38). The second level (see line 40) is comparison of the prefix and it is applied only if the first level was successful. The third level (see line 42) is

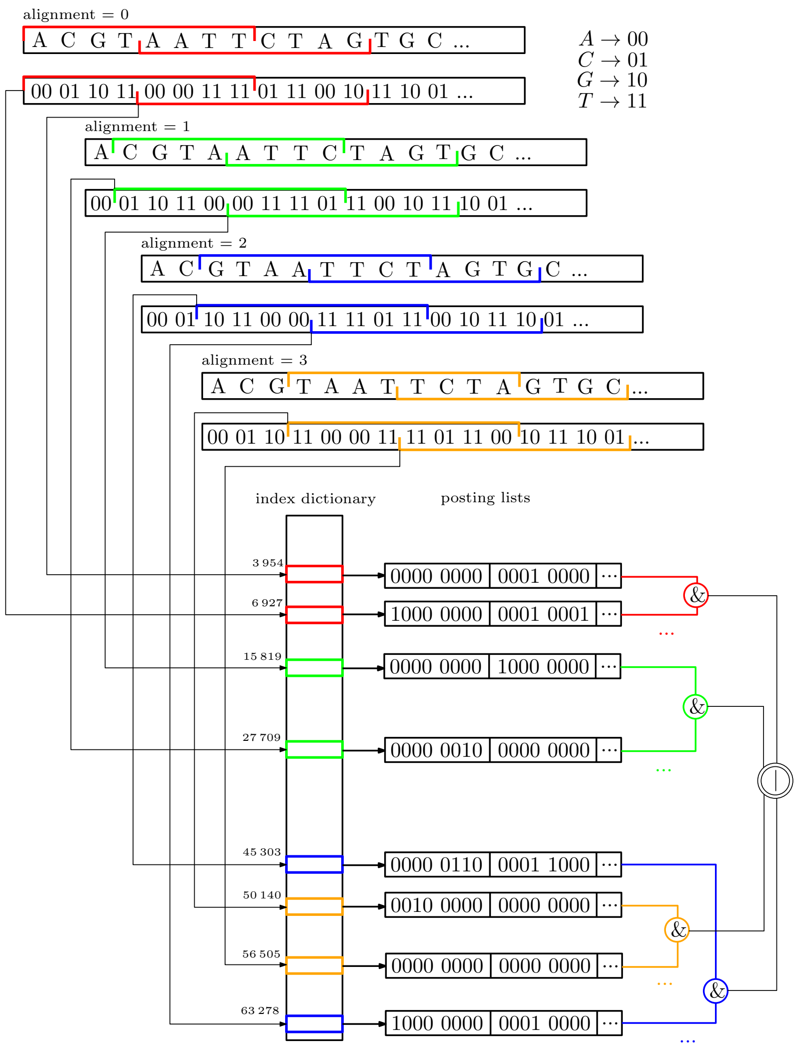


Figure 2 Block q-gram inverted index. Single colors (red, green, blue, yellow) represent the alignments $a \in \{0, 1, 2, 3\}$ of the pattern.

comparison of the suffix and it is applied only if the second level was successful. If all levels of the comparison are successful the algorithm reports a new occurrence at the corresponding position $4 \times (i - o) - a + 1$ in the raw text (line 44).

The preprocessing phase of the algorithm needs clearly $\mathcal{O}(m)$ time at most. In the function BUILDDICTIONARY, the algorithm consumes $\mathcal{O}(m)$ time to perform encoding (line 6) and $\mathcal{O}(\frac{m}{4})$ time to perform the `while` cycle (line 9). Other steps of the function are performed in the constant time. The function BUILDMASK contains all steps that are performed in the constant time. The worst-case time complexity for the searching phase of the algorithm is given by the `while` cycle (line 35) traversing the text and the `for each` cycle traversing the list of couples (line 37). Thus, the upper bound is $\mathcal{O}(nm)$. However, the average time is lower than linear for real genomic data. According to our tests on real data, the most of the factors (especially for BAPM8) occur only once in the pattern and therefore majority of the lists pointed from the dictionary D contain only one element. Furthermore, especially for longer patterns where the size of the pattern is significantly greater than the size of the tabulated factors, the algorithm jumps over majority of the processed text and so achieves lower than linear time. We can conclude that the worst-case time of the algorithm is $\mathcal{O}(nm)$, however the average expected time is lower than linear $\mathcal{O}(n)$.

The next logical step in improving efficiency of the searching is to add an index data structure. Navarro et al. [16] proved the efficiency of the block inverted index in combination with sequential scanning of the encoded text. `BAPM` works with the encoded q-grams ($q \in \{4, 8\}$) so we decided to supply it with the block q-gram inverted index. The $q = 8$ proved to be optimal in our experiments. The 8-gram factors are encoded into two-byte long values (short data type in C) which means that they are easily addressed and manipulated. Figure 2 gives a brief description of generating and applying the index. The index dictionary stores all encoded factors of all pattern alignments $a \in \{0, 1, 2, 3\}$. Every engaged dictionary entry points to a posting list implemented as a bitmap. Single bits correspond to blocks in the encoded text and are set to one when the q-gram occurs in the block.

Searching using the inverted index has the following steps. The pattern has to be encoded. For each alignment $a \in \{0, 1, 2, 3\}$ (represented by different colors in Figure 2) all two-byte long values are retrieved. The posting lists of all retrieved values (of a given alignment) are processed and bitwise-and operation is applied. Next, bitwise-or operation is applied among intermediate results of single alignments. Finally, the blocks of the encoded text corresponding to the set bits contain a possible occurrences of the pattern and need to be processed using `BAPM` to confirm the occurrences and report the exact positions in the text.

4 Experiments

We present experimental results that give a detailed comparison of our newly presented algorithms `BAPM4` and `BAPM8` with the state-of-the-art best algorithms. We considered all known algorithms focused especially on searching middle-sized and long patterns. The essence of `BAPM` (its byte orientation and its principle of tabulating all factors) predetermines this algorithm to search for patterns with length $m \geq 8$ bases. In particular, we compared `BAPM4` and `BAPM8` with the following algorithms (all of them from SMART library[2]):

- Exact Packed String Matching (EPSM) [7],
- Shift-Or algorithm (SO) [1],
- Backward-SRN-DAWG-Matching (BSDM4 and BSDM8) [6],
- Simplified BNDM with q-grams (SBNDMQ4 and SBNDMQ8) [22].

All the tested algorithms were implemented in C programming language[3]. We carried out our tests on Intel® Core™ i7-4702MQ 2.20 GHz, 8 GB RAM. We used compiler gcc version 5.4.0 with compiler optimization -O3. The tested patterns were chosen randomly from the input text and their length m was ranging from 12 to 256. All experiments were run in loop 1 000 times and we report the mean of the running time in milliseconds. All reported times represent measured `user` time + `sys` time and they always include any necessary preprocessing. For evaluating the algorithms, we used `Ecoli.txt` file from the Canterbury corpus[4] and `human100MB.txt` file that contains the sequence of human chromosome 15 from the project Ensembl[5].

[2] `http://www.dmi.unict.it/~faro/smart/`
[3] `BAPM4` and `BAPM8` implementation is available at `http://www.stringology.org/bapm/bapm.zip`
[4] `http://corpus.canterbury.ac.nz/descriptions/large/E.coli.html`
[5] `http://www.ensembl.org/info/data/ftp/index.html`

Table 1 `Ecoli.txt`: Locate times in milliseconds. The best results are bolded.

m	BAPM4	BAPM8	EPSM	SO	BSDM4	BSDM8	SBNDMQ4	SBNDMQ8
12	1.032	1.188	**1.031**	4.015	**1.031**	1.984	1.156	2.953
16	0.813	**0.609**	1.468	4.000	0.843	1.125	0.906	1.625
20	0.672	**0.407**	1.515	4.015	0.734	0.812	0.765	1.125
24	0.609	**0.328**	1.312	4.015	0.687	0.640	0.671	0.875
28	0.562	**0.265**	1.328	4.015	0.656	0.531	0.625	0.703
32	0.515	**0.219**	1.265	4.000	0.625	0.468	0.578	0.593
36	0.546	**0.203**	1.265	3.062	0.609	0.406	0.546	0.625
40	0.515	**0.187**	1.250	3.093	0.593	0.375	0.546	0.609
44	0.500	**0.172**	1.250	3.078	0.593	0.359	0.546	0.609
48	0.484	**0.156**	1.234	3.000	0.563	0.328	0.546	0.625
52	0.469	**0.141**	1.234	3.046	0.562	0.312	0.546	0.625
56	0.454	**0.140**	1.218	3.078	0.562	0.296	0.546	0.625
60	0.454	**0.125**	1.218	3.093	0.562	0.296	0.546	0.625
64	0.453	**0.124**	1.203	3.015	0.546	0.281	0.546	0.609
96	0.422	**0.102**	1.203	3.062	0.531	0.250	0.546	0.609
128	0.391	**0.094**	1.172	3.015	0.531	0.234	0.546	0.609
160	0.375	**0.092**	1.171	3.046	0.515	0.218	0.546	0.625
192	0.344	**0.092**	1.155	3.125	0.515	0.218	0.546	0.625
224	0.328	**0.095**	1.155	3.015	0.515	0.218	0.562	0.609
256	0.312	**0.098**	–	–	–	–	–	–

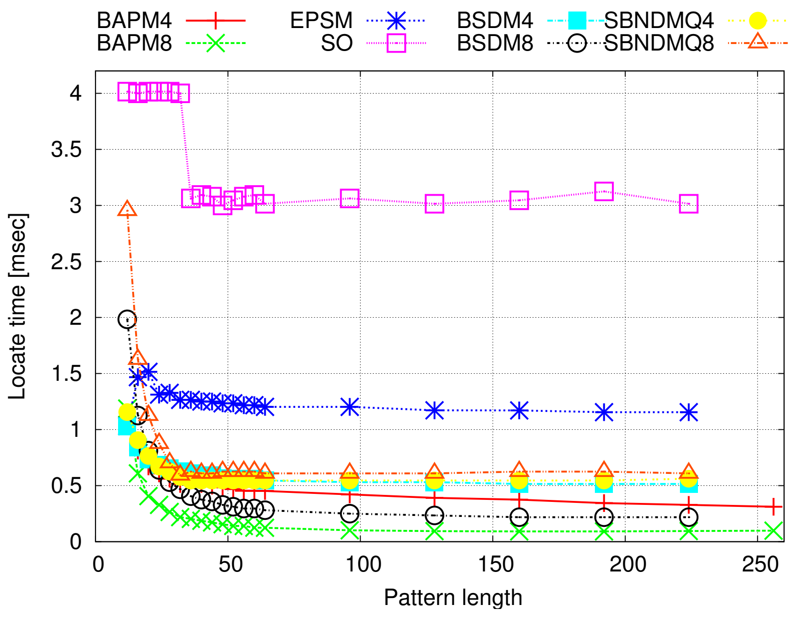


Figure 3 `Ecoli.txt`: Locate time depending on the length of the searched pattern m.

Table 1 and Figure 3 report the results of single algorithms when performed on the file `Ecoli.txt`. Our `BAPM8` algorithm achieved the best locate time for almost all pattern lengths. For the shortest pattern $m = 12$, the algorithms EPSM and BSDM4 overcame all the other competitors. The safe shift distance is limited for `BAPM`. It is given as the number of complete bytes of the encoded pattern minus one. In practise, it means shifting by $\lfloor \frac{12}{4} \rfloor - 1 = 2$ bytes for a pattern of length $m = 12$ and thus omitting only 50 % of the input text.

At the same time, `BAPM4` achieved better result than `BAPM8` for $m = 12$. The tabulated encoded factors of length one byte are sufficient for efficient filtration in the first step of searching. The higher memory consumption of `BAPM8` is not balanced by significantly more

Table 2 `human100MB.txt`: Locate times in milliseconds. The best results are bolded.

m	BAPM4	BAPM8	EPSM	SO	BSDM4	BSDM8	SBNDMQ4	SBNDMQ8
12	**21.250**	22.030	22.031	79.531	21.375	38.578	24.796	59.234
16	16.560	**11.250**	31.562	79.547	17.890	22.766	19.531	33.656
20	14.370	**7.500**	32.031	79.546	16.171	16.375	16.688	23.906
24	12.650	**5.940**	30.171	79.547	15.047	12.828	15.141	18.312
28	11.880	**4.680**	30.421	79.844	14.266	10.891	14.031	15.343
32	11.100	**3.900**	29.876	79.562	13.969	9.875	13.250	12.828
36	11.720	**3.440**	29.938	59.421	13.641	9.296	12.796	13.359
40	10.940	**3.130**	29.734	59.437	13.343	8.828	12.812	13.359
44	10.780	**2.810**	29.672	59.453	13.141	8.671	12.828	13.375
48	10.620	**2.650**	29.922	59.422	13.000	8.562	12.750	13.375
52	10.470	**2.500**	29.875	59.438	12.859	8.516	12.766	13.343
56	10.310	**2.340**	30.296	59.453	12.672	8.468	12.781	13.360
60	9.850	**2.340**	30.250	59.438	12.546	8.391	12.750	13.359
64	9.840	**2.190**	30.219	59.453	12.500	8.375	12.796	13.360
96	9.060	**2.030**	29.734	59.453	12.093	8.343	12.750	13.328
128	8.590	**2.030**	29.078	59.438	11.860	8.375	12.812	13.359
160	7.960	**2.030**	28.266	59.453	11.781	8.390	12.781	13.313
192	7.500	**2.030**	27.703	59.422	11.656	8.391	12.781	13.343
224	7.030	**2.030**	26.938	59.453	11.578	8.406	12.828	13.328
256	6.560	**2.030**	–	–	–	–	–	–

Table 3 Block inverted index: Locate time per occurrence in milliseconds (`Ecoli.txt`, `human100MB`).

	`Ecoli.txt`			`human100MB`		
m	BAPM8	LZ77	RLCSA	BAPM8	LZ77	RLCSA
12	0.717	**0.077**	0.156	**0.087**	0.096	0.175
16	0.507	**0.099**	0.159	**0.072**	0.092	0.159
20	0.287	**0.127**	0.170	0.104	**0.092**	0.155
24	0.177	**0.158**	0.173	0.126	**0.080**	0.153
28	**0.115**	0.177	0.177	0.127	**0.074**	0.126
32	**0.087**	0.176	0.167	0.117	**0.078**	0.129
36	**0.056**	0.195	0.180	0.109	**0.101**	0.134
40	**0.044**	0.251	0.192	**0.088**	**0.088**	0.121
44	**0.043**	0.271	0.181	**0.070**	0.111	0.123
48	**0.029**	0.262	0.175	**0.074**	0.108	0.123
52	**0.030**	0.283	0.175	**0.055**	0.126	0.127
56	**0.031**	0.273	0.196	**0.058**	0.189	0.139
60	**0.044**	0.301	0.195	**0.048**	0.187	0.136
64	**0.044**	0.329	0.194	**0.044**	0.216	0.137
96	**0.044**	0.460	0.222	**0.035**	0.550	0.200
128	**0.043**	0.606	0.251	**0.042**	0.799	0.240
160	**0.044**	0.775	0.268	**0.047**	1.215	0.295
192	**0.060**	0.958	0.302	**0.053**	1.701	0.367
224	**0.060**	1.108	0.329	**0.060**	2.021	0.401
256	**0.059**	1.553	0.350	**0.057**	2.226	0.436

efficient filtration and the search speed of `BAPM8` is lower for $m = 12$. The more efficient filtration outweighs for the longer patterns where $m \geq 16$. Similar results were achieved also on the file `human100MB.txt` (see Table 2 and Figure 4). `BAPM4` achieved the best result for $m = 12$ and `BAPM8` proved to be superior for $m \geq 16$. For $m = 224$, `BAPM8` is more than four times faster than the second fastest algorithm BSDM8.

We present the comparison of different indexing methods in Figure 5. Our `BAPM8` works together with block q-gram inverted index. We performed the experiments with $q = 8$ and the size of the block 102 400 bytes. `BAPM8` (together with the inverted index) was compared with LZ77 self-index [12] and RLCSA [19]. The presented results prove that block q-gram

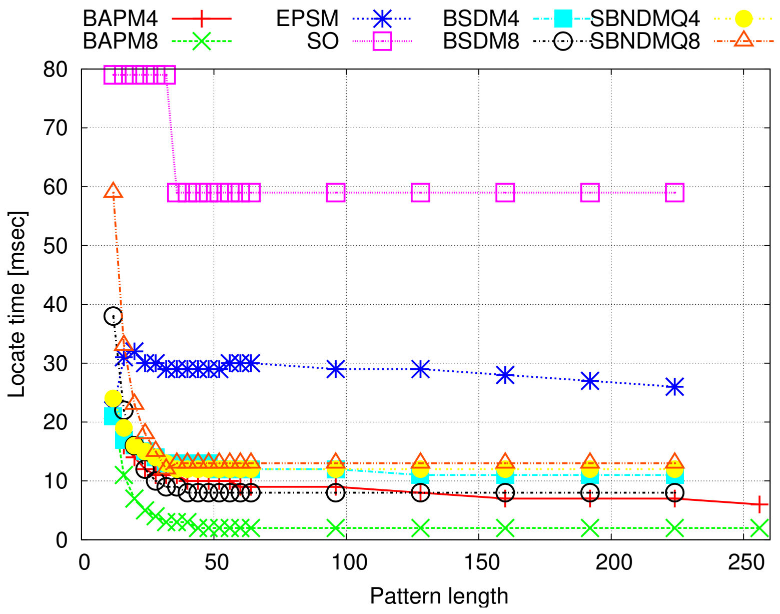


Figure 4 `human100MB.txt`: Locate time depending on the length of the searched pattern m.

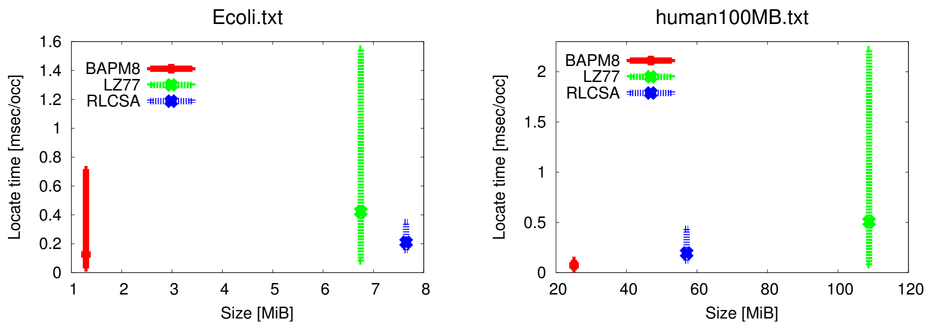


Figure 5 Comparison `BAPM8` supplemented with the block q-gram inverted index with other indexing methods. Minimum, average and maximum locate time per occurrence for different patterns (m ranging from 12 to 256) are reported. The horizontal axis presents the spaces consumption of single methods in MiB.

inverted index together with `BAPM8` represent a very good alternative to the other indexing methods, especially for sequences obtained using so-called *De Novo Sequencing* when LZ77 self-index and RLCSA cannot exploit their ability to compress highly similar sequences.

5 Conclusion and Future work

We presented a novel pattern matching algorithm, named `BAPM` (Byte-Aligned Pattern Matching), optimized for searching in encoded genomic sequences. The presented algorithm is based on two crucial properties: (i) processing at byte level of the input text; (ii) tabulating all factors of the pattern and searching for them in the filtration step. These two principles provide extraordinary efficiency of searching that was proved on real genomic data. We demonstrated that `BAPM` together with block q-gram inverted index can overcome other indexing methods and achieve locate time in the order of tens of microseconds per one occurrence.

In our future work, we aim to extend BAPM to be applicable for texts of larger alphabets (e.g., protein sequences, natural language texts). Furthermore, we intend to present a version of the algorithm for the degenerate strings (e.g., genomic sequences composed of symbols of IUPAC alphabet[6]) with its possible applications like searching for so-called Clustered-Clumps. The solution of this problem consists in an efficient data structure allowing the access to the sparse alphabet of the pattern factors in constant time.

References

1 Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, October 1992.

2 Ricardo A. Baeza-yates. Text retrieval: Theory and practice. In *In 12th IFIP World Computer Congress, volume I*, pages 465–476. Elsevier Science, 1992.

3 Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, October 1977.

4 Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1994.

5 B. Dömölki. An algorithm for syntactical analysis. *Computational Linguistics*, 3:29–46, 1964. Hungarian Academy of Science, Budapest.

6 Simone Faro and M. Oğuzhan Külekci. Fast and flexible packed string matching. *J. of Discrete Algorithms*, 28(C):61–72, September 2014.

7 Simone Faro and Thierry Lecroq. A fast suffix automata based algorithm for exact online string matching. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata: 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, pages 149–158. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

8 P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS'00, pages 390–398, Washington, DC, USA, 2000. IEEE Computer Society.

9 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. of the Thirty-second Annual ACM Symposium on Theory of Computing*, pages 397–406, New York, NY, USA, 2000.

10 R. Nigel Horspool. Practical fast searching in strings. *Software: Practice and Experience*, 10(6):501–506, 1980.

11 Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, March 1977.

12 Sebastian Kreft and Gonzalo Navarro. LZ77-like compression with fast random access. In *Proceedings of the 2010 Data Compression Conference*, DCC'10, pages 239–248, Washington, DC, USA, 2010. IEEE Computer Society.

13 Udi Manber. A text compression scheme that allows fast searching directly in the compressed file. *ACM Trans. Inf. Syst.*, 15(2):124–136, April 1997.

14 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Disc. Algorithms*, SODA'90, pages 319–327, Philadelphia, USA, 1990. Society for Industrial and Applied Mathematics.

15 Udi Manber and Sun Wu. Glimpse: A tool to search through entire file systems. In *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, WTEC'94, page 4, Berkeley, CA, USA, 1994. USENIX Association.

[6] `https://iupac.org/`

16 Gonzalo Navarro, Edleno Silva de Moura, Marden S. Neubert, Nivio Ziviani, and Ricardo A. Baeza-Yates. Adding compression to block addressing inverted indexes. *Inf. Retr.*, 3(1):49–77, 2000. `doi:10.1023/A:1009934302807`.

17 Gonzalo Navarro and Mathieu Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, CPM'98, pages 14–33, London, UK, UK, 1998. Springer-Verlag.

18 Simon J. Puglisi, W. F. Smyth, and Andrew Turpin. Inverted files versus suffix arrays for locating patterns in primary memory. In Fabio Crestani, Paolo Ferragina, and Mark Sanderson, editors, *String Processing and Information Retrieval: 13th International Conference, SPIRE 2006, Glasgow, UK, October 11-13, 2006. Proceedings*, pages 122–133. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

19 J. Sirén, N. Välimäki, V. Mäkinen, and G. Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In *Proc. of the 15th International Symposium on String Processing and Information Retrieval*, SPIRE'08, pages 164–175, Berlin, Heidelberg, 2009. Springer-Verlag.

20 Daniel M. Sunday. A very fast substring search algorithm. *Commun. ACM*, 33(8):132–142, August 1990.

21 J. Tarhio, J. Holub, and E. Giaquinta. Technology beats algorithms (in exact string matching). *CoRR*, abs/1612.01506, 2016. URL: `http://arxiv.org/abs/1612.01506`.

22 Branislav Ďurian, Jan Holub, Hannu Peltola, and Jorma Tarhio. Tuning bndm with q-grams. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 29–37, Philadelphia, USA, 2009. Society for Industrial and Applied Mathematics.

23 P. Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 1–11, Oct 1973.

Analysis of Min-Hashing for Variant Tolerant DNA Read Mapping*

Jens Quedenfeld[1] **and Sven Rahmann**[2]

1 **Chair of Theoretical Computer Science, Technical University of Munich, Munich, Germany; and**
Bioinformatics, Computer Science XI, TU Dortmund, Dortmund, Germany
`jens.quedenfeld@in.tum.de`

2 **Genome Informatics, Institute of Human Genetics, University Hospital Essen, University of Duisburg-Essen, Essen, Germany; and**
Bioinformatics, Computer Science XI, TU Dortmund, Dortmund, Germany
`Sven.Rahmann@uni-due.de`

Abstract

DNA read mapping has become a ubiquitous task in bioinformatics. New technologies provide ever longer DNA reads (several thousand basepairs), although at comparatively high error rates (up to 15%), and the reference genome is increasingly not considered as a simple string over ACGT anymore, but as a complex object containing known genetic variants in the population. Conventional indexes based on exact seed matches, in particular the suffix array based FM index, struggle with these changing conditions, so other methods are being considered, and one such alternative is locality sensitive hashing.

Here we examine the question whether including single nucleotide polymorphisms (SNPs) in a min-hashing index is beneficial. The answer depends on the population frequency of the SNP, and we analyze several models (from simple to complex) that provide precise answers to this question under various assumptions. Our results also provide sensitivity and specificity values for min-hashing based read mappers and may be used to understand dependencies between the parameters of such methods. We hope that this article will provide a theoretical foundation for a new generation of read mappers.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases read mapping, min-hashing, variant, SNP, analysis of algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.21

1 Introduction

In bioinformatics, DNA read mapping has become a basic first step of many sequence analysis tasks. Formally, one is given millions of short DNA fragments ("reads"), i.e., strings of typical length 100–300 over the DNA alphabet $\Sigma := \{A,C,G,T\}$, and a reference genome, which is a long DNA sequence (approx. $3 \cdot 10^9$ basepairs for the human genome). For each read, one seeks the (ideally unique) interval of the reference, where the read (or its reverse complement) matches best, i.e., with smallest edit distance. Leaving aside problems such as repetitive regions in the genome, structural rearrangements, split reads, etc., read mapping is thus simply a large-scale approximate string matching problem.

* This work has been supported by the DFG, Collaborative Research Center SFB 876, project C1 (`http://sfb876.tu-dortmund.de/`).

The most efficient methods today use an index data structure to quickly locate long maximal exact matches (MEMs) between each read and reference. These "seed" matches are then extended to full alignments for verification. Several popular read mappers use the FM index (a compressed representation of a suffix array based on the Burrows Wheeler Transform [5]) because of its small space requirements. Filtering with long MEMs works efficiently if the number of differences between read and reference is small, but breaks down for higher error rates, as one obtains no specific long MEMs, but many unspecific short ones.

Two trends are currently changing the landscape of read mappers. First, there are new technologies on the market that output much longer reads (e.g., up to 60 000 basepairs), but at higher error rates (10%–15%). Second, it is becoming more accepted that the human reference genome is not well represented by a single string over Σ. Each person (even each single cell) has its own individual genome, and the field of *pan-genomics* is exploring how to best represent the entirety of known genomic information of a species [11]. Most (but not all) of the genetic variation between individuals arises from *single nucleotide polymorphisms* (SNPs), which are single positions in the genome at which there exist at least two different basepair choices in the population. Other important types of variations are variable-length insertions and deletions and rearrangements of genomic regions. As index data structures based on suffix arrays struggle with these changing conditions, alternative types of indexes are being explored.

One such alternative indexing method is by *locality sensitive hashing* on q-gram sets of genomic windows. This approach has been already used for the genome assembly tools MHAP [1] and Minimap [8] as well as for the read mappers BALAUR [9] and MashMap [6]. The hashing functions can be constructed to *take variants into account* (see Section 2). We have recently designed and implemented a prototype of a variant tolerant read mapper called VATRAM [10] based on min-hashing [2]. During the design, we noted the following question, which we call the *variant indexing decision problem*: Given a reference genome (string) and a variant with its frequency p in the population, should the variant be included in the index or not? Inclusion of a variant will allow to match reads containing the variant more reliably to the correct region, at the expense of a lower sensitivity of the region for reads that do not contain the variant. The answer therefore depends of the population frequency of the variant.

The purpose of this article is to present a detailed analysis of the variant indexing decision problem for a min-hashing index. In Section 2, we present an overview of read mapping based on min-hashing. We then analyze the variant indexing problem (Section 3), moving from a simple model to a general one, yielding widely applicable results and a limit theorem. Note that our results hold not only for VATRAM, but also for other window-based min-hashing read mappers such as *BALAUR* [9].

2 Background: Read mapping with min-hashing

Let $\mathcal{Q}$ be a set of basic objects (here, the set of all 4^q DNA q-grams). Locality sensitive hashing in general (and min-hashing as a special case) is a method that assigns an integer number (hash value) $h(Q)$ to each object set $Q \subset \mathcal{Q}$, such that the probability that two sets Q, Q' obtain the same hash value $h(Q) = h(Q')$ depends on a well-defined similarity measure if the hash function h is chosen randomly among a well-defined collection of hash functions.

Min-hashing on q-grams in particular works as follows [2]. We identify each q-gram with its base-4 numerical representation after mapping A$\mapsto$0, C$\mapsto$1, G$\mapsto$2, T$\mapsto$3. This provides a natural (lexicographic) order among the q-grams. Let π be a permutation on $\mathcal{Q}$; it induces a different order on $\mathcal{Q}$. For a q-gram set $Q \subset \mathcal{Q}$, let $\min_\pi(Q)$ be the (numerical representation of the) smallest q-gram in Q according to π. Min-hashing consists of choosing a random permutation π and using $h(Q) := \min_\pi(Q)$. The following lemma states that min-hashing is locality sensitive hashing using the Jaccard coefficient as similarity value.

▶ **Lemma 1** (Min-hash property). *Given two sets $Q \subset \mathcal{Q}$, $Q' \subset \mathcal{Q}$ and the set Π of all permutations on $\mathcal{Q}$, let $\pi \in \Pi$ be a random permutation. For any $Q \subset \mathcal{Q}$, define $h(Q) := \min_\pi(Q)$. The probability that Q and Q' are hashed to the same value $h(Q) = h(Q')$ is equal to the Jaccard coefficient of Q and Q',*

$$P(h(Q) = h(Q')) = \frac{|Q \cap Q'|}{|Q \cup Q'|}. \tag{1}$$

A proof can be found in [2].

The read mapper VATRAM [10] uses min-hashing to create an index on a given reference genome. The genome is divided into overlapping windows of length w; here w should be slightly larger than the typical length n of the reads (e.g. $w = 1.4n$ [10]). The distance between two window start positions is denoted by o; we assume $o < w$, so the windows overlap. (For long reads, the reads may be divided into overlapping windows as well.) Let $a = (a_i)$ be the sequence of genome window sequences. From each window a_i, we obtain the q-grams (substrings of length q) it contains, and we denote the resulting q-gram set by Q_i. We then apply min-hashing to each window and to each read (or read window), choosing the same permutation π. The hash value of a window or read is also called its *signature value* (with respect to π). If a read's signature value is equal to $h(Q_i)$, there is a certain probability that the read originates from window a_i. However, this agreement may also just be a random hit. Therefore, several different independent permutations are used to improve sensitivity and specificity. If the number of common signature values between the read and window a_i is higher than expected by chance, the probability that the read originates from that window is high.

The index data structure efficiently maps signature values to genome windows and may itself be implemented using hashing. Note that the memory usage for the index grows linearly with the number of permutations.

In practice, choosing a random permutation among all $(4^q)!$ ones is impossible even for small q due to limitations of pseudo-random number generation with finite memory. Furthermore it is important that $\min_\pi(Q)$ can be computed efficiently. Therefore in VATRAM both q-grams and permutations are represented by $2q$-bit vectors. Applying a permutation is simulated by combining the q-gram bit-vector and the permutation bit-vector with an *exclusive or* (XOR) operation. The signature value is the smallest of the resulting values,

$$h_\pi(Q) = \min\{x \oplus \pi \mid x \in Q\}. \tag{2}$$

Using $2q$ random bits and the XOR technique instead of a true random permutation means that the pre-conditions of Lemma 1 do not hold and the min-hash property may be violated [3]. However, empirical studies have shown that in practice the XOR technique approximates the desired property well [4].

VATRAM supports single nucleotide polymorphisms where one nucleotide is replaced by another. For each known variant in window a_i of the reference genome, we may add not only the original reference q-grams to set Q_i, but also all q-grams resulting variant. In this case Q_i has a larger cardinality than sets from windows without variants.

3 Analysis of variant-tolerant min-hashing

Adding q-grams resulting from a variant increases the number of common q-grams between the reference window and a read containing the variant, so the *collision probability* that both are mapped to the same signature value becomes larger. However, for reads not containing the variant, adding new q-grams increases the size of the union in Eq. (1), so the collision probability is reduced.

The *variant indexing decision problem* asks how frequently a variant must occur in the population, such that adding the variant is beneficial on average. We provide an answer to this question under certain assumptions in each subsection. These assumptions start out strong (and unrealistic) and are successively relaxed; this organization should allow for an accessible exposition.

We always assume that the variant under consideration has a population frequency of $0 < p \leq 0.5$, i.e., the variant appears with a probability of p in a random read. (If $p > 0.5$, we would define the variant as reference and vice versa.) We use the following notation.

- The window length is w; the number of q-grams in the window is $v = w - q + 1$.
- The read length is n; the number of q-grams in the read is $m = n - q + 1$.

3.1 Basic model

Our initial assumptions are as follows.

- The entire read is contained in a single reference window a_i.
- There is a single SNP inside the window.
- The SNP occurs "far" from the ends of the read and window, such that exactly q of the q-grams are affected.
- There are no sequencing errors in the read, i.e., all other positions correspond exactly to the reference.
- Each q-gram in Q_i (including those stemming from the variant) occurs only once in window a_i.
- Only a single signature value is used.

We compare two strategies: In the Default strategy we only use the original nucleotides in the reference genome to build the index. In the SNP strategy we use both the original nucleotides and those containing the variant. For both cases, the collision probabilities P_{Default} and P_{SNP} are easily calculated.

▶ **Lemma 2.** *Under the above assumptions, with $v = w - q + 1$ and $m = n - q + 1$,*

$$P_{\text{Default}} = (1 - p) \cdot m/v + p \cdot (m - q)/(v + q), \tag{3}$$

$$P_{\text{SNP}} = m/(v + q). \tag{4}$$

Adding the variant to the index is beneficial if and only if

$$p > m/(m + v). \tag{5}$$

Proof. All collision probabilities follow from calculating the cardinalities of intersections and unions of q-gram sets and applying Eq. (1).

For the Default strategy, with a probability of $1 - p$ (variant does not occur in the read) all q-grams of the read belong to the window's q-gram set, so the collision probability is m/v. With probability p (the variant occurs in the read), the intersection's size is reduced by q and the union's size is enlarged by q, leading to a collision probability of $(m-q)/(v+q)$.

In the SNP strategy, the intersection size is always m and the union size is always $v+q$. Adding the variant is beneficial if and only if $P_{\text{SNP}} > P_{\text{Default}}$, which is equivalent to inequality (5) by the next lemma, whose proof is elementary algebra. ◀

▶ **Lemma 3.** *If the condition*

$$P_{\text{SNP}} > P_{\text{Default}} \textit{ is equivalent to } s > (1-p)\cdot d_0 + p\cdot d_1 \tag{6}$$

for some constants s, d_0, d_1 independent of p with $d_0 > s > 0$ and $d_0 > d_1 > 0$, then the condition is also equivalent to

$$p > (d_0 - s)/(d_0 - d_1). \tag{7}$$

A typical parameter configuration that produces good results for reads with a length of $n = 100$ bases is $w = 140$ and $q = 16$ [10]. For these parameters the frequency threshold is $p > 40.5\%$. Most known variants are less frequent than that; therefore the benefits of the SNP strategy are likely marginal in this simplified model. However, some of the assumptions were unrealistic, and we now consider increasingly realistic models.

3.2 Consideration of errors

So far we did not take sequencing errors or unknown variants into account. Both types of differences between the read and the reference are called *errors*. We assume that

- exactly $e < m - q$ of the q-grams in the read are affected by errors,
- the errors occur far from the SNP position,
- the q-grams produced by the errors are different from the existing q-grams in the read and in the window.

▶ **Lemma 4.** *Under the above assumptions, with $v = w - q + 1$ and $m = n - q + 1$,*

$$P_{\text{Default}} = (1-p)\cdot(m-e)/(v+e) + p\cdot(m-q-e)/(v+q+e), \tag{8}$$

$$P_{\text{SNP}} = (m-e)/(v+q+e). \tag{9}$$

Adding the variant to the index is beneficial if and only if

$$p > (m-e)/(m+v), \tag{10}$$

which is equivalent to (5) *for $e = 0$.*

Proof. The proof is similar to that of Lemma 2. Intersection sizes are reduced by e, while union sizes grow by e in comparison to Lemma 2. With $d_0 = (m-e)/(v+e)$, $d_1 = (m-q-e)/(v+q+e)$ and $s = (m-e)/(v+q+e)$, the conditions of Lemma 3 are satisfied for $e < m - q$, and (10) follows by elementary algebra. ◀

The result shows that the population frequency threshold decreases when more errors are considered. For example, if we consider the standard configuration ($n = 100$, $w = 140$ and $q = 16$) and $e = 0, q, 2q$, corresponding to no, one and two isolated errors, the thresholds are 40.5%, 32.9% and 25.2%, respectively. (In the extreme case $e = m - q - 1$, the frequency threshold becomes 8.1%. Of course, for so many errors, the indexing strategy breaks down as a whole and the variant indexing decision problem is meaningless.)

The above analysis assumed that the errors did non interfere the q-grams generated by the variant, which is unrealistic. Instead, errors will be distributed randomly, so we may assume that they affect (averaging over many reads) q-grams of the variant and q-grams outside the variant with an equal proportion. In this model, we call $\varepsilon := e/m$ the error rate. Using this modified error model has considerable effects on the threshold.

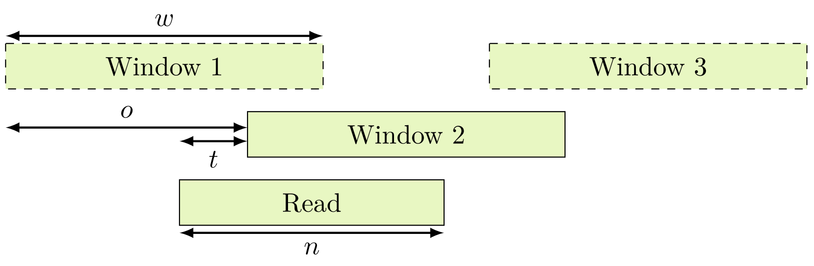


Figure 1 Visualization of the parameters n (read length), w (window length), o (window distance, $o < w$ ensures overlapping windows) and t (the number of read's q-grams located outside of the major window). Note that the number of bases located outside (shown in the visualization) is equal to the number of read's q-grams located outside if and only if $t \leq n-q+1 = m$, so the read and the major window have at least $q-1$ common positions. This conditions is always fulfilled when using realistic parameter configurations. We assume that $o > w - n + 1$, so there exist configurations where the read is not contained in a single window.

▶ **Lemma 5.** *Under the above assumptions,*

$$P_{\text{DEFAULT}} = (1-p)\cdot\frac{m(1-\varepsilon)}{v+m\varepsilon} + p\cdot\frac{(m-q)(1-\varepsilon)}{v+q+(m-q)\varepsilon}, \tag{11}$$

$$P_{\text{SNP}} = \frac{m(1-\varepsilon)}{v+q+m\varepsilon}. \tag{12}$$

Adding the variant to the index is beneficial if and only if

$$p > \frac{m}{v+m} - \frac{eq}{(v+m)(v+e+q)}. \tag{13}$$

Proof. The arguments are the same as in the proof of Lemma 4, but it is not a fixed number e that is subtracted from the intersection and added to the union, but the intersection is reduced by a factor of $(1-\varepsilon)$; the corresponding number is added to the union. Note that only the constant factor of p differs between (8) and (11) and (9) is identical to (12). The threshold (13) follows from Lemma 3; a detailed derivation is presented in Appendix A.1. ◀

In this model, the effect of e is marginal for typical parameter configurations, which is different from the model considered in Lemma 4. With the same parameters as above, the behavior for the extreme case $e = m - 1$ yields a threshold of 37.6%, which is close to the threshold of 40.5% for $e = 0$.

3.3 Consideration of partial overlaps between read and windows

So far we assumed that the read is contained entirely in one window. Indeed, this is the case if the distance o between two window starting points satisfies $o \leq w - n + 1$. Now we consider $o > w - n + 1$, where a read may overlap with two windows. We consider the window with the larger overlap (see Fig. 1) and call it the *major window*.

When we slide the read over the reference, there are o different configurations how the read overlaps two windows. For each configuration we track the number t of q-grams of the read that do *not* overlap the major window. For $w-n+1$ configurations, the major window contains the entire read and so $t = 0$. When the read moves further along the reference, t increases until the next window becomes the major window and t decreases towards zero. So there are $o - w + n - 1$ configurations with a non-zero value of t.

If $o-w+n-1$ is even, the maximum value of t that is attained is $t_{\max} = (o-w+n-1)/2$ and each value is attained twice.

If $o-w+n-1$ is odd, the maximum value of t that is attained is $t_{\max} = (o-w+n)/2$, this value is attained once and each lower $t = 1, \dots, t_{\max} - 1$ is attained twice.

For a unified consideration of intersection and union sizes in Lemma 1, we introduce a collision probability parameterized by t and two additional numbers I and U, which stand for the number of q-grams by which the intersection is reduced and the union is enlarged, respectively,

$$P_{t,I,U,0} := \frac{m-t-I}{v+t+U}.$$

As in Section 3.2, we generalize this quantity by considering q-grams affected by errors, calling again $\varepsilon := e/m$ the q-gram error rate (typical ε of interest are $\varepsilon = 0$, q/m, $2q/m$, etc.) and define

$$P_{t,I,U,\varepsilon} := \frac{(m-t-I)(1-\varepsilon)}{(v+t+U)+\varepsilon(m-t-I)}\,. \tag{14}$$

If we consider each window configuration equally likely, we may obtain average collision probabilities $P^{+}_{I,U,\varepsilon}$ by summing $P_{t,I,U,\varepsilon}$ over the relevant values of $t = 0, 1, \dots, t_{\max}$ with appropriate weights ω_t:

$$P^{+}_{I,U,\varepsilon} := \sum_{t=0}^{t_{\max}} \omega_t \cdot P_{t,I,U,\varepsilon}, \tag{15}$$

where, according to the discussion above, $t_{\max} = \lfloor (o-w+n)/2 \rfloor$, and the weights are $\omega_0 = (w-n+1)/o$, and, if $o-w+n-1$ is even, $\omega_t = 2/o$ for $1 \le t \le t_{\max}$, but if $o-w+n-1$ is odd, $\omega_t = 2/o$ for $1 \le t < t_{\max}$ and $\omega_{t_{\max}} = 1/o$.

▶ **Lemma 6.** *Under the assumptions and with the notation of this section,*

$$P_{\text{DEFAULT}} = (1-p)\cdot P^{+}_{0,0,\varepsilon} + p \cdot P^{+}_{q,q,\varepsilon},$$
$$P_{\text{SNP}} = P^{+}_{0,q,\varepsilon}.$$

Proof. The lemma is merely re-stating the arguments of the previous sections: For the DEFAULT strategy we have $U = I = 0$ if the read does not contain the variant (an event with probability $1-p$) and $U = I = q$ if the read does contain it (an event with probability p). For the SNP strategy, we always have $I = 0$ and $U = q$. ◀

Before stating a general threshold result (Sec. 3.5), we consider an even more complex model considering more than one signature value.

3.4 Consideration of signature length

So far we considered the collision probabilities for a single signature value. To improve sensitivity and specificity, we may instead use S different random permutations and consider the event where at least s out of S signature values of the read match the corresponding ones of the window. Then the collision probability $P_{t,I,U,\varepsilon}$ in (14) becomes

$$P^{(s,S)}_{t,I,U,\varepsilon} = \sum_{k=s}^{S} \binom{S}{k} \left(P_{t,I,U,\varepsilon}\right)^{k} \left(1-P_{t,I,U,\varepsilon}\right)^{S-k}. \tag{16}$$

For $s = 1$ (thereby only increasing sensitivity but not specificity), we obtain

$$P^{(1,S)}_{t,I,U,\varepsilon} = 1 - (1 - P_{t,I,U,\varepsilon})^S \,. \tag{17}$$

Correspondingly, the average (15) over window configurations generalizes to

$$P^{+(s,S)}_{I,U,\varepsilon} := \sum_{t=0}^{t_{\max}} \omega_t \cdot P^{(s,S)}_{t,I,U,\varepsilon} \tag{18}$$

with the same weights ω_t as before, distinguishing the cases when $o = w + n - 1$ is odd resp. even.

3.5 General results

The following result is the most general we present, considering errors, partial overlaps between read and windows and signature length.

▶ **Theorem 7.** *Under the assumptions and with the notation of Sec. 3.4,*

$$\begin{aligned} P_{\text{Default}} &= (1-p) \cdot P^{+(s,S)}_{0,0,\varepsilon} + p \cdot P^{+(s,S)}_{q,q,\varepsilon}, \\ P_{\text{SNP}} &= P^{+(s,S)}_{0,q,\varepsilon}. \end{aligned}$$

Adding the variant to the index is beneficial if and only if

$$p > T := \frac{P^{+(s,S)}_{0,0,\varepsilon} - P^{+(s,S)}_{0,q,\varepsilon}}{P^{+(s,S)}_{0,0,\varepsilon} - P^{+(s,S)}_{q,q,\varepsilon}} \,. \tag{19}$$

Proof. To see that Lemma 3 applies for the threshold, note that for every choice of $(n, w, o, q, \varepsilon, s, S)$, the average collision probability $P^{+(s,S)}_{0,I,U,\varepsilon}$ is a decreasing function of both I and U. ◀

We do not see a way to considerably simplify the threshold T in (19). Before presenting numerical results (Sec. 3.6), we consider a limit result for large signatures.

▶ **Theorem 8.** *Let (n, w, o, q) be given parameters and let $s := 1$. For any variant population frequency $p > 0$ and for each number e of q-grams affected by errors, there exists a signature length S such that $P_{\text{SNP}} > P_{\text{Default}}$. In other words, for each $\varepsilon := e/m$, the threshold*

$$T^{(1)} := \frac{P^{+(1,S)}_{0,0,\varepsilon} - P^{+(1,S)}_{0,q,\varepsilon}}{P^{+(1,S)}_{0,0,\varepsilon} - P^{+(1,S)}_{q,q,\varepsilon}}$$

converges to zero for increasing signature length S.

Proof. To show $P_{\text{SNP}} > P_{\text{Default}}$, it is sufficient to show that for large enough S, the inequality

$$P^{(1,S)}_{t,0,q,\varepsilon} > (1-p) \cdot P^{(1,S)}_{t,0,0,\varepsilon} + p \cdot P^{(1,S)}_{t,q,q,\varepsilon} \tag{20}$$

is satisfied for all t because P_{SNP} and P_{Default} are positively weighted sums of these terms with the same weights (ω_t), cf. (18). By (17), (20) is equivalent to

$$p > \frac{(1 - P_{t,0,0,\varepsilon})^S - (1 - P_{t,0,q,\varepsilon})^S}{(1 - P_{t,0,0,\varepsilon})^S - (1 - P_{t,q,q,\varepsilon})^S} \tag{21}$$

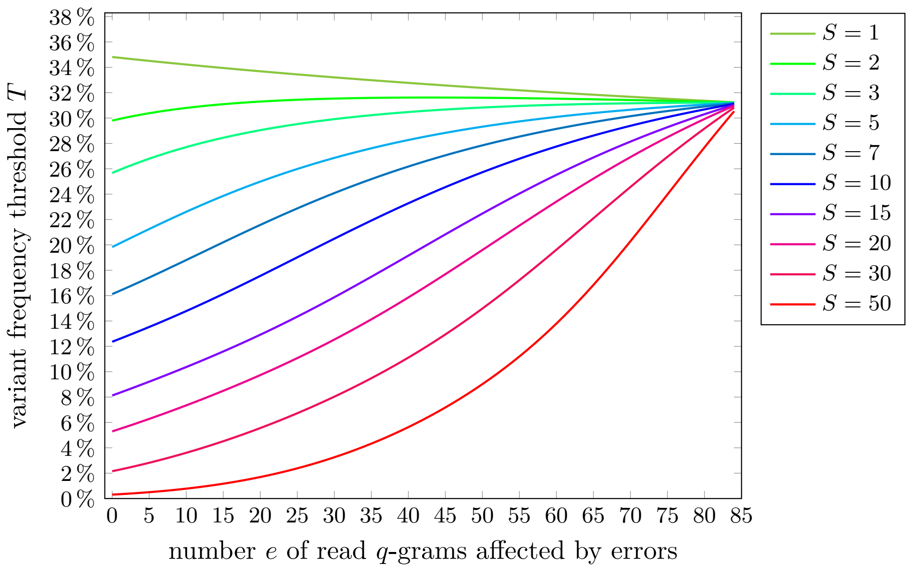


Figure 2 Variant frequency threshold as a function of the number e of erroneous q-grams for different signature lengths S (color-coded). If the probability of a SNP variant is greater than the threshold, the SNP strategy is better than the DEFAULT strategy. Parameters are $n = 100$, $w = 140$, $o = 125$ and $q = 16$, $s = 1$.

for every t. Writing $\ell_{t,I,U} := \ln(1 - P_{t,I,U,\varepsilon})$ (for fixed ε), this is equivalent to

$$p > \frac{\exp(S\ell_{t,0,0}) - \exp(S\ell_{t,0,q})}{\exp(S\ell_{t,0,0}) - \exp(S\ell_{t,q,q})} = \frac{\exp(S(\ell_{t,0,q} - \ell_{t,0,0})) - 1}{\exp(S(\ell_{t,q,q} - \ell_{t,0,0})) - 1} =: T_{t,S} \tag{22}$$

With elementary means, we verify that $P_{t,I,U,\varepsilon}$ is a decreasing function of both I and U, so

$$P_{t,0,0,\varepsilon} > P_{t,0,q,\varepsilon} > P_{t,q,q,\varepsilon} \qquad \text{and} \qquad \ell_{t,q,q} > \ell_{t,0,q} > \ell_{t,0,0}. \tag{23}$$

It follows that both numerator and denominator of $T_{t,S}$ tend towards infinity and to find the limit, we may apply De L'Hôpital's rule and find

$$\lim_{S\to\infty} T_{t,S} = \lim_{S\to\infty} \exp[S((\ell_{t,0,q} - \ell_{t,0,0}) - (\ell_{t,q,q} - \ell_{t,0,0}))] = 0, \tag{24}$$

because $\ell_{t,0,q} - \ell_{t,q,q} < 0$. So for arbitrarily small $p > 0$ and each t there exists $S(t)$ such that $p > T_{t,S(t)}$ is satisfied. Take $S := \max_{t=0,\dots,t_{\max}} S(t)$; then (20) is satisfied, proving the theorem. ◀

3.6 Numerical results

We show numerical threshold values for a realistic case, where the parameters $n = 100$ (read length), $w = 140$ (window length), $o = 125$ (window distance) and $q = 16$ result from extensive experimentation [10]. Therefore $m = n - q + 1 = 85$ and $v = w - q + 1 = 125$. Figure 2 shows the frequency threshold from (19) as a function of the number e of erroneous q-grams (recall $\varepsilon = e/m$) for different signature lengths S using $s = 1$.

For $S = 1$, the graph looks similar to the results of Lemma 5: The threshold decreased by about 5% points (because we consider the possibility of partial overlaps), and the influence of errors is marginal (about 3% points between $e = 0$ and $e = m - 1$).

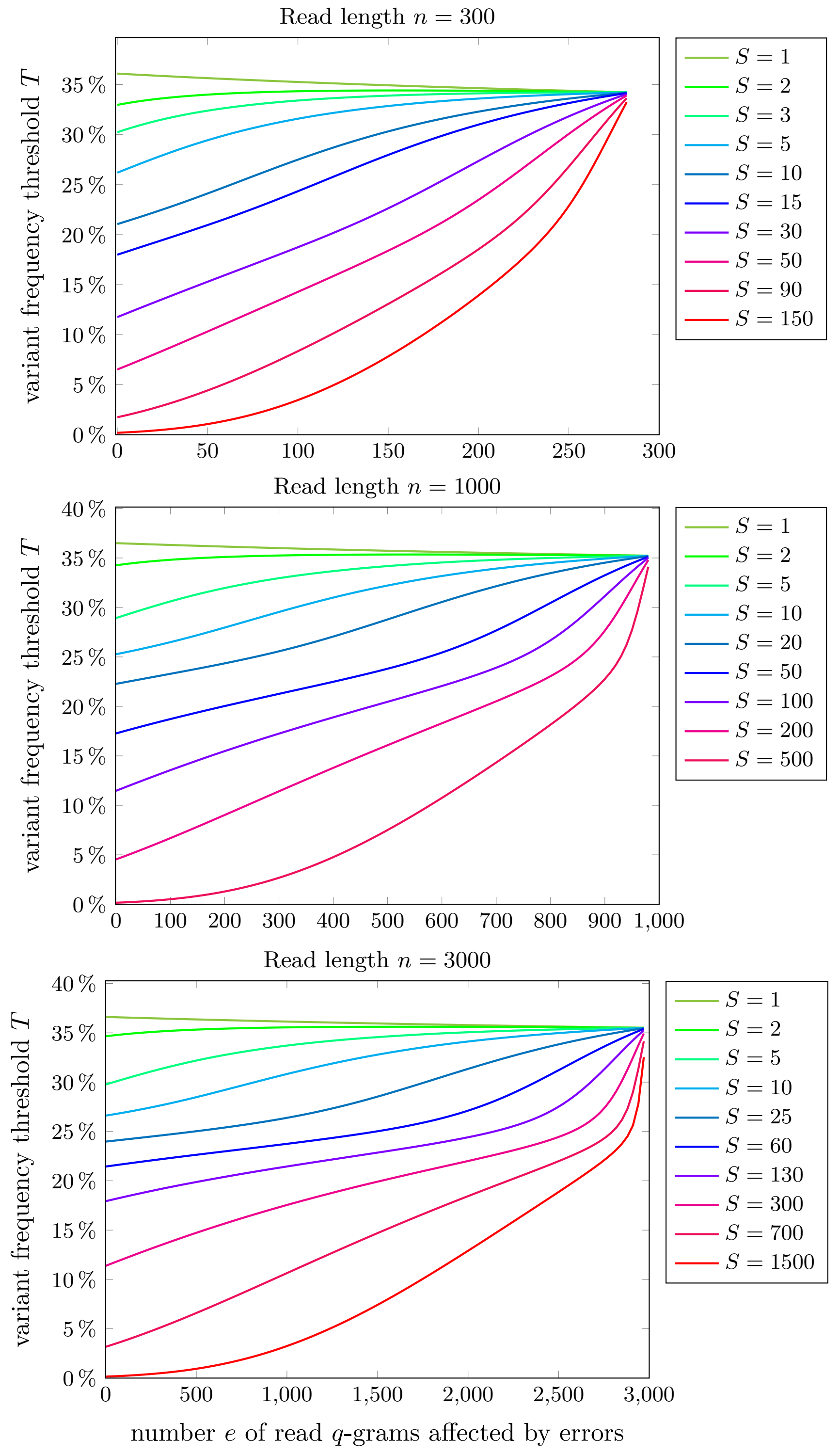


Figure 3 Variant frequency threshold as a function of the number e of erroneous q-grams for different signature lengths S (color-coded) and read lengths n (see heading of the diagrams). If the probability of a SNP variant is greater than the threshold, the SNP strategy is better than the DEFAULT strategy. Parameters are $w = 1.4n$, $o = 1.25n$ and $q = 16$, $s = 1$.

For larger signatures, the picture changes, and the threshold decreases rapidly with S (for fixed e). For error-free reads, the SNP strategy is already better if the variant's population frequency is as small as 0.3%, if use $S = 50$ permutations. As the number of errors increases, the effect of using more signatures is less important, but Theorem 8 shows that enough signature values will always favor the SNP strategy.

Figure 3 shows the frequency threshold for longer reads ($n \in \{300, 1000, 3000\}$). The window length and distance are fit to the read length, i.e. $w = 1.4n$ and $o = 1.25n$, the q-gram length is still 16. The memory consumption increases linearly with S/o, so for longer reads we can increase the signature length S to keep the memory consumption constant. Therefore the maximal signature length plotted in figure 3 is $S = n/2$. If the ratio of q-grams affected by errors is low, then the frequency threshold is almost independent of the read length. If much more than the half of q-grams are affected by errors, then the frequency threshold decreases significantly with the read length. This is especially useful for very long reads with high rates produced by third-generation sequencing machines.

4 Discussion and conclusion

As third-generation sequencing techniques produce ever longer reads with (comparatively) high error rates and the human reference genome is about to replaced by a pan-genome reference, the community is considering alternatives to an FM index for representing the human genome. A fundamental question is whether adding known variants, and in particular SNPs, which represent about 90% of the known variants, should be added to a given index data structure. We investigated this question for min-hashing, a particular form of locality sensitive hashing and found practical decision rules that depend on the population frequency p of a SNP. Theorem 8 shows that (under the right circumstances) it can be beneficial to add even rare variants to a min-hashing index.

Assumptions

Our calculations are based on several assumptions that we did not relax during the development of Theorem 7:

1. All q-grams resulting from a SNP or errors are different from all other q-grams in the read or in the window.
2. We consider each SNP in isolation and assume that exactly q of the q-grams in the read are affected.
3. Errors affect q-grams of the variant and in the remaining part of the read with equal proportion ε.

We consider these assumptions reasonable in practice for the following reasons. While we cannot rule out the possibility that one of the erroneous q-grams is equal to another q-gram in the read, such an event occurs rarely if $v \ll 4^q$, which is always the case in real applications. The second assumption can be relaxed in our framework by choosing different values for U and I in (16) than 0 and q to model specific configurations of SNP positions inside a window. One could even compute weighted averages over all such configurations for a given SNP rate, say 0.5%, across the genome. We here decided to focus on the simple case of a single isolated SNP, which is a case common enough to be relevant. The proportional error distribution may be criticized as averaging early over important effects such as clumping of affected q-grams, but we found that the results obtained with this model match extensive simulations performed in the context of the development of VATRAM (data not shown; [10]).

Sensitivity and specificity analysis

We focused on the population frequency threshold of a variant above which it becomes beneficial to add its q-grams to the index. Of course, our formulas for P_{SNP} and P_{DEFAULT} may also be used to compute sensitivity and specificity values of the min-hashing indexing approach in the first place (see [1] for first arguments). We may define the sensitivity P^* as the desired collision probability (i.e., the probability to find the correct window for a read). Choosing the optimal strategy for each variant depending on its population frequency, we have $P^* = \max\{P_{\text{SNP}}, P_{\text{DEFAULT}}\}$. A sensitivity close to 1 is desirable and we may optimize free parameters (w, o, q, s, S) to achieve a desired sensitivity. On the other hand, it is in our best interest to keep the number of false positive collisions (i.e., of collisions resulting from random coincidence of signature values) as low as possible. By estimating the size distribution of the intersection of the q-gram set of a random read with a random window, we obtain a null collision probability P^0 and the expected number $E = P^0 \cdot G/o$ of colliding windows for a read, where G is the genome size. It follows that Ew basepairs must be aligned to the read for verification. It is mainly the choice of q, S and s that influences P^0, and choosing sufficiently large values will decrease P^0. On the other hand, the memory requirements of the index are proportional to $(4^q + G/o) \cdot S$, so we are interested in keeping this quantity as small as possible. In the future, we aim to use the results presented in this article to optimize the min-hashing parameters to achieve optimum sensitivity with a tolerable false positive rate for a given amount of available memory.

Acknowledgments. This work has been supported by the DFG, Collaborative Research Center SFB 876, project C1 (`http://sfb876.tu-dortmund.de/`). We thank all members and advisors of Project Group 583 [7] for their input.

References

1 Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P. Drake, Jane M. Landolin, and Adam M. Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.*, 33(6):623–630, 2015. Corrigendum in Nat. Biotechnol. 33(10), 1109 (2015).

2 Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)*, pages 21–29. IEEE, 1997.

3 Andrei Z. Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the 30th annual ACM symposium on Theory of computing (STOC)*, pages 327–336. ACM, 1998.

4 Matthew Casperson. Minhash for dummies. `http://matthewcasperson.blogspot.de/2013/11/minhash-for-dummies.html`, November 2013.

5 P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52(4):552—581, 2005.

6 Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. In *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer, 2017.

7 Benjamin Kramer, Jens Quedenfeld, Sven Schrinner, Marcel Bargull, Kada Benadjemia, Jan Stricker, and David Losch. VATRAM – VAriant Tolerant ReAd Mapper. Technical report, Project Group PG583, Computer Science, TU Dortmund, Germany, 2015.

8 Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, page btw152, 2016.

9 Victoria Popic and Serafim Batzoglou. Privacy-preserving read mapping using locality sensitive hashing and secure kmer voting. *bioRxiv*, page 046920, 2016.
10 Jens Quedenfeld and Sven Rahmann. Variant tolerant read mapping using min-hashing. *arXiv*, 1702.01703, 2017.
11 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, Oct 2016. Online first,. `doi:10.1093/bib/bbw089`.

A Appendix: Proof details

A.1 Detailed calculation for Lemma 5

In Lemma 5,

$$P_{\text{SNP}} \geq P_{\text{DEFAULT}}$$

is by Lemma 3 equivalent to

$$\begin{aligned}
p &> \frac{d_0 - s}{d_0 - d_1} \\
&= \frac{\frac{m-e}{v+e} - \frac{m-e}{v+e+q}}{\frac{m-e}{v+e} - \frac{m-q-e+eq/m}{v+q+e-eq/m}} \\
&= \frac{(m-e)\cdot\frac{(v+e+q)-(v+e)}{(v+e)(v+e+q)}}{\frac{(m-e)(mv+mq+mv-eq)-(m^2-mq-me+eq/m)(v+e)}{(v+e)(mv+mq+me-eq)}} \\
&= \frac{\frac{q(m-e)}{v+e+q}}{\frac{m^2q-meq+qmv-eqv}{mv+mq+me-eq}} \\
&= \frac{(m-e)(mv+mq+me-eq)}{(v+e+q)\,(m^2-me+mv-ev)} \\
&= \frac{m(v+e+q)-eq}{(v+e+q)(m+v)} \\
&= \frac{m}{m+v} - \frac{eq}{(v+m)(v+e+q)}\,,
\end{aligned}$$

which is the final statement of Lemma 5. ◀

Efficient and Accurate Detection of Topologically Associating Domains from Contact Maps[*]

Abbas Roayaei Ardakany[1] and Stefano Lonardi[2]

1 **Department of Computer Science, University of California, Riverside, CA, USA**
2 **Department of Computer Science, University of California, Riverside, CA, USA**

Abstract

Continuous improvements to high-throughput conformation capture (Hi-C) are revealing richer information about the spatial organization of the chromatin and its role in cellular functions. Several studies have confirmed the existence of structural features of the genome 3D organization that are stable across cell types and conserved across species, called *topological associating domains* (TADs). The detection of TADs has become a critical step in the analysis of Hi-C data, e.g., to identify enhancer-promoter associations. Here we present East, a novel TAD identification algorithm based on fast 2D convolution of Haar-like features, that is as accurate as the state-of-the-art method based on the directionality index, but 75-80$\times$ faster. East is available in the public domain at `https://github.com/ucrbioinfo/EAST`.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems I.4.6 Image Segmentation J.3 Life and Medical Sciences

Keywords and phrases Chromatin, TADs, 3D genome, Hi-C, contact maps

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.22

1 Introduction

Recent studies have revealed that genomic DNA is not arbitrarily packed into the nucleus. The chromatin has a well organized and regulated structure in accordance to the stage of the cell cycle and environmental changes [15, 16]. The structure of chromatin in the nucleus plays a critical role in gene expression, epigenetic organization, and DNA replication, among others [7, 6, 18, 17].

With the advent of genome-wide DNA proximity ligation (Hi-C), life scientist have shed new light on the way that chromatin folds and its relation to cellular functions [13, 1, 2, 10]. The analysis of Hi-C data has revealed surprising new findings including the discovery of new structural features of chromosomes such as topologically associating domains [7] and chromatin looping [17].

Topological associating domains (TADs) are large, megabase-sized contiguous local chromatin interaction domains that have a high average interaction within and a low average interaction with their surrounding regions. Because of the role that TADs play in cellular functions they have been widely explored since their discovery. TADs are stable across different cell types and highly conserved across species [7]. TADs tend to interact with each other in a tree-like structure and form a hierarchy of domains-within-domains (metaTAD),

[*] This work was partially supported by NSF IIS-1526742.

which can scale up to the size of chromosomes [9]. metaTADs show correlation with genetic and epigenomic features. TAD boundaries are enriched for the insulator binding protein CTCF, housekeeping genes, transfer RNAs and histone modifications [7, 8]. More importantly, enhancers tend to interact with gene promoters within the same TAD [11]. Disruption of TAD boundaries can affect the expression of nearby genes and lead to developmental disorders or cancer [14].

Several methods have been developed to identify TADs genome-wide. Dixon *et al.* were the first group to define and identify TADs [7]. In their seminal work, they proposed an identification method based on the *directionality index* (DI) which measures the frequency of interaction of a genomic locus with a fixed-sized neighborhood. Drastic changes of the DI score are expected at TAD boundaries where the region tends to have a high rate of both upstream and downstream interactions.

Filippova *et al.* [8] introduced a single parameter, two-step dynamic programming method. Assuming that there exist a few characteristic resolutions across which TADs are similar, they identify a set of non-overlapping domains that are persistent across the resolutions.

Crane *et al.* [4] proposed a method based on the *insulation index* (IS). For each chromosome segment, IS score is the average number of interactions that cross the segment in a pre-specified size neighborhood. Given that interactions tend to be isolated within TADs, IS local minima are expected to occur at TAD boundaries. The IS score can be computed efficiently by sliding a window across the diagonal of the contact matrix and computing the average number of interactions that fall inside the window.

Chen *et al.* [3] translated the TAD identification problem into a graph segmentation/-clustering problem. In this method, domains at different scales are identified by running the spectral graph cuts algorithm recursively until the connectivity of the graph reaches some predefined threshold.

In this paper, we present a novel TAD calling algorithm called East (for "Efficient and Accurate Summed-area-table-based TAD calling") that takes advantage of fast 2D convolution. Experimental results show that East is as accurate in detecting TADs as the DI method [7], which is accepted as the state-of-the art algorithm. East is however, 75-80$\times$ faster than DI.

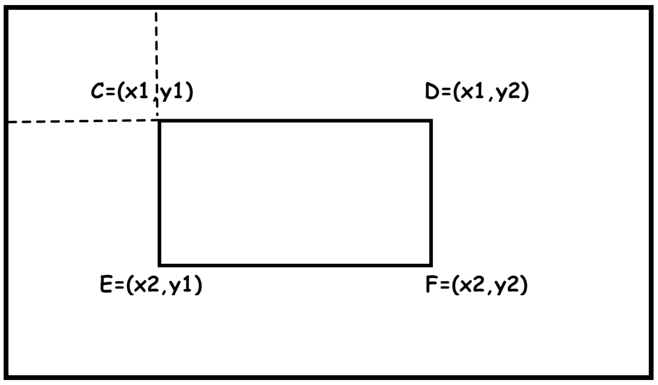
2 Methods

Each chromosome is segmented into evenly sized fragments, where the size of the segment is based on the resolution of the data. In a Hi-C *contact map* (or *interaction matrix*) A, each entry $A[i, j]$ represents the number of times segments i and j are observed together in a DNA proximity ligation experiment. Larger values of $A[i, j]$ indicate closer loci i and j in 3D space inside the nucleus. Segments that are close in genomic 1D distance tend to form dense areas which can be seen as isolated high frequency blocks along the matrix diagonal, namely, TADs. TADs have high intra-frequency within and low inter-frequency with their neighboring blocks. The aim is to identify TADs efficiently and accurately.

We propose an algorithm called East that utilizes rectangular Haar-like features [21] and dynamic programming to identify TADs. Genomic regions are scored based on an objective function that measures their likelihood of containing a TAD with respect to the characteristics mentioned above. We use Haar-like features to describe such a scoring function.

2.1 Summed area table and Haar-like features

A *Haar-like feature* is a set of adjacent rectangular regions each of which has a certain weight. Weights of rectangular regions indicate certain characteristics of a particular area of



$$\sum_{\substack{x_1 \le x \le x_2 \\ y_1 \le y \le y_2}} A(x,y) = A_{\mathrm{SAT}}(C) + A_{\mathrm{SAT}}(F) - A_{\mathrm{SAT}}(D) - A_{\mathrm{SAT}}(E)$$

Figure 1 If the summed area table A_{SAT} is available, computing the sum of values in any rectangular region takes $O(1)$ time.

the image. By convolving Haar-like features, i.e., by computing the weighted sum of pixel values for a particular location, we obtain a value that represents how well a region (window) satisfies the characteristics we are looking for. To compute the weighted sum efficiently we use the summed area table.

A *summed area table* (SAT), also known as *integral image* in computer vision, is a data structure used for efficiently calculating the sum of values in a rectangular region. By precomputing the summed area table one can obtain the sum of values in any arbitrary rectangular region using only a constant number of operations. SAT was first introduced to computer graphics in 1984 by Frank Crow [5] and later to computer vision in 2001 by Lewis [21] in a popular face detection framework called Viola-Jones. The value of a point (x, y) in a summed area table A_{SAT} is the sum of all pixels above and to the left of that point in the original grid A, including the (x, y) point itself.

$$A_{\mathrm{SAT}}(x,y) = \sum_{x' \le x,\ y' \le y} A(x', y') .$$

Since the value of each point in the SAT can be computed based on the values of neighboring points, the formula can be rewritten as

$$A_{\mathrm{SAT}}(x,y) = A(x,y) + A_{\mathrm{SAT}}(x-1,y) + A_{\mathrm{SAT}}(x,y-1) - A_{\mathrm{SAT}}(x-1,y-1) .$$

Given the summed area table, computing the sum of values in an arbitrary size rectangular region can be done in $O(1)$ time (see Figure 1).

2.2 TAD objective function

To score TADs we need to define a function f that quantifies the quality of an arbitrary region along the matrix diagonal with respect to the following properties:

1. The average frequency inside the region must be “high”.
2. The average frequency with the neighborhood must be “low”.
3. The average frequency between start and end segments of the region must be higher than the average frequency inside the region.

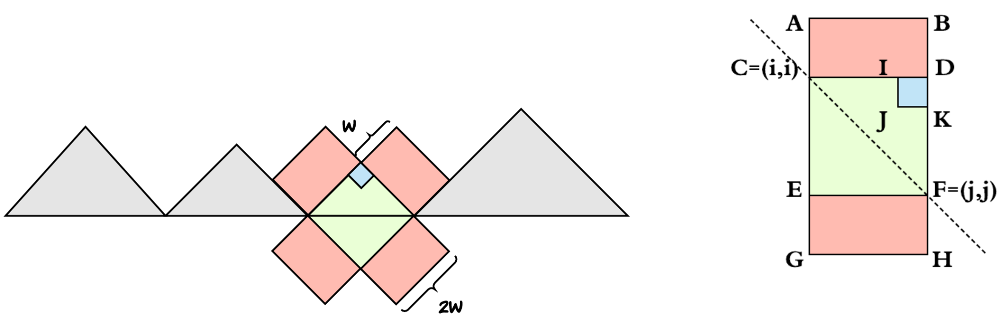


Figure 2 Objective function f. (LEFT) Representation of a TAD of size $2w$. High interaction frequency expected inside the TAD's domain (green) while low interaction frequency is expected between the TAD and surrounding domains (red) (RIGHT) Coordinates of Haar-like representation of a TAD.

The last property derives from the fact that TADs are the result of a compact locality or loop formation in the chromatin. To explain the design of the objective function f we refer to Figure 2, where different colors indicates different weighting. The area in green color is the region we expect to have a high frequency of interaction (intra-frequency), as opposed to the area in red where lower frequency is expected (inter-frequency). The corner region which is colored in blue in Figure 2 has a higher weight in order to account for the last property in the list above. Using the SAT data structure, function f can be computed as follows.

$$f([i,j]) = \frac{CDEF^{\diamond} - \alpha \cdot (ABGH^{\diamond} - CDEF^{\diamond}) + \beta \cdot IDJK^{\diamond}}{\mathcal{N}}$$

where $CDEF^{\diamond}$, $ABGH^{\diamond}$ and $IDJK^{\diamond}$ represent the sum of pixel values inside the rectangular regions $CDEF$ (defined by interval $[i,j]$), $ABGH$ and $IDJK$ respectively, which can be computed in $O(1)$ time from the SAT of the interaction matrix A. Parameters α and β are dataset-independent, and they can be determined experimentally. Parameter $\mathcal{N}$ is a normalization factor discussed in Subsection 3.1.

2.3 Finding the optimal set of domains

Given a $n \times n$ interaction matrix A, the problem of TAD identification is an optimization problem aimed at identifying the set of contiguous non-overlapping domains for which the

$$\sum_{d_i \in D} f(d_i)$$

is maximized, where $D = \{d_i | d_i = [s_i, e_i]\}$ is a set of non-overlapping intervals, i.e., $e_j < s_i$ or $e_i < s_j$ for $i \neq j$.

We use dynamic programming to solve this optimization problem. The optimal solution $OPT(i)$ for the sub-problem $[1,i]$ can be expressed by following recurrence relation

$$OPT(i) = \max_{0 \le k \le i-1} \{OPT(k) + f([k+1,i])\}.$$

By gradually increasing the size of the sub-problem and keeping track of the set of extracted domains, the optimal set of TADs for the entire interaction matrix can be computed. As we grow the size of the sub-problem, for each bin i, we need to find the optimal location to break the sub-problem $[1, i]$ into a sub-problem $[1, k]$ and a domain $d = [k + 1, i]$. The overall time-complexity is $O(n^2)$, where n is the number of bins/segments.

If we do not allow TADs to be larger than L, the optimal break point for a sub-problem $[1, i]$ can always be found in the interval $[max\{i - L, 0\}, i - 1]$. Therefore, the overall time complexity decreases to $O(nL)$.

▶ **Theorem 1.** *Let $D^* = \{[a_1, a_2], [a_2, a_3], \ldots, [a_{s-1}, a_s]\}$ be an optimal set of domains for the interaction matrix A for which*

$$\sum_{d_i \in D^*} f(d_i)$$

is maximized. Then,

$$OPT^*(n) = OPT(n)$$

where

$$OPT^*(i) = \max_{max\{i-L,0\} \leq k \leq i-1} \{OPT^*(k) + f([k + 1, i])\} .$$

Proof. We prove the theorem by induction. For the base case $OPT^*(a_1) = OPT(a_1) = 0$. Now, suppose $OPT^*(a_{i-1}) = OPT(a_{i-1})$ then we have

$$\begin{aligned} OPT^*(a_i) &= OPT^*(a_{i-1}) + f([a_{i-1} + 1, a_i]) \\ &= OPT(a_{i-1}) + f([a_{i-1} + 1, a_i]) \\ &= OPT(a_i) \text{ for } k = a_{i-1} \end{aligned}$$

where k satisfies the inequality $\max\{a_i - L, 0\} \leq k \leq a_i - 1$. ◀

3 Experimental results

We performed the analysis on Hi-C data for two mouse cell types (cortex and embryonic stem cell), and one human cell type (embryonic stem cell) at bin resolution of 40kb. The Hi-C data was obtained from [7].

3.1 Parameter settings

In addition to α, β and L, East relies on two additional parameters. The first is the minimum quality threshold τ that is used to filter out low-quality TADs. If we assume that TAD quality scores are distributed according to a Gaussian distribution, we define the threshold $\tau = \mu - \sigma$ where μ and σ are the mean and standard deviation of the distribution of scores. Observe that parameter τ can be computed from the analysis of the dataset.

The second parameter is the normalization parameter $\mathcal{N}$ for the function f. Since the quality measure f is proportional to the sum of interactions inside the domains, f grows as the TAD size increases. Figure 3 illustrates how the sum of interactions inside a domain grows as the size of the TAD increases for the three datasets used in the experimental results below and for a synthetic interaction matrix. In the synthetic data, the number of interactions was set to be inversely proportional to the genomic distance. For purpose of comparison, the sum of interactions is normalized by the sum of the largest domain.

Observe that the curve for the mouse embryonic data roughly matches the curve for the synthetic data. This suggests that the average interaction frequency of two loci in the

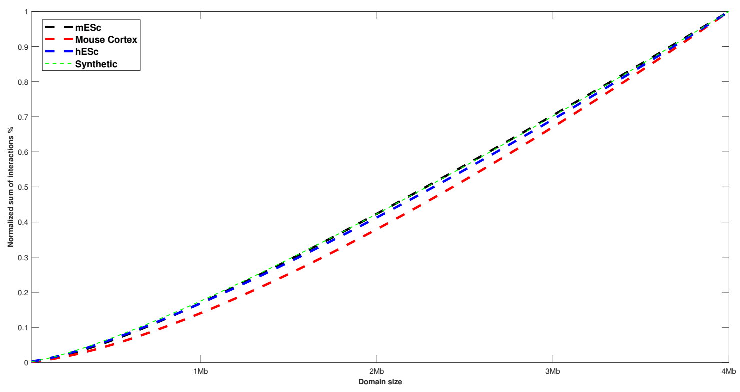


Figure 3 Growth of the quality measure f as the size of the TAD increases on the three datasets used in the experimental results and for a synthetic interaction matrix (see text).

mESC dataset is inversely proportional to their genomic distance. The growth function of the synthetic data can be estimated by $(n/L)^{1.2}$ where L is the largest domain size we are evaluating.

Also observe the hESC and mouse cortex curves are slightly different from the curve for the synthetic data, and they can be estimated by $(n/L)^{1.36}$ and $(n/L)^{1.4}$ respectively. We experimentally determined that as the curves diverge from the curve for the synthetic data, the normalization factor needs to adjusted accordingly. We set $\mathcal{N} = n^{0.4}$, $\mathcal{N} = n^{0.43}$ and $\mathcal{N} = n^{0.38}$ for hESC, mESC and mouse cortex, respectively. Parameters α and β were optimized experimentally to values $\alpha = 0.2$ and $\beta = 0.2$, and they are dataset-independent.

3.2 Comparison with existing methods

Based on the availability and popularity of TAD calling methods, we decided to compare East with the directionality index method [7], insulation score method [4] and multiscale method in [8]. We hereafter refer to these methods as DI, INS and MR respectively.

East, DI, INS and MR were ran on an Intel Core-i7 2.7GHz CPU with 16GB of memory. For the DI method we ran the experiments with posterior marginal probability threshold 0.99 and up/downstream span size of 2Mb (default parameters according to [7]). For the INS method, we set the insulation delta span to 200kb and the insulation square size to 500kb. For the MS method, we set the highest resolution parameter to 0.5.

In our experiment we investigated the enrichment of epigenetic characteristics of chromatin near the TAD boundaries. Although the mechanism behind the formation of TADs and their role in gene regulation are not fully understood, multiple studies have shown that some proteins and histone marks are enriched at the TAD boundary regions, implying that these boundaries play a role in gene transcription. As it was done in other studies [8, 20, 3], we can therefore use these genomic markers to evaluate the quality of the computed TADs.

To produce enrichment plots, we used each method to determine the boundary locations of TADs. Then, the frequency of each marker was calculated in 10kb bins in a window of 1Mb centered at the TAD boundaries. Each plot show the distribution of specific markers for each tool in the region centered at the TAD boundaries.

For mouse cortex and stem cells we evaluated the enrichment of transcription factor CTCF, promoter related marks RNA Polymerase II and H3K4me3, and enhancer-related histone modification H3K27ac. This marker data was collected from [19]. For human stem

Table 1 Running time of East, INS, MS and DI on the three datasets used in this work.

	hESC	mESC	Cortex
East	58s	50s	48s
INS	52s	44s	42s
DI	4,721s	3,845s	3,628s
MS	762s	545s	520s

cells we assessed the enrichment of CTCF near TAD boundaries. The CTCF data was obtained from [12].

Figure 4 shows that CTCF binding sites are almost twice as enriched near the TAD boundaries than the surrounding regions, suggesting that TAD boundaries are associated with insulator genomic regions and their mediator protein CTCF. Figure 5 and Figure 6 show that promoter marks RNA Polymerase II and H3K4me3 peak within the TAD boundaries for both mouse cortex and embryonic stem cells. Observe in Figure 7 that histone modification mark H3K27ac is highly enriched around TAD boundaries in mouse embryonic stem cells but not in mouse cortex cells. Also observe in Figure 8 that enhancer marks are highly depleted around TAD boundaries in mouse cortex cells but not in mouse embryonic stem cells.

Overall, observe in Figures 4–8 that the blue curve for East is almost always higher than the other three tools, suggesting that our tool generates TADs with very accurate boundaries. The closest competitor is DI (green curve), but East is significantly faster than DI.

We compared the running time of East with that of DI, MS and INS on Hi-C data for human embryonic stem cells, mouse embryonic stem cells and mouse cortex [7]. Table 1 shows that East and INS are comparable in speed, MS is 10–14× slower, DI is 75-80× slower.

Figure 9 illustrates the size distribution of TADs for all four methods for the human embryonic stem cells. The numbers of TADs extracted by East, DI, MS and INS are 2229, 2429, 12427 and 4708 respectively. Observe that East and DI roughly produce the same size distribution.

In summary, these experimental results show that while East can identify the TAD boundaries as accurately as the best method (DI), but it is much more time efficient.

4 Conclusion

In this paper, we introduced an efficient algorithm called East, to accurately identify topological associating domains in chromatin from interaction matrices obtained from high-throughput chromosome conformation capture (Hi-C). East can be downloaded from `https://github.com/ucrbioinfo/EAST`.

We performed a comparative evaluation of East on Hi-C data for human stem cells, mouse stem cells and mouse cortex cells. We showed that our algorithm extracts TADs as accurately as the state-of-the art. TADs identified by East show substantial enrichment of various epigenetic modification factors at their boundaries, confirming similar findings in previous studies. By comparing the running time of East with the other published methods, we showed that our method is very time efficient. For a given Hi-C dataset, the only parameter in East that might need to be tuned by the user is the normalization factor for which we have given some guidance in Subsection 3.1.

The framework we presented here for TAD identification is based on fast 2D-convolution of Haar-like features. We believe that this framework could be adapted to other chromatin feature detection problems such as chromatin loops [17]. We also plan to extend our work to efficiently identify chromatin features at arbitrary scales.

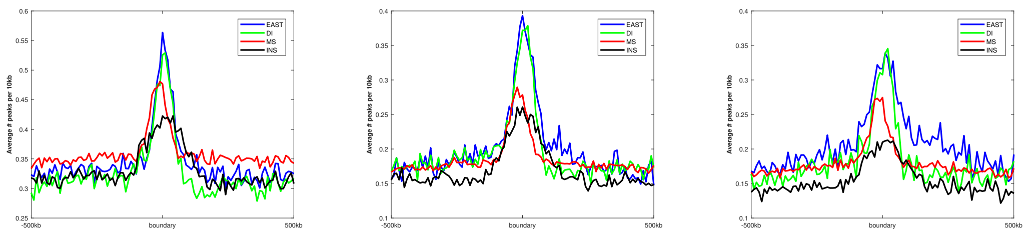


Figure 4 CTCF enrichment in human embryonic stem cells, (left) mouse embryonic stem cells (center) and mouse cortex cells (right).

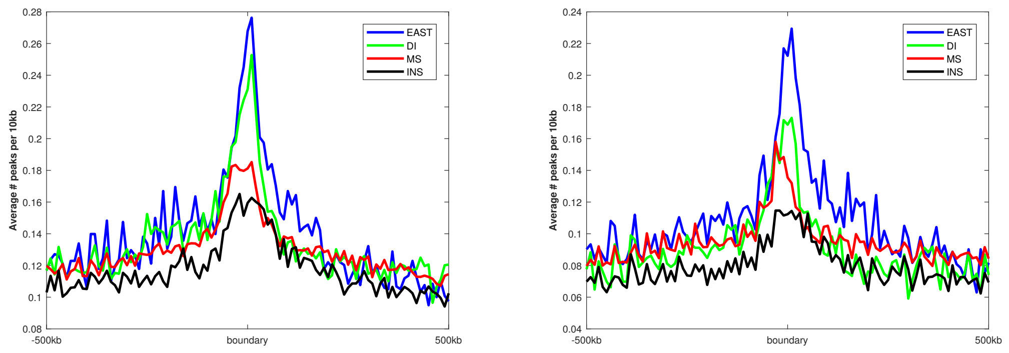


Figure 5 H3K4me3 enrichment in mouse embryonic stem cells (left) and mouse cortex cells (right).

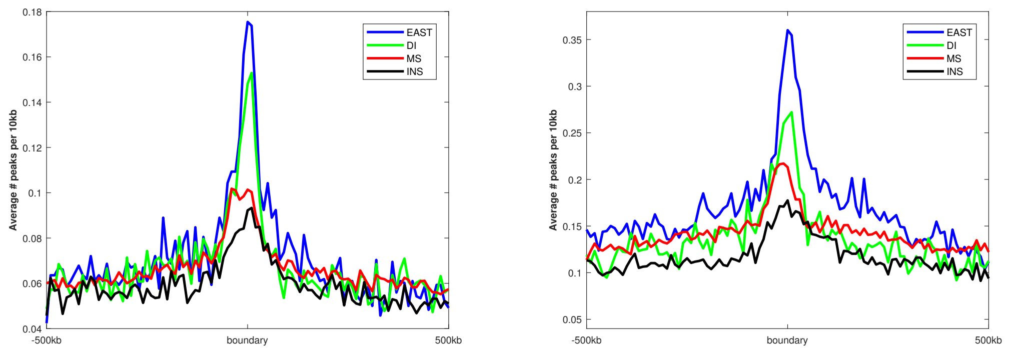
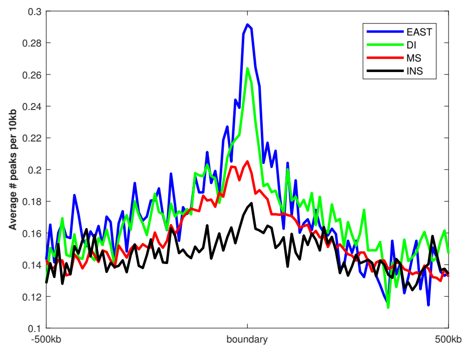


Figure 6 polII enrichment in mouse embryonic stem cells (left) and mouse cortex cells (right).



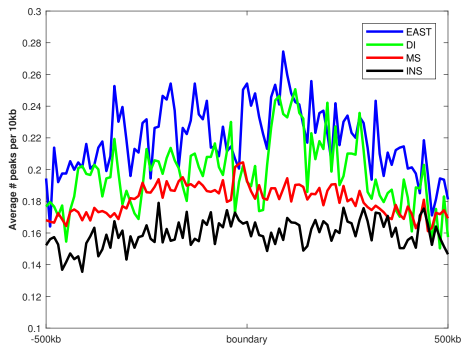
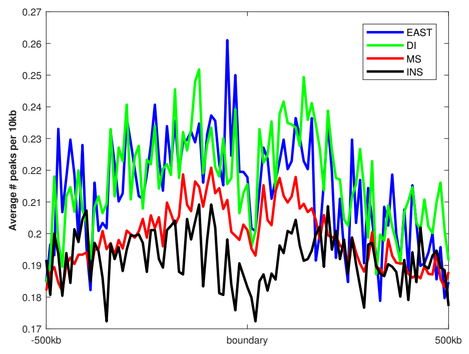


Figure 7 H3K27ac enrichment in mouse embryonic stem cells (left) and mouse cortex cells (right).



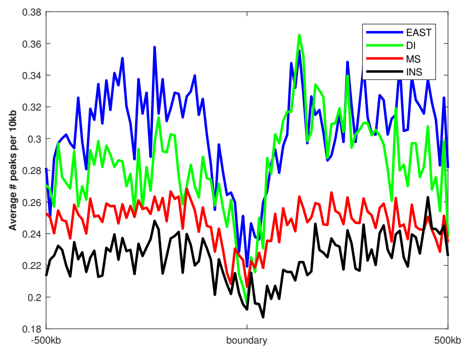


Figure 8 Enhancer enrichment in mouse embryonic stem cells (left) and mouse cortex cells (right).

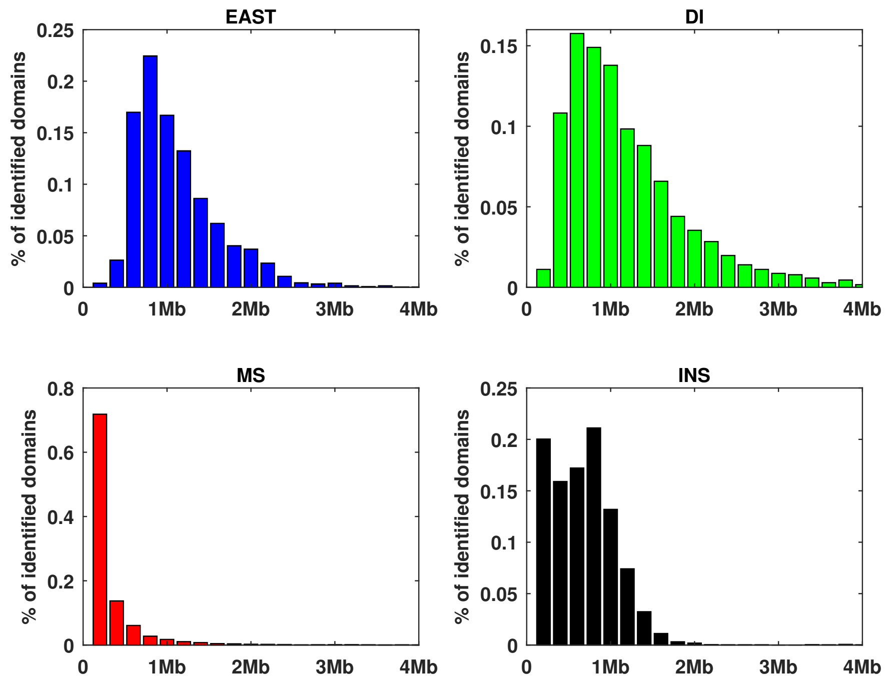


Figure 9 Comparison of the distribution of TAD size.

References

1 Giacomo Cavalli and Tom Misteli. Functional implications of genome topology. *Nat. Struct. Mol. Biol.*, 20(3):290–299, 5 March 2013.

2 Haiming Chen, Jie Chen, Lindsey A. Muir, Scott Ronquist, Walter Meixner, Mats Ljungman, Thomas Ried, Stephen Smale, and Indika Rajapakse. Functional organization of the human 4D nucleome. *Proc. Nat'l Acad. Sci. USA*, 112(26):8002–8007, 30 June 2015.

3 Jie Chen, Alfred O. Hero, 3rd, and Indika Rajapakse. Spectral identification of topological domains. *Bioinformatics*, 32(14):2151–2158, 15 July 2016.

4 Emily Crane, Qian Bian, Rachel Patton McCord, Bryan R. Lajoie, Bayly S. Wheeler, Edward J. Ralston, Satoru Uzawa, Job Dekker, and Barbara J. Meyer. Condensin-driven remodelling of X chromosome topology during dosage compensation. *Nature*, 523(7559):240–244, 9 July 2015.

5 Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH'84, pages 207–212, New York, NY, USA, 1984. ACM.

6 Jesse R. Dixon, Inkyung Jung, Siddarth Selvaraj, Yin Shen, Jessica E. Antosiewicz-Bourget, Ah Young Lee, Zhen Ye, Audrey Kim, Nisha Rajagopal, Wei Xie, Yarui Diao, Jing Liang, Huimin Zhao, Victor V. Lobanenkov, Joseph R. Ecker, James A. Thomson, and Bing Ren. Chromatin architecture reorganization during stem cell differentiation. *Nature*, 518(7539):331–336, 19 February 2015.

7 Jesse R. Dixon, Siddarth Selvaraj, Feng Yue, Audrey Kim, Yan Li, Yin Shen, Ming Hu, Jun S. Liu, and Bing Ren. Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature*, 485(7398):376–380, 17 May 2012.

8 Darya Filippova, Rob Patro, Geet Duggal, and Carl Kingsford. Multiscale identification of topological domains in chromatin. In *Algorithms in Bioinformatics*, pages 300–312. Springer, Berlin, Heidelberg, 2 September 2013.

9 James Fraser, Carmelo Ferrai, Andrea M. Chiariello, Markus Schueler, Tiago Rito, Giovanni Laudanno, Mariano Barbieri, Benjamin L. Moore, Dorothee C. A. Kraemer, Stuart Aitken, Sheila Q. Xie, Kelly J. Morris, Masayoshi Itoh, Hideya Kawaji, Ines Jaeger, Yoshihide Hayashizaki, Piero Carninci, Alistair R. R. Forrest, FANTOM Consortium, Colin A. Semple, Josée Dostie, Ana Pombo, and Mario Nicodemi. Hierarchical folding and reorganization of chromosomes are linked to transcriptional changes in cellular differentiation. *Mol. Syst. Biol.*, 11(12):852, 23 December 2015.

10 David U. Gorkin, Danny Leung, and Bing Ren. The 3D genome in transcriptional regulation and pluripotency. *Cell Stem Cell*, 14(6):762–775, 5 June 2014.

11 Daniel Jost, Cédric Vaillant, and Peter Meister. Coupling 1D modifications and 3D nuclear organization: data, models and function. *Curr. Opin. Cell Biol.*, 44:20–27, 2017.

12 Galih Kunarso, Na-Yu Chia, Justin Jeyakani, Catalina Hwang, Xinyi Lu, Yun-Shen Chan, Huck-Hui Ng, and Guillaume Bourque. Transposable elements have rewired the core regulatory network of human embryonic stem cells. *Nat. Genet.*, 42(7):631–634, 6 June 2010.

13 Erez Lieberman-Aiden, Nynke L van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R. Lajoie, Peter J. Sabo, Michael O. Dorschner, Richard Sandstrom, Bradley Bernstein, M. A. Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A Mirny, Eric S. Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 9 October 2009.

14 Darío G Lupiáñez, Malte Spielmann, and Stefan Mundlos. Breaking TADs: How alterations of chromatin domains result in disease. *Trends Genet.*, 32(4):225–237, 1 April 2016.

15 Yiqin Ma, Kiriaki Kanakousaki, and Laura Buttitta. How the cell cycle impacts chromatin architecture and influences cell fate. *Front. Genet.*, 6:19, 3 February 2015.

16 T. Pederson. Chromatin structure and the cell cycle. *Proc. Nat'l Acad. Sci. USA*, 69(8):2224–2228, August 1972.

17 Suhas S. P. Rao, Miriam H. Huntley, Neva C. Durand, Elena K. Stamenova, Ivan D. Bochkov, James T. Robinson, Adrian L. Sanborn, Ido Machol, Arina D. Omer, Eric S. Lander, and Erez Lieberman Aiden. A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell*, 159(7):1665–1680, 18 December 2014.

18 Tom Sexton, Eitan Yaffe, Ephraim Kenigsberg, Frédéric Bantignies, Benjamin Leblanc, Michael Hoichman, Hugues Parrinello, Amos Tanay, and Giacomo Cavalli. Three-dimensional folding and functional organization principles of the drosophila genome. *Cell*, 148(3):458–472, 3 February 2012.

19 Yin Shen, Feng Yue, David F. McCleary, Zhen Ye, Lee Edsall, Samantha Kuan, Ulrich Wagner, Jesse Dixon, Leonard Lee, Victor V. Lobanenkov, and Bing Ren. A map of the cis-regulatory sequences in the mouse genome. *Nature*, 488(7409):116–120, 2 August 2012.

20 Hanjun Shin, Yi Shi, Chao Dai, Harianto Tjong, Ke Gong, Frank Alber, and Xianghong Jasmine Zhou. TopDom: an efficient and deterministic method for identifying topological domains in genomes. *Nucleic Acids Res.*, 44(7):e70, 20 April 2016.

21 P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001.

Outlier Detection in BLAST Hits*

Nidhi Shah[1], Stephen F. Altschul[2], and Mihai Pop[3]

1 Department of Computer Science and Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA
nidhi@cs.umd.edu

2 Computational Biology Branch, NCBI, NLM, NIH, Bethesda, MD, USA
altschul@ncbi.nlm.nih.gov

3 Department of Computer Science and Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA
mpop@umiacs.umd.edu

Abstract

An important task in a metagenomic analysis is the assignment of taxonomic labels to sequences in a sample. Most widely used methods for taxonomy assignment compare a sequence in the sample to a database of known sequences. Many approaches use the best BLAST hit(s) to assign the taxonomic label. However, it is known that the best BLAST hit may not always correspond to the best taxonomic match. An alternative approach involves phylogenetic methods which take into account alignments and a model of evolution in order to more accurately define the taxonomic origin of sequences. The similarity-search based methods typically run faster than phylogenetic methods and work well when the organisms in the sample are well represented in the database. On the other hand, phylogenetic methods have the capability to identify new organisms in a sample but are computationally quite expensive. We propose a two-step approach for metagenomic taxon identification; i.e., use a rapid method that accurately classifies sequences using a reference database (this is a filtering step) and then use a more complex phylogenetic method for the sequences that were unclassified in the previous step. In this work, we explore whether and when using top BLAST hit(s) yields a correct taxonomic label. We develop a method to detect outliers among BLAST hits in order to separate the phylogenetically most closely related matches from matches to sequences from more distantly related organisms. We used modified BILD (Bayesian Integral Log Odds) scores, a multiple-alignment scoring function, to define the outliers within a subset of top BLAST hits and assign taxonomic labels. We compared the accuracy of our method to the RDP classifier and show that our method yields fewer misclassifications while properly classifying organisms that are not present in the database. Finally, we evaluated the use of our method as a pre-processing step before more expensive phylogenetic analyses (in our case TIPP) in the context of real 16S rRNA datasets. Our experiments demonstrate the potential of our method to be a filtering step before using phylogenetic methods.

1998 ACM Subject Classification B.2.4 Algorithms

Keywords and phrases Taxonomy classification, Metagenomics, Sequence alignment, Outlier detection

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.23

* S.F.A. was supported by the Intramural Research Program of the National Library of Medicine at the National Institutes of Health. N.S. and M.P. were partly supported by the NSF, award IIS-1513615 to M.P.

1 Introduction

One of the goals of metagenomic analyses is to characterize the biological diversity of microbial communities. This is usually achieved by targeted amplicon sequencing of the 16S rRNA gene, either as a whole gene or focused on a hypervariable region within the gene [21]. The 16S rRNA gene is commonly used for this purpose because it is universally found in bacteria and contains a combination of highly conserved and highly variable regions. Advances in sequencing technology, targeted to a specific gene, have generated millions to hundreds of millions of reads per study [6]. Assigning accurate taxonomic labels to these reads is one of the critical steps for downstream analyses.

The most common approach for assigning taxonomic labels to reads involves comparing them to a database of sequences from known organisms. These similarity-based methods typically run rapidly and work well when organisms in the sample are well represented in the database. However, a majority of microorganisms cannot be easily cultured in laboratories, and even if they are culturable, a smaller number have been sequenced. Thus, not all environmental organisms may be represented in the sequence database. This prevents the similarity-based methods from accurately characterizing organisms within a sample that are only distantly related to the sequences in the reference database. Phylogenetic-tree based methods can characterize novel organisms within a sample by statistically modeling the evolutionary processes that generated these sequences [15, 13]. However, such methods incur a high computational cost, limiting their applicability in the context of the large datasets generated in current studies. Ideally, we would want to use similarity-based methods to assign labels to sequences from known organisms, and to use phylogenetic methods to assign labels to sequences from unknown organisms.

We propose a two-step method for taxonomy assignment where we use a rapid assignment method that can accurately assign labels to sequences that are well represented in the database, and then use more complex phylogenetic methods to classify only those sequences unclassified in the first step. In this work, we study whether and when a method can assign accurate taxonomic labels using a similarity search of a reference database. We employ BLAST because it is one of the most widely used similarity search methods [1]. However, it has been shown that the best BLAST hit may not always provide the correct taxonomic label [11]. Most taxonomic-assignment methods utilizing BLAST employ ad-hoc techniques such as recording the consensus label among the top five hits, or using a threshold based on E-value, percent identity, or bit-score [20, 22, 16, 8]. Here we propose an alternative approach for detecting whether and when the top BLAST hits yield correct taxonomic labels. We model the problem of separating phylogenetically correct matches from matches to sequences from similar but phylogenetically more distant organisms as a problem of outlier detection among BLAST hits. Our preliminary results involving simulated and real metagenomic datasets demonstrate the potential of employing our method as a filtering step before using phylogenetic methods.

2 Background

2.1 Taxonomy assignment using BLAST

Several metagenomic analyses use BLAST to assign taxonomic labels to uncharacterized reads in a sample [20, 22, 16]. BLAST is a sequence similarity search tool, and it calculates an E-value and a bit-score to assess the quality of each match. An E-value represents the number of hits of equal or greater score expected to arise by chance. A bit-score can be understood as representing the size of the space one would need to search in order to find as strong a match by chance. However all 16S sequences are related, and therefore these scores, derived from a model of random sequences, do not provide simple information for separating sequences from different phylogenetic categories.

2.2 BILD scores for multiple sequence alignment

Multiple sequence alignments employ scoring functions to assess the quality of columns of aligned letters. Such functions have included Sum-of-the-Pairs (SP) scores [14], entropy scores [19], tree scores [17, 18] and the recently developed Bayesian Integral Log-Odds (BILD) score [2, 3]. For local pairwise alignment, substitution scores are implicitly of log-odds form [10]. BILD scores extend the log-odds formalism to multiple sequence alignments. They may be used in numerous contexts such as the construction of hidden Markov model profiles, the automated selection of optimal motifs, and the selection of insertion and deletion locations, and they can inform the decision of whether to include a sequence in a multiple sequence alignment. BILD scores can also be used to classify related sequences into subclasses, as we describe below.

3 Methods

Broadly, our approach constructs a multiple alignment from all the top hits obtained by comparing a query sequence to a database. We use BILD scores to determine whether the multiple alignment can be split into two groups that model the data better than does a single group. In essence, we find a subset of the sequences that are more closely related to one another and to the query than to the rest of the sequences in the multiple alignment. When there is no such subset i.e. when the single alignment models the data better, we leave the query unclassified and such a query sequence is then classified in the second step by a phylogenetic method.

3.1 Processing query sequences

Let S be the set of sequences in the reference database, each with a taxonomic label, and Q be a set of uncharacterized reads (i.e. query sequences). We first align each sequence in Q to sequences in S using BLAST (*-max_target_seqs 100 -outfmt 5 -task megablast*). For each $q \in Q$, we construct the ordered set S_q that contains the segments yielding the top 100 bit-scores, in decreasing order of their score. We discard all segments $l \in S_q$ where the BLAST alignment of q and l covers $\leq 90\%$ of q. We use the BLAST-generated local alignments involving q to impose a multiple alignment (M^q) on the sequences in $q \cup S_q$. Where several segments in S_q involves insertions at the identical location in q, we align these insertions to one another by left justification.

3.2 Scoring for Multiple Sequence Alignments and Cuts

We base our score for a multiple alignment (M^q) on the Bayesian Integral Log-Odds (BILD) scores described in [2]. For each alignment column, we take the prior for the nucleotide probabilities to be a Dirichlet distribution with parameters $\vec{\alpha}$, and define $\alpha^* = \sum_{k=1}^{4} \alpha_k$. (Here, we always use Jeffreys' prior [9], for which all $\alpha_k = 0.5$, and $\alpha^* = 2$.) For the j^{th} column M_j^q of the alignment and ignoring null characters, the log-probability of observing

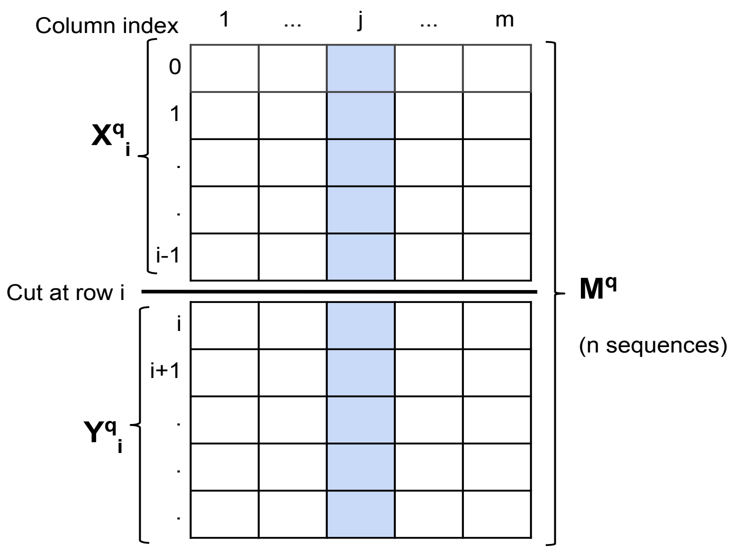


Figure 1 An example of how a cut divides an MSA into two disjoint groups.

the particular collection of c^*_j nucleotides, with count vector $\vec{c_j}$, is then given by

$$L(M^q_j) = \log \left[\frac{\Gamma(\alpha^*)}{\Gamma(\alpha^* + c^*_j)} \prod_{k=1}^{4} \frac{\Gamma(\alpha_k + c_{jk})}{\Gamma(\alpha_k)} \right].$$

Here, Γ is a *gamma* function. As suggested in [2], the log-odds score for preferring a cut, at row i, of the column M^q_j into the two sub-columns X^q_{ji} and Y^q_{ji}, as illustrated in Figure 1, is given by

$$V^q_{ji} = L(X^q_{ji}) + L(Y^q_{ji}) - L(M^q_j). \tag{1}$$

Taking all columns into account, the log-odds score for preferring a cut at row i is simply formula 1 summed over all columns. However, we have found it useful to give greater weight to columns with greater diversity. Thus we adopt the score V^q_i for a cut at row i given by the formula

$$V^q_i = \sum_{j=1}^{m} e^a_j V^q_{ji},$$

where M^q has m columns, $e_j = -\sum_{k=1}^{4} (c_{jk}/c^*_j) \log_4(c_{jk}/c^*_j)$ is the entropy (base 4) of column j, and a is an arbitrary positive parameter. Note that, using this formula, perfectly conserved columns have entropy 0 and thus weight 0, whereas columns with uniform nucleotide usage have entropy 1 and thus weight 1. We have found, by experimentation, that a useful value for the parameter a is 2.7.

3.3 Outlier detection and taxonomy assignment

We are interested in finding the phylogenetically most closely related matches in the database to the query sequence q. We proceed by computing V^q_i for cuts with increasing i, from $i = 0$, and identify first i' for which $V^q_{i'} \geq 0$, $V^q_{i'} > V^q_{(i'-1)}$, and $V^q_{i'} > V^q_{(i'+1)}$. In other words, we

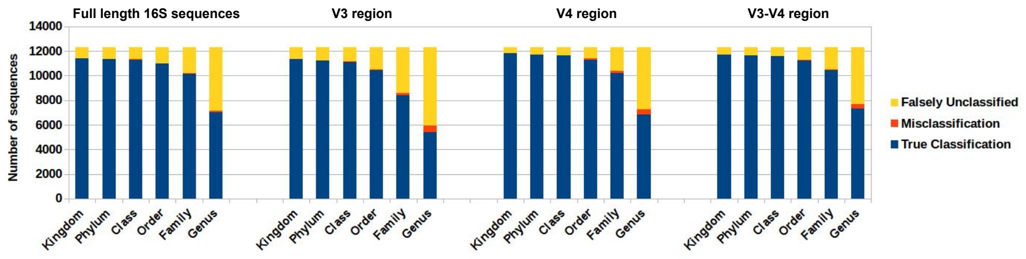


Figure 2 Leave-one-sequence-out validation of our outlier method using a simulated 16S rRNA dataset (RTS) for full-length, V3, V4, and V3-V4 regions.

find the first peak among those scores that imply the data are better explained by a split alignment. (Scores below zero favor a single alignment.) The first $i' - 1$ sequences from S_q we take as forming an outlier set $O_q = S_q[1 : i' - 1]$ for q. We extract the taxonomic labels of all sequences in O_q and assign the lowest common ancestor (LCA; [8]) of these labels to q. In the case when scores favor a single alignment, we leave the query sequence unclassified. The unclassified query sequences then should be classified, in step two of a two-step process, using a phylogenetic method.

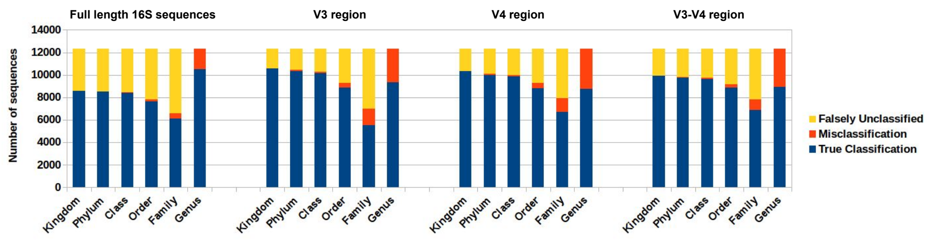
4 Evaluation

4.1 Datasets

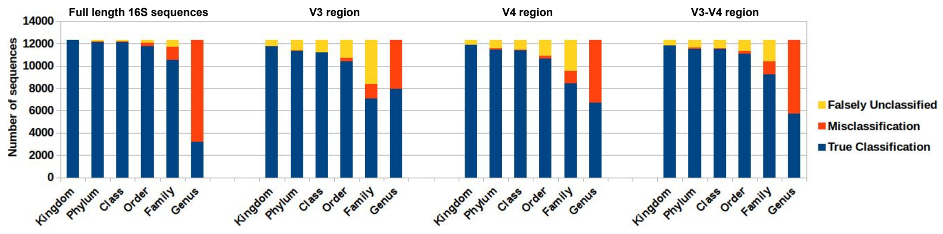
We used the RDP 16S rRNA gene v16 dataset (RTS), which has taxonomy annotated for each of its 13,212 sequences [5], considering only the 12,320 sequences that had taxonomic labels for all six levels - Kingdom, Phylum, Class, Order, Family, and Genus. These sequences belong to 2,320 genera with, on average, 6 sequences per genus. To evaluate our outlier detection method, we compared taxonomic labels assigned to query sequences by our method to their true labels as given in RTS. First, we used V-Xtractor with default parameters to extract the V3, V4 and V3-V4 hypervariable regions of the sequences [7]. We then used these V3 (SIM-2), V4 (SIM-3), V3-V4 (SIM-4) and full (SIM-1) sequences as query datasets and RTS sequences as a reference database. We also used a real metagenomic dataset (Dataset-1) to study the effectiveness of our method in actual practice. Dataset-1 has 58,108 sequences from the V1-V2 hypervariable region.

4.2 Leave-one-out validation

In the RTS simulated dataset, we know true taxonomic labels for all query sequences. For each taxonomic level, we compare the taxonomic labels assigned by our method to the true labels to find the number of queries that are correctly classified, misclassified or falsely unclassified. To identify correctly classified query sequences at each level, we compare, for all query sequences, the taxonomic labels assigned by our method to the true taxonomic label at that level. If the label assigned to a query by our method matches its true label, or if our method leaves the query sequence unassigned when there are no other sequences in the database with its particular label, we consider the query sequence as properly classified. For each taxonomic level, we consider misclassified those query sequences for which the assigned taxonomic label does not match the true label. We also consider falsely unclassified those sequences that were not assigned a taxonomic label at a particular level when the true label existed independently in the database.



(a)

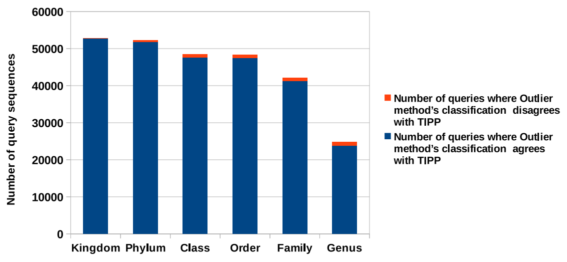


(b)

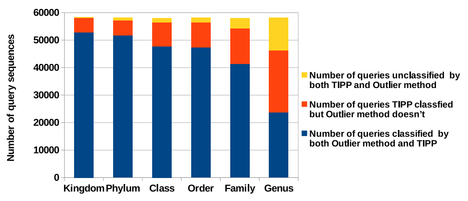
Figure 3 (a) Leave-one-genus-out validation of our outlier method using a simulated 16S rRNA dataset (RTS) for full-length, V3, V4, and V3-V4 regions. (b) Leave-one-genus-out validation of the RDP classifier on same 16S rRNA datasets.

Figure 2 shows the number of correctly classified, misclassified and falsely unclassified sequences calculated by leave-one-out cross-validation, where we assign a taxonomic label to a query sequence (full or hypervariable region) after removing its associated sequence from the database. For all query datasets, our method rarely misclassified at all taxonomic levels, generally assigned correct labels at higher levels, but tended not to assign labels at lower levels. This may be because our method uses the LCA of taxonomic labels of outlier sequences. When there are closely related sequences in the database, our method chooses to be conservative by not assigning labels at lower taxonomic levels.

To study the effectiveness of our method in classifying sequences with taxonomy unrepresented in the database, we performed genus-level leave-one-out cross-validation. Specifically, for each query, we removed all sequences from the database belonging to the same genus, and assigned taxonomic labels with our method and the RDP classifier [23]. We ran the RDP classifier using the QIIME [4] pipeline with the default confidence threshold of 80%. We calculated the number of queries that were correctly classified, misclassified and falsely unclassified as explained above. Figures 3a and 3b show results for our method and RDP respectively. Because the genus to which a query sequence belongs is never present in the database, any label assigned at genus level will result in a misclassification error, and no assignment will result in correct classification. We observed that for higher taxonomic levels (down to Order) RDP and our method have comparable misclassification rates. However, at the Family and Genus levels, our method has a lower misclassification rate. For all datasets, RDP misclassified more query sequences at the Genus level than did our method. This is primarily because RDP aggressively tries to classify as many sequences as it can, whereas our method prefers to classify only when it can do so accurately, leaving other sequences to be dealt with later by a phylogenetic method. This experiment shows that even when sequences from the same genus as the query are absent from the database, our method has high precision and makes few mistakes.



(a) Number of query sequences for which our method's classification agrees with TIPP's classification



(b) Number of query sequences classified by our method and TIPP vs. unclassfied by both

Figure 4 Evaluation of our outlier method using TIPP on a real metagenomic dataset.

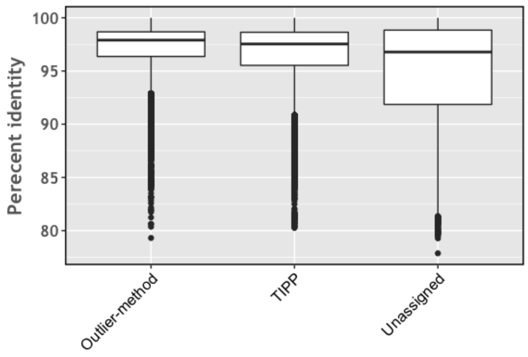


Figure 5 Box plot of percent identity of the best BLAST hit for all query sequences that were assigned label at genus level by our method and TIPP vs. queries that remained unassigned by both methods.

4.3 Validation using Phylogenetic-tree based assignment

To study the effectiveness of our outlier detection method in a realistic setting, we tested it on a real metagenomic dataset. Since we do not know the true taxonomic label for all query sequences, we compared our results with those produced by TIPP [15], a phylogenetic-tree based taxonomic assignment method. We used the RDP 2014 16S reference database for both methods. In this dataset, there were 58,108 query sequences for which our method assigned 41,256 sequences at the Family level or below. Figure 4a shows that our method has a high precision for all taxonomic levels. Also, Figure 4b suggests that using our outlier method to make taxonomic assignments (at least down to the Family level) can significantly reduce the workload of a phylogenetic-tree based method like TIPP. To classify 58,108 query sequences, our method required about 29 CPU hours (including BLAST time), whereas for the same dataset TIPP needed about 300 CPU hours. This shows the potential of our method as a filtering step before using phylogenetic-tree based methods. About 11,000 sequences remained unclassified by both TIPP and our method, and we investigated whether the best BLAST hit's percent identity correlates with the ability of these programs to make classifications; see Figure 5. Unfortunately, there is no clear percent-identity cutoff one can employ to recognize sequences that will remain unassigned by both methods, although a large number of the unassigned sequences have low similarity to the nearest database sequence.

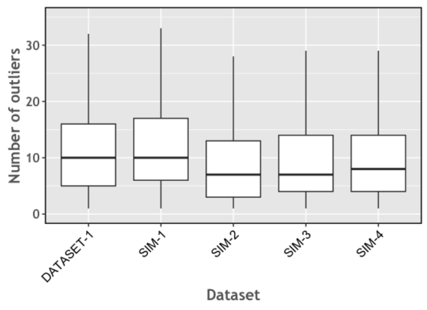


Figure 6 Box plot showing the variation in the number of outliers detected per query sequence in DATASET-1, SIM-1, SIM-2, SIM-3 and SIM-4.

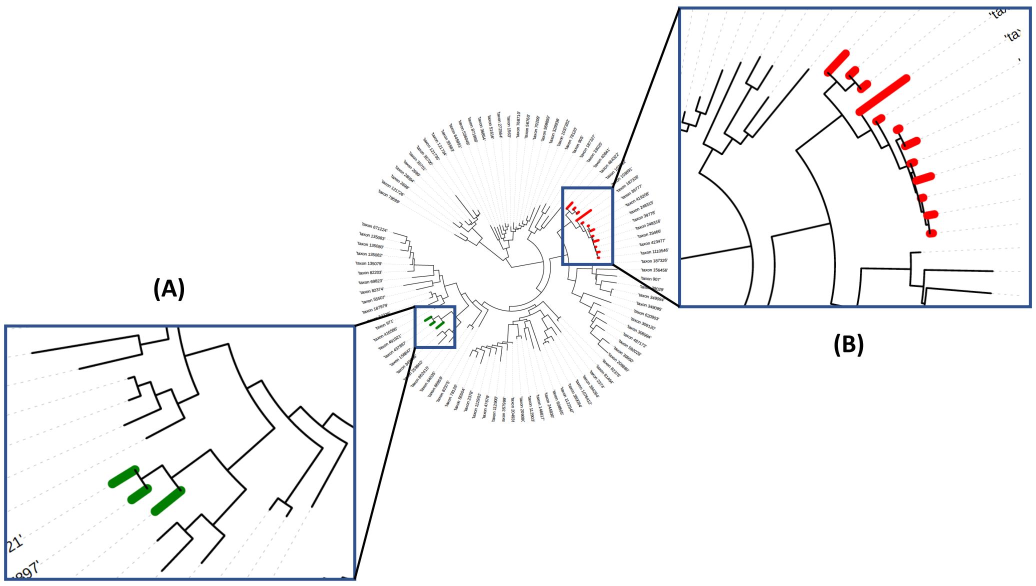


Figure 7 Phylogenetic tree showing outliers detected for two example query sequences.

4.4 Distribution of outliers

Since prior approaches restrict the analysis to just a fixed number of top hits, we evaluated the number of outliers proposed by our method. As seen in Figure 6, the number of outliers has large variance, so a single cutoff (say, the best or top five BLAST hits) will not identify all phylogenetically related matches from the database. In this case, we relied on data for which the true taxonomic label is not known. To validate whether the set of outliers detected by our method is reasonable, and to better understand the performance of our approach, we evaluated the placement of the outlier sequences within a phylogenetic tree of the database. For this, we used the phylogenetic tree for the RDP 2014 database that was bundled in the TIPP reference package, and used the Interactive Tree Of Life web tool to visualize outliers [12]. In general, we noticed that the outliers are grouped close to each other in the phylogenetic tree (see examples in Figure 7), suggesting that our method produces reasonable results. This analysis also revealed insights into the resolution level of the annotations provided by our method. When the outlier sequences cluster tightly within the phylogeny (Figure 7A), a reliable classification can be made at a low taxonomic level. When the outliers are distributed along a broader section of the tree (Figure 7B), the classification can only be made at a higher taxonomic levels.

5 Conclusion and Discussion

We propose a two-step approach for taxonomic assignment, in which we gain as much information as we reliably can from BLAST output before using computationally expensive phylogenetic-tree based methods on sequences that are difficult to classify. In this paper, we developed an outlier detection method for taxonomy assignment using BLAST hits that separates phylogenetically correct matches from matches to sequences from similar but phylogenetically more distant organisms. This method can thus be used for step one of a two-step approach, to identify sequences that can be assigned accurate labels using just a BLAST search of a reference database.

Because all 16S rRNA sequences are related, statistics like BLAST's E-value or bit-score do not provide ready information for separating sequences from different phylogenetic categories. Our experiments show also that there isn't any single cutoff that can be used to select BLAST hits for correctly assigning taxonomic labels. We have experimented with finding outliers using bit-score distributions, but found they provided insufficient information to detect phylogenetically correct matches (data not shown). Our experiments also show that although the percent identity of its best BLAST hit is correlated to a sequence's being assigned a taxonomic label, no particular percent-identity cutoff can separate those sequences that can be classified from those that cannot. This has motivated our development of a BILD-score based method to identify when the top BLAST hits will yield accurate taxonomic labels.

Because our method is used as a filtering step, we seek to accurately classify as many query sequences as possible while making few misclassifications. The sequences that we leave unclassified are then to be handled by a phylogenetic method. Our results on simulated and real 16S rRNA metagenomic datasets show that our method has high precision at all taxonomic levels, assigning correct labels at higher levels to a majority of sequences, and that it is computationally efficient compared to phylogenetic-tree based taxonomic assignment methods. This demonstrates the promise of a two-step taxonomic assignment approach, using our method as a filtering step.

In the future, we plan to study sequences that were classified correctly by phylogenetic methods but not by ours, to gain insight for possible improvements. We also plan to study the effectiveness of restricting phylogenetic-tree based methods to the subtree spanned by our method's outliers. Finally, note that our method was developed for and tested on 16S rRNA data, and is not applicable as it stands to whole genome sequencing (WGS) datasets. However, the idea of using a two-step approach for taxonomy assignment in WGS datasets is an interesting avenue for research.

Acknowledgement. We wish to thank anonymous reviewers for helpful comments.

References

1 Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

2 Stephen F. Altschul, John C. Wootton, Elena Zaslavsky, and Yi-Kuo Yu. The construction and use of log-odds substitution scores for multiple sequence alignment. *PLoS Comput Biol*, 6(7):e1000852, 2010.

3 Michael Brown, Richard Hughey, Anders Krogh, I. Saira Mian, Kimmen Sjölander, and David Haussler. Using Dirichlet mixture priors to derive hidden markov models for protein families. In *Ismb*, volume 1, pages 47–55, 1993.

4 J. Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D. Bushman, Elizabeth K. Costello, Noah Fierer, Antonio Gonzalez Peña, Julia K. Goodrich, Jeffrey I. Gordon, et al. QIIME allows analysis of high-throughput community sequencing data. *Nature methods*, 7(5):335–336, 2010.

5 James R. Cole, Qiong Wang, Jordan A. Fish, Benli Chai, Donna M. McGarrell, Yanni Sun, C. Titus Brown, Andrea Porras-Alfaro, Cheryl R. Kuske, and James M. Tiedje. Ribosomal Database Project: data and tools for high throughput rRNA analysis. *Nucleic acids research*, page gkt1244, 2013.

6 Jack A. Gilbert, Janet K. Jansson, and Rob Knight. The Earth Microbiome project: successes and aspirations. *BMC biology*, 12(1):69, 2014.

7 Martin Hartmann, Charles G. Howes, Kessy Abarenkov, William W. Mohn, and R. Henrik Nilsson. V-Xtractor: an open-source, high-throughput software tool to identify and extract hypervariable regions of small subunit (16s/18s) ribosomal RNA gene sequences. *Journal of Microbiological Methods*, 83(2):250–253, 2010.

8 Daniel H. Huson, Alexander F. Auch, Ji Qi, and Stephan C. Schuster. MEGAN analysis of metagenomic data. *Genome research*, 17(3):377–386, 2007.

9 Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London a: mathematical, physical and engineering sciences*, 186(1007):453–461, 1946.

10 Samuel Karlin and Stephen F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, 1990.

11 Liisa B. Koski and G. Brian Golding. The closest BLAST hit is often not the nearest neighbor. *Journal of molecular evolution*, 52(6):540–542, 2001.

12 Ivica Letunic and Peer Bork. Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display and annotation. *Bioinformatics*, 23(1):127–128, 2007.

13 Frederick A. Matsen, Robin B. Kodner, and E. Virginia Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC bioinformatics*, 11(1):538, 2010.

14 Michio Murata, Jane S. Richardson, and Joel L. Sussman. Simultaneous comparison of three protein sequences. *Proceedings of the National Academy of Sciences*, 82(10):3073–3077, 1985.

15 Nam-phuong Nguyen, Siavash Mirarab, Bo Liu, Mihai Pop, and Tandy Warnow. TIPP: taxonomic identification and phylogenetic profiling. *Bioinformatics*, 30(24):3548–3555, 2014.

16 Mihai Pop, Alan W. Walker, Joseph Paulson, Brianna Lindsay, Martin Antonio, M. Anowar Hossain, Joseph Oundo, Boubou Tamboura, Volker Mai, Irina Astrovskaya, et al. Diarrhea in young children from low-income countries leads to large-scale alterations in intestinal microbiota composition. *Genome biology*, 15(6):R76, 2014.

17 David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975.

18 David Sankoff and Robert J. Cedergren. Simultaneous comparison of three or more sequences related by a tree. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison/edited by David Sankoff and Joseph B. Krustal*, 1983.

19 Thomas D. Schneider, Gary D. Stormo, Larry Gold, and Andrzej Ehrenfeucht. Information content of binding sites on nucleotide sequences. *Journal of molecular biology*, 188(3):415–431, 1986.

20 Raúl Y. Tito, Simone Macmil, Graham Wiley, Fares Najar, Lauren Cleeland, Chunmei Qu, Ping Wang, Frederic Romagne, Sylvain Leonard, Agustín Jiménez Ruiz, et al. Phylotyping and functional analysis of two ancient human microbiomes. *PLoS One*, 3(11):e3703, 2008.

21 Susannah G. Tringe and Philip Hugenholtz. A renaissance for the pioneering 16S rRNA gene. *Current opinion in microbiology*, 11(5):442–446, 2008.

22 Susannah Green Tringe, Christian Von Mering, Arthur Kobayashi, Asaf A. Salamov, Kevin Chen, Hwai W. Chang, Mircea Podar, Jay M. Short, Eric J. Mathur, John C. Detter, et al. Comparative metagenomics of microbial communities. *Science*, 308(5721):554–557, 2005.

23 Qiong Wang, George M. Garrity, James M. Tiedje, and James R. Cole. Naive bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Applied and environmental microbiology*, 73(16):5261–5267, 2007.

Finding Local Genome Rearrangements*

Pijus Simonaitis[1] **and Krister M. Swenson**[2]

1 **ENS Lyon, Lyon, France**
`pijus.simonaitis@ens-lyon.fr`

2 **LIRMM, CNRS – Université Montpellier, Montpellier, France; and Institut de Biologie Computationnelle (IBC), Montpellier, France**
`swenson@lirmm.fr`

Abstract

The Double Cut and Join (DCJ) model of genome rearrangement is well studied due to its mathematical simplicity and power to account for the many events that transform genome architecture. These studies have mostly been devoted to the understanding of minimum length scenarios transforming one genome into another. In this paper we search instead for DCJ rearrangement scenarios that minimize the number of rearrangements whose breakpoints are unlikely due to some biological criteria. We establish a link between this Minimum Local Scenario (MLS) problem and the problem of finding a Maximum Edge-disjoint Cycle Packing (MECP) on an undirected graph. This link leads us to a 3/2-approximation for MLS, as well as an exact integer linear program. From a practical perspective, we briefly report on the applicability of our methods and the potential for computation of distances using a more general DCJ cost function.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms, G.2.2 Graph Algorithms, J.3 Life and Medical Sciences

Keywords and phrases genome rearrangement, double cut and join, maximum edge-disjoint cycle packing, Hi-C, NP-complete, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.24

1 Overview

The problem of sorting genomes by a prescribed set of biologically plausible rearrangements has been a central problem in comparative genomics for roughly a quarter century. The Double Cut and Join (DCJ) model covers a diverse set of these possible rearrangements while being grounded in a very simple mechanism [15, 2]. An important step forward is the development of methodology to find plausible rearrangement scenarios using biological constraints.

We recently introduced a model for weighting DCJs that is suitable for representing certain biological constraints. The model groups breakpoint regions between adjacent genes (or syntenic blocks) into equivalence classes that are likely to participate in a rearrangement [14]. The 3D spacial proximity of breakpoint regions could be used as such, and the data is becoming increasingly available due to an experiment called Hi-C [9, 13]. (The pertinence of this model is discussed in Section 7.) The model colors adjacencies for use with a binary cost function, where a DCJ acting on adjacencies with the same color is of zero cost while those acting on different colors are of cost one. We showed that the problem of finding – out

* This work is partially supported by the IBC (Institut de Biologie Computationnelle) (ANR-11-BINF-0002), by the Labex NUMEV flaship project GEM, and by the CNRS project Osez l'Interdisciplinarité.

of all minimum length rearrangement scenarios – a scenario that minimizes the number of costly moves, takes polynomial time [14].

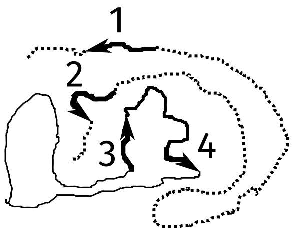
In this paper we disregard the length of the scenario and instead focus solely on the number of costly moves. We show that the Minimum Local Scenario problem is NP-Hard, while admitting a 3/2-approximation. This is done by exploiting a relationship to the Maximum Edge-disjoint Cycle Packing problem, a problem which was first linked to genome rearrangement in a different way by Alberto Caprara (sorting by unsigned reversals is NP-Hard [4]).

In our method, MLS is transformed into an edge elimination problem on a *junction graph*, representing the transitions between colors encountered when traversing the connected components of the adjacency graph. We give an exact formula for the number of costlyy moves based solely on the number of edges and the number of cycles in the MECP of the junction graph. We also propose an exact algorithm for MLS that is exponential in the number of colors and not in the number of genes and discuss an example where this algorithm is computationally feasible. Finally, we show how to bound the number of non-parsimonious moves when using a more general cost function.

The paper is organized as follows. Section 2 introduces basic definitions. In Section 3 our main result, relating MLS to MECP by way of the junction graph, is described in the simplified realm of sets of pairs. In this context, the junction graph is Eulerian. Section 4 extends the results to the general case where the junction graph is not Eulerian. Section 5 presents our algorithmic results. Finally, Section 6 discusses a more general cost function, while Section 7 reports on practical aspects of coloring adjacencies.

2 Genome and DCJ rearrangements

A *genome* consists of *chromosomes* that are linear or circular molecules partitioned into uniquely labeled directed genes (or equivalently *syntenic blocks* of genes) and intergenic regions separating them.



The genome depicted above consists of a linear and a circular chromosome each having two genes. Arrows in the picture indicate the *head extremities* of the genes. We can represent a genome by a set of *adjacencies* between the gene extremities and such a set for a genome from our example is $\{\{1_h\}, \{1_t, 2_t\}, \{2_h\}, \{3_h, 4_t\}, \{4_h, 3_t\}\}$. Here 1_h denotes the head extremity of a gene 1. An *adjacency* is either an unordered pair of gene extremities that are adjacent on a chromosome, called *internal* adjacency, or a single gene extremity adjacent to one of the two ends of a linear chromosome, called an *external* adjacency.

▶ **Definition 1** (Double Cut and Join). DCJ move acts on one or two adjacencies as follows:

1. $\{a, b\}, \{c, d\} \to \{a, c\}, \{b, d\}$ or $\{a, d\}, \{b, c\}$
2. $\{a, b\}, \{c\} \to \{a, c\}, \{b\}$ or $\{b, c\}, \{a\}$
3. $\{a, b\} \to \{a\}, \{b\}$
4. $\{a\}, \{b\} \to \{a, b\}$

We first treat a simplified instance of sets of pairs where all the adjacencies are internal and DCJs can only swap the elements between them in Section 3. Then in Section 4 we show how genomes can be extended to the sets of pairs and use the previously obtained results to solve the Minimum Local Scenario problem.

3 Minimum Local Scenario for sets of pairs

3.1 Cost of a DCJ scenario

Given two sets of *pairs*:

$$A = \{\{1,2\},\ldots,\{2n-1,2n\}\}, \text{ and}$$
$$B = \{\{q_1,q_2\},\ldots,\{q_{2n-1},q_{2n}\}\},$$

with *pairs* being unordered and $(q_1,\ldots,q_{2n})$ being a permutation of $(1,\ldots,2n)$, our goal is to transform A into B with a sequence of DCJ moves $\{a,b\},\{c,d\} \to \{a,c\},\{b,d\}$ and $\{a,b\},\{c,d\} \to \{a,d\},\{b,c\}$.

A *coloring* of a set of pairs A over a set of colors Δ is a function $col : A \to \Delta$ partitioning A into the subsets of different colors. A *coloring* is used to define the *cost* of a DCJ move. A move is *local* and of zero cost if it acts on the pairs with equal colors and it is *non-local* and of cost 1 otherwise. The *cost* of a sequence of DCJ moves, a DCJ *scenario*, is the sum of the costs of its constituent moves.

A DCJ move $A \to A'$ transforming a set of pairs A into A' will also transform col into col', a coloring of A'. This means that a DCJ scenario transforming A into B will transform A's coloring col into B's coloring col_B. For a pair $p \in A$ we use notation $(p, col(p))$ of a *colored pair*. Four different DCJ moves on colored pairs $(\{a,b\},x)$ and $(\{c,d\},y)$ are allowed in our model giving four possible outcomes:

$$(\{a,c\},x),(\{b,d\},y), \text{ or } (\{a,d\},x),(\{b,c\},y), \text{ or}$$
$$(\{a,c\},y),(\{b,d\},x), \text{ or } (\{a,d\},y),(\{b,c\},x).$$

The biological interpretation of this model is that intergenic regions are broken and repaired at their borders with the gene extremities. We discuss the applicability of such a model in Section 7.

In our previous work [14] we have treated the Minimum Local Parsimonious Scenario problem.

▶ **Problem 1** (MLPS). *For two sets of pairs A, B and a coloring of A find a minimum cost scenario among the DCJ scenarios of minimum length transforming A into B.*

We have shown that MLPS takes polynomial time, however the real evolutionary scenario might be non-parsimonious. In this paper we study the Minimum Local Scenario problem which asks for such non-parsimonious scenarios.

▶ **Problem 2** (MLS). *For two sets of pairs A, B and a coloring of A find a minimum cost DCJ scenario transforming A into B.*

MLS has the following combinatorial interpretation. Pairs of uniquely labeled balls (set of pairs A) are partitioned into the bins (coloring of A). Given a partition T of the balls into pairs (set of pairs B), find a minimum length sequence of ball swaps between the bins (DCJ moves) so that for all pairs $\{a,b\} \in T$, a and b end up in the same bin (A is transformed into B).

3.2 Adjacency and junction graphs

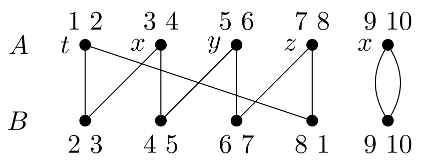
The *Adjacency* graph was introduced in [2] for the study of DCJ rearrangements. We introduce a transformation of the adjacency graph, called a *junction* graph, that incorporates the information on a coloring.

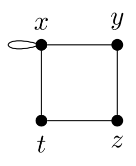
▶ **Definition 2** (Adjacency graph)**.** For two sets of pairs A and B the adjacency graph $AG(A, B)$ is defined as a bipartite multi-graph whose vertices are $A \cup B$ and for each $p \in A$ and $q \in B$ there are exactly $|p \cap q|$ edges joining these two vertices.

▶ **Definition 3** (Junction graph)**.** For two sets of pairs A, B and a coloring col of A over Δ we define a multi-graph $J(A, B, col) = (\Delta, E)$. For every pair $\{a, b\} \in B$ we add an edge (x, y) to E such that x and y are the colors of the pairs of A adjacent to $\{a, b\}$ in $AG(A, B)$.

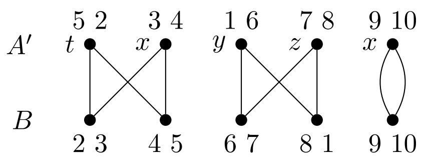
▶ **Example 4.** For two sets of pairs A, B and a coloring col of A we present below $AG(A, B)$ on the left and $J(A, B, col)$ on the right.

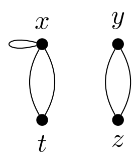
$$A = \big\{(\{1,2\},t), (\{3,4\},x), (\{5,6\},y), (\{7,8\},z), (\{9,10\},x)\big\}$$
$$B = \big\{\{2,3\}, \{4,5\}, \{6,7\}, \{8,1\}, \{9,10\}\big\}$$





A DCJ move $(\{1,2\},t), (\{5,6\},y) \rightarrow (\{5,2\},t), (\{1,6\},y)$ transforming A into A' transforms adjacency and junction graphs as follows.





All connected components of $AG(B, B)$ are cycles of length 2 thus at the end of a DCJ scenario transforming A into B we are left with a junction graph whose edges are all loops, we call such a graph *terminal*.

▶ **Definition 5** (A DCJ move on a graph)**.** Edges (x, y) and (z, t) of a graph are deleted and replaced by either (x, z) and (y, t) or (x, t) and (y, z).

▶ **Lemma 6.** *For a DCJ scenario of cost w transforming A into B there exists a DCJ scenario of length at most w transforming $J(A, B, col)$ into a* terminal *graph and vice versa.*

Proof. From Example 4 it should be clear that for every DCJ move $A \rightarrow A'$ we have that a transformation $J(A, B, col)$ to $J(A', B, col')$ is a DCJ move on a graph. If a DCJ move $A \rightarrow A'$ is of zero cost, then $J(A, B, col) = J(A', B, col')$ and such moves will be omitted from a DCJ scenario on a graph. This means that a DCJ scenario of cost w transforming A (and its coloring col) into (B and its coloring col_B) provides us with a DCJ scenario of length at most w on a graph transforming $J(A, B, col)$ into a terminal graph $J(B, B, col_B)$. On the other hand for every DCJ move on a graph $J \rightarrow J'$ a DCJ move $A \rightarrow A'$ can be found such that $J(A', B, col') = J'$. For any DCJ scenario on a graph of length w transforming

$J(A, B, col)$ into a terminal graph we obtain a DCJ scenario of length w, thus of cost at most w, transforming A and its coloring col into C and its coloring col_C such that $J(C, B, col_C)$ is terminal. This means that C's pairs belonging to the same connected component of $AG(C, B)$ are of the same color. A DCJ scenario transforming C into B and only acting on the pairs belonging to the same connected components of an adjacency graph can be easily found. Such a scenario is of zero cost and at the end we obtain a DCJ scenario transforming A into B of cost at most w. ◀

3.3 Linking DCJ scenarios and Maximum Edge-disjoint Cycle Packings

Using Lemma 6 we can shift our attention from a DCJ scenario on a set of pairs to a DCJ scenario on a junction graph J. From now on we will shorten "DCJ move on a graph" to "DCJ move".

▶ **Definition 7** (Maximum Edge-disjoint Cycle Packing (MECP)). Maximum Edge-disjoint Cycle Packing of a graph G is a largest set of edge-disjoint cycles in G.

For a graph $G = (V, E)$ we note $E(G) = |E|$ and $c(G)$ the size of its MECP. For a junction graph J we write $w(J)$ to indicate the minimum length of a DCJ scenario transforming J into a *terminal* graph.

▶ **Theorem 8.** *For a junction graph J we have $w(J) = E(J) - c(J)$.*

Proof. It is easy to transform a cycle of length $n > 1$ into n loops in $n-1$ DCJ moves. Given a cycle packing C of J we construct a DCJ scenario transforming J into a terminal graph while transforming all of its cycles separately. The length of such a scenario is $E(J)-|C|$, and if we take a Maximum Edge-disjoint Cycle Packing we obtain $w(J) \leq E(J) - c(J)$.

Now take a scenario of m DCJ moves transforming $J = J_0$ into a terminal graph J_m, with J_k being the junction graph after $k \geq 0$ moves of the scenario. We enumerate J's edges $E = \{e_1, \ldots, e_{E(J)}\}$ and define their partition into singletons $P_0 = \{\{1\}, \ldots, \{E(J)\}\}$. A move k of a scenario acts on two edges e_i and e_j of J_{k-1}, deleting them and introducing two new edges to give J_k. We call one of these edges e_i and another e_j, preserving the enumeration of the edges of J_k. Let S^i and S^j be the subsets of a partition P_{k-1} including e_i and e_j respectively. We define a partition P_k of $\{e_1, \ldots, e_{E(J)}\}$ obtained from P_{k-1} by merging S^i and S^j into $S^i \cup S^j$. At the end we obtain a partition P_m of cardinality at least $E(J) - m$ as there were at most m merges on the way. We will show that P_m is a cycle packing of J.

For J's vertices V, a subset $S \subset \{e_1, \ldots, e_{E(J)}\}$ and $k \in \{0, \ldots, m\}$, we define $S_{J_k} = (V, S)$ a subgraph of J_k. For any graph G and its vertex v we denote $d_G(v)$ the degree of v in G.

▶ **Lemma 9.** *For $k \in \{0, \ldots m\}$, a subset S in a partition P_k, and a vertex v of J we have*

$$d_{S_J}(v) = d_{S_{J_k}}(v).$$

Proof. $J_0 = J$, thus the equality is true for $k = 0$. We suppose that equality is true for every S and v with $k-1$ and proceed by induction on k. We fix a vertex v and a subset $S \in P_k$. The k-th move of a scenario acts on the edges e_i and e_j that by construction belongs to the same subset $S' \in P_k$. There are three possibilities:

1. ($S' \neq S$) In this case $S \in P_{k-1}$ and $S_{J_k} = S_{J_{k-1}}$, as the edges in S are unaffected by a DCJ move. Using the inductive hypothesis we obtain

$$d_{S_{J_k}}(v) = d_{S_{J_{k-1}}}(v) = d_{S_J}(v)\,.$$

2. ($S' = S$ and $S = S^i \cup S^j$ with S^i and S^j being the different subsets in P_{k-1} including e_i and e_j respectively) S_{J_k} is obtained from $S_{J_{k-1}}$ via a DCJ move and, as a DCJ move does not affect the degrees of the vertices we obtain, we use the inductive hypothesis and the fact that $S = S^i \cup S^j$ to get

$$d_{S_{J_k}}(v) = d_{S_{J_{k-1}}}(v) = d_{S^i_{J_{k-1}}}(v) + d_{S^j_{J_{k-1}}}(v) = d_{S^i_J}(v) + d_{S^j_J}(v) = d_{S_J}(v)\,.$$

3. ($S' = S$ and e_i, e_j already present in the same subset S of P_{k-1}) S_{J_k} is obtained from $S_{J_{k-1}}$ via a DCJ move which does not affect the degrees of the vertices and thus we obtain (using inductive hypothesis)

$$d_{S_{J_k}}(v) = d_{S_{J_{k-1}}}(v) = d_{S_J}(v)\,.$$

As equality is preserved by a DCJ move and true for $k = 0$ we obtain the result by induction. ◀

All edges of J_m are loops, thus for every subset S in P_m and vertex v of J, $d_{S_{J_m}}(v)$ is even and so, using Lemma 9, we know that $d_{S_J}(v)$ is even as well. This means that the connected components of S_J are Eulerian and thus S is a union of J's cycles. As P_m contains at least $E(J) - m$ subsets, we know that it partitions the edges of J into at least $E(J) - m$ cycles. If we take a scenario of length $w(J)$ we obtain the inequality $c(J) \geq E(J) - w(J)$, which ends the proof of Theorem 8. ◀

4 Minimum Local Scenario for genomes

4.1 Cost of a DCJ scenario

Given two genomes with enumerated extremities

$$A = \big\{\{1,2\},\dots,\{2n-1,2n\},\{2n+1\},\dots,\{2n+2m\}\big\},$$
$$B = \big\{\{q_1,q_2\},\dots,\{q_{2l-1},q_{2l}\},\{q_{2l+1}\},\dots,\{q_{2n+2m}\}\big\},$$

and $(q_1,\dots,q_{2n+2m})$ being a permutation of $(1,\dots,2n+2m)$. Our goal is to transform A into B using DCJ moves defined in Definition 1. As in the case of pairs-only, we define a coloring $col : A \to \Delta$ which is transformed by DCJ moves as follows:

1. $(\{a,b\},x),(\{c,d\},y) \to (\{a,c\},x),(\{b,d\},y)$ or $(\{a,d\},x),(\{b,c\},y)$ or $(\{a,c\},y),(\{b,d\},x)$ or $(\{a,d\},y),(\{b,c\},x)$
2. $(\{a,b\},x),(\{c\},y) \to (\{a,c\},x),(\{b\},y)$ or $(\{a,c\},y),(\{b\},x)$ or $(\{b,c\},x),(\{a\},y)$ or $(\{b,c\},y),(\{a\},x)$
3. $(\{a,b\},x) \to (\{a\},x),(\{b\},z)$ or $(\{a\},z),(\{b\},x)$ with any color z
4. $(\{a\},x),(\{b\},y) \to (\{a,b\},x)$ or $(\{a,b\},y)$

The cost of a DCJ move is equal to 0 if $z = x$ or $x = y$, 1 otherwise.

4.2 Genome extensions

A genome can be extended into a set of pairs by adding artificial gene extremities that represent *telomeres* marking the ends of each linear chromosome.

▶ **Definition 10** (Genome extensions)**.** For a genome A we define a set A_+ of the sets of pairs that are *genome extensions* of A. $\hat{A} \in A_+$ is of a form:

$$\{\{1,2\},\dots,\{2n-1,2n\},\{2n+1,\circ_1\},\dots,\{2n+2m,\circ_{2m}\},$$
$$\{\circ_{2m+1},\circ_{2m+2}\},\dots,\{\circ_{2m+2l-1},\circ_{2m+2l}\}\}$$

with $l \in \mathbb{N}$ and $(\circ_1,\dots,\circ_{2m+2l})$ being a permutation of $(2n+2m+1,\dots,2n+4m+2l)$.

A pair $\{i,j\}$ with $i,j > 2n+2m$ will be called a *telomeric* pair. By construction, adjacencies of a genome and non-telomeric pairs of a genome extension can be mapped one to one as internal adjacencies of a genome are present in the genome extension, and external adjacencies are simply complemented by an artificial gene extremity. A coloring *col* of A can be trivially extended to a coloring $\hat{col}$ of $\hat{A} \in A_+$ by keeping the same colors for the non-telomeric pairs and choosing any colors for the telomeric ones. For every DCJ move $A \to A'$ acting on two adjacencies of a genome there is an *induced* DCJ move $\hat{A} \to \hat{A}'$ of the same cost with $\hat{A}' \in A'_+$ acting on the corresponding pairs of a genome extension. For example $(\{a\},x),(\{b\},y) \to (\{a,b\},x)$ induces $(\{a,\circ_1\},x),(\{b,\circ_2\},y) \to (\{a,b\},x),(\{\circ_1,\circ_2\},y)$ and $(\{a,b\},x)(\{c\},y) \to (\{a,c\},x),(\{b\},y)$ induces $(\{a,b\},x),(\{c,\circ_1\},y) \to (\{a,c\},x),(\{b,\circ_1\},y)$. A DCJ move of the form $(\{a,b\},x) \to (\{a\},x),(\{b\},z)$ or $(\{a\},z),(\{b\},x)$ acting on a single adjacency is different, as in this case we need a telomeric adjacency of color z to be present in a genome extension. For example $(\{a,b\},x) \to (\{a\},x),(\{b\},z)$ induces $(\{a,b\},x),(\{\circ_1,\circ_2\},z) \to (\{a,\circ_1\},x),(\{b,\circ_2\},z)$ on a genome extension including $(\{\circ_1,\circ_2\},z)$.

▶ **Lemma 11.** *For a DCJ scenario transforming genome A into B and a coloring of A there exist genome extensions $\hat{A} \in A_+$, $\hat{B} \in B_+$ and a scenario of the same cost transforming $\hat{A}$ into $\hat{B}$.*

Proof. In a DCJ scenario there is a certain number l of the DCJ moves acting on a single adjacency. We take a genome extension $\hat{A} \in A_+$ with l telomeric pairs. Every DCJ move $(\{a,b\},x) \to (\{a\},x),(\{b\},z)$ or $(\{a\},z),(\{b\},x)$ will induce a move acting on a different telomeric pair of a genome extension and its color will be a color z required by that DCJ move on a genome. In this way every DCJ move on a genome will induce a move on a genome extension and after a scenario of cost w we will end up with $\hat{B}$, an extension of genome B. ◀

▶ **Lemma 12.** *For a DCJ scenario transforming $\hat{A} \in A_+$ into $\hat{B} \in B_+$ and a coloring of A there exists a DCJ scenario of the same cost or smaller transforming A into B.*

Proof. We start with a couple $(A,\hat{A})$ and apply a scenario transforming $\hat{A}$ into $\hat{B}$ step by step, transforming A on the way. After the first k moves of a scenario whose cost is w_k we get a couple $(A^k,\hat{A}^k)$ with $\hat{A}^k \in A^k_+$ and A^k obtainable from A by a scenario of cost at most w_k. A couple $(A^{k+1},\hat{A}^{k+1})$ is constructed as follows. The $k+1$st move of a scenario is $\hat{A}^k \to \hat{A}^{k+1}$.

1. If $\hat{A}^{k+1} \in A^k_+$, then output $(A^k,\hat{A}^{k+1})$.
2. If $\hat{A}^{k+1} \notin A^k_+$, then we can easily find a genome C such that $\hat{A}^{k+1} \in C_+$ and there is a DCJ move $A^k \to C$ of the same cost as $\hat{A}^k \to \hat{A}^{k+1}$. Output $(C,\hat{A}^{k+1})$.

Now $\hat{A}^{k+1} \in A^{k+1}_+$ and the scenario transforming A into A^{k+1} is of cost at most w_{k+1}. We continue until we obtain $(B,\hat{B})$ with a scenario transforming A into B of cost at most w. ◀

▶ **Definition 13** (Adjacency graph)**.** For two genomes A and B the adjacency graph $AG(A,B)$ is defined as a bipartite multi-graph whose vertices are $A \cup B$ and for each $p \in A$ and $q \in B$ there are exactly $|p \cap q|$ edges joining these two vertices.

▶ **Definition 14** (Junction graph). For two genomes A, B and a coloring col of A over Δ we define a multi-graph $J(A, B, col) = (\Delta, E)$. For every internal adjacency $\{a, b\} \in B$ we add an edge (x, y) to E such that x and y are the colors of the pairs of A adjacent to $\{a, b\}$ in $AG(A, B)$.

We define an *Eulerian extension* of a graph to be an Eulerian graph obtained from the initial one by adding some edges. By construction $J(\hat{A}, \hat{B}, \hat{col})$ is an Eulerian extension of $J(A, B, col)$. We close this section by relating Eulerian extensions of $J(A, B, col)$ to the junction graphs of genome extensions, the proof is provided in the appendix.

▶ **Lemma 15.** *For every Eulerian extension J' of $J(A, B, col)$ there exists genome extensions $\hat{A} \in A_+$ and $\hat{B} \in B_+$ such that $J(\hat{A}, \hat{B}, \hat{col})$ and J' have exactly the same non-loop edges. We say that such graphs are* loop-equal*.*

4.3 Minimum Local Scenario

▶ **Theorem 16.** *The minimum cost w of a DCJ scenario transforming genome A into B is $E(J) - c(J)$ with $J = J(A, B, col)$.*

Proof. For a cycle packing C of J of cardinality $c(J)$ we define an Eulerian extension J' with every edge of J not belonging to C duplicated, we denote the number of such edges by k. A union of C and k cycles of length 2 created by the added edges will be a cycle packing C' of J'. Using Theorem 8 we obtain a DCJ scenario of length $E(J') - |C'| = E(J) + k - c(J) - k = E(J) - c(J)$ transforming J' into a terminal graph. Using Lemma 15 we obtain the sets of pairs $\hat{A} \in A_+$ and $\hat{B} \in B_+$ such that $J(\hat{A}, \hat{B}, \hat{col})$ is loop-equal to J'. Using Lemma 6 we obtain a DCJ scenario of cost at most $E(J) - c(J)$ transforming $\hat{A}$ into $\hat{B}$ from which we obtain a DCJ scenario of cost at most $E(J) - c(J)$ transforming A into B while using Lemma 12, meaning that $w \leq E(J) - c(J)$.

For a DCJ scenario of cost w transforming A into B we use Lemma 11 to obtain the sets of pairs $\hat{A} \in A_+$ and $\hat{B} \in B_+$, and a scenario of cost w transforming $\hat{A}$ into $\hat{B}$. This leads to a DCJ scenario transforming $J' = J(\hat{A}, \hat{B}, \hat{col})$ into a terminal graph in at most w moves using Lemma 6. Theorem 8 gives us a cycle packing C' of J' such that $w \geq E(J') - |C'|$. We then define C to be the union of the cycles in C' consisting entirely of the edges of $J = J(A, B, col_A)$. While counting edges and cycles we obtain

$$w \geq E(J') - |C'| = E(J) - |C| + E(J') - E(J) - |C' \setminus C|.$$

Due to the construction of C every cycle in $C' \setminus C$ admits at least one edge from J' not belonging to J and thus $E(J') - E(J) \geq |C' \setminus C|$. So we have inequality $w \geq E(J) - |C| \geq E(J) - c(J)$, which ends the proof. ◀

5 Algorithms for MLS

5.1 NP-completeness of MLS

▶ **Theorem 17.** *The decision version of* Minimum Local Scenario *is NP-complete.*

Proof. The decision version of MLS is clearly in NP. We reduce the decision version of MECP on Eulerian graphs, which is NP-hard [7] (and APX-hard [5]), to MLS. Without loss of generality, take an instance $G = (V, E)$ and a bound k of MECP, where G is Eulerian and connected. Consider an Eulerian cycle $u_1, u_2, \ldots, u_n, u_1$ of G and construct genomes

$$A = \big\{\{1,2\}, \{3,4\}, \ldots, \{2n-1, 2n\}\big\}, \text{and}$$
$$B = \big\{\{2,3\}, \{4,5\}, \ldots, \{2n, 1\}\big\},$$

and a coloring *col* over the set V such that $col\big(\{2i-1, 2i\}\big) = u_i$ for all $i \in \{1, \ldots, n\}$ to obtain $J(A, B, col) = G$. Theorem 16 says that an optimal solution to MLS of cost $w(G)$ implies the existence of a cycle packing of size $E(G) - w(G)$. Thus there is an MECP of size k if and only if $E(G) - w(G) \geq k$. ◀

5.2 3/2-approximation for MLS

For a graph G we denote the number of edges by $E(G)$, the number length one and two cycles by $L(G)$ and $B(G)$ respectively. A simple counting argument leads to the following theorem proved in the appendix.

▶ **Theorem 18.** *For genomes A, B and a coloring col of A, the cost w_{MLS} of a* MLS *transforming A into B respects*

$$w_{\text{MLS}} \geq \frac{2}{3}E(J) - \frac{1}{3}B(J) - \frac{2}{3}L(J), \textit{where } J = J(A, B, col).$$

In Theorem 16 we have shown how a cycle packing C of $J = J(A, B, col)$ gives a DCJ scenario of cost $w \leq E(J) - |C|$ transforming A into B. If we take C consisting of $B(J)$ pairwise edge-disjoint cycles of length two and $L(J)$ loops, we obtain a scenario of cost $w \leq E(J) - B(J) - L(J) = w'$. Using Theorem 18 we have

$$w_{\text{MLS}} \geq \frac{2}{3}E(J) - \frac{1}{3}B(J) - \frac{2}{3}L(J) = \frac{2}{3}(w' + \frac{1}{2}B(J))$$

and obtain

$$\alpha = \frac{w}{w_{\text{MLS}}} \leq \frac{3}{2}\frac{w}{w' + \frac{1}{2}B(J)} \leq \frac{3}{2}.$$

5.3 An exact algorithm for MLS

Consider a junction graph J with $L(J)$ loops and $B(J)$ length two cycles. A simple observation that there exists a MECP of J that includes all of these cycles allows us to simplify the problem by removing them from J. This leaves us with a simple graph $\bar{J}$ such that the cost of MLS is equal to $E(J) - L(J) - B(J) - c(\bar{J})$. A straightforward way to compute $c(\bar{J})$ is to take all of $\bar{J}$'s simple cycles and solve the Maximum Set Packing problem on their sets of edges formulated as an integer linear program. The number of simple cycles might be exponential, but it depends on the size of a simple graph $\bar{J}$ having $|\Delta|$ vertices and not the number of genes. We see in Section 7 that our algorithm solves MLS on instances between *drosophila melanogaster* and *yakuba*.

6 Towards a more general cost function

Our work opens the door to the development of a more general model for genome rearrangements with positional constraints, where local moves are attributed nonzero cost. In such a model the costs of local and non-local moves would be respectively ω_L and ω_N with $0 < \omega_L < \omega_N$. For any DCJ scenario ρ we will denote $\omega(\rho)$, $N(\rho)$ and $L(\rho)$ as its cost, its number of non-local, and local moves respectively. We categorize the different DCJ problems based on the cost pair (ω_L, ω_N) with $0 \leq \omega_L \leq \omega_N$ where we look for a ρ that minimizes the cost function $\omega(\rho) = \omega_L L(\rho) + \omega_N N(\rho)$:

- $(0, 1)$ is the Minimum Local Scenario problem,
- $(1, 1)$ is the traditional Double Cut and Join problem,
- (ω_L, ω_N) with $\frac{\omega_L}{\omega_N - \omega_L} > n$, where n is the number of adjacencies, is the Minimum Local Parsimonious Scenario problem,
- (ω_L, ω_N) with $0 < \omega_L < \omega_N$ is the problem that we consider in this section.

It is clear that for positive k the cost pairs (ω_L, ω_N) and $(k\omega_L, k\omega_N)$ define the same minimum scenarios, thus for $0 < \omega_L < \omega_N$ it suffices to treat the normalized pair $(1, 1+\alpha)$ with a positive α. For a scenario ρ we denote $\delta(\rho) = N(\rho) + L(\rho) - d_{DCJ}$ the difference of its length and the length of a parsimonious DCJ scenario. If δ were small we would have an algorithmic tool in the search for the genomic distances. For $(\omega_L, \omega_N) = (1, 1+\alpha)$, we have

$$\omega(\rho) = L(\rho) + N(\rho) + N(\rho)\alpha = \delta(\rho) + d_{DCJ} + N(\rho)\alpha,$$

By d_{MLPS} and d_{MLS} we denote the numbers of non-local moves in Minimum Local Parsimonious Scenario and Minimum Local Scenario respectively. For a scenario ρ^* minimizing the cost $L(\rho) + (1+\alpha)N(\rho)$ we have $d_{DCJ} + d_{MLPS}\alpha \geq \omega(\rho*)$ as $d_{DCJ} + d_{MLPS}\alpha$ is the cost of a MLPS and subtracting d_{DCJ} we obtain $d_{MLPS}\alpha \geq \delta(\rho*) + N(\rho^*)\alpha$. By definition $N(\rho^*) \geq d_{MLS}$ and thus we obtain

$$(d_{MLPS} - d_{MLS})\alpha \geq \delta(\rho^*).$$

In general $d_{MLPS} - d_{MLS}$ can be large. For the experiments in Section 7, however, it was found to be smaller than 0.8 on average. This means that finding a scenario of minimum cost among those with a small δ, for example $\delta = 1$, might be of interest in practice.

7 The practice of coloring adjacencies

In this section we address the applicability of our model that colors adjacencies. We summarize our experimental results on *drosophila melanogaster* and *yakuba* reported in [12].

We use the Hi-C data for *drosophila* as a similarity function on the pairs of the adjacencies of a genome. We generate colorings using a centroid-based clustering [10]; the adjacencies are clustered based on Hi-C similarity, and two adjacencies get the same color if they are in the same cluster. Weights are assigned to the colorings based on how well they respect the within-clusters similarity. MLS is then computed on the colorings.

The first positive result reported in [12] is that, despite the NP-hardness of the Minimum Local Scenario problem, it can be computed exactly (using our algorithms from Section 5.3) for all of the colorings encountered between *drosophila melanogaster* and *yakuba* (the DCJ distance being roughly 90).

We find that colorings created uniformly at random have high MLS cost, while colorings created using the Hi-C data have low MLS cost. As we introduce randomness to the good colorings, a significant correlation between MLS and the weights of the colorings is observed no matter how many clusters are created. Indeed, the Pearson's correlation is better than 0.77 for all reasonable k, and is as high as 0.92 in some cases.

A significant correlation is also found between the differences $d_{MLPS} - d_{MLS}$, and the weights of the colorings. Further, for the colorings that were optimized on the similarity function this difference never exceeded 4, and no matter the number of colors assigned, it is less than 0.8 on average for both species. The implication is that in many cases MLPS is an optimal solution for both MLS and the problem considered in Section 6. In all cases $\bar{J}$ has less than 25 simple cycles on average, and never more than 300. The hope is that MLS will remain tractable for the more distant genomes.

8 Conclusion and further work

Aside from problems that consider rearrangement length, little is known about weighted rearrangement problems [3, 11, 8, 1, 6]. In [14], we showed that with a simple cost function based on a partition of the adjacencies of one of the genomes into equivalence classes, one can choose – from the exponentially large set of shortest scenarios – a scenario that minimizes the number of moves acting across classes. In this paper we showed that the genome rearrangement problem with an objective function based solely on the cost of DCJs is NP-Hard, even for a simple binary cost function. We gave a 3/2-approximation derived from bounds on the sizes of cycles in a cycle packing of the junction graph. We also presented an exact algorithm and found that an exact solution can be computed between *drosophila*.

This work opens the door to the development of more complex models of genome rearrangement with positional constraints, where local moves would be attributed nonzero cost. To this end we established a useful link between the weighted distance, and the difference between Minimum Local Parsimonious Scenario and Minimum Local Scenario. Experimental results indicate that a problem of finding a minimum cost scenario among those of length only slightly greater than that of a parsimonious scenario might be of practical interest, however further experiments must be conducted to confirm this.

References

1 M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting by length-weighted reversals. *J. of Comp. and System Sciences*, 74(5):744–774, 2008.

2 Anne Bergeron, Julia Mixtacki, and Jens Stoye. *A Unifying View of Genome Rearrangements*, pages 163–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

3 M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172(1):GC11–GC17, 1996.

4 Alberto Caprara. Sorting by reversals is difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, RECOMB '97, pages 75–83, New York, NY, USA, 1997. ACM.

5 Alberto Caprara, Alessandro Panconesi, and Romeo Rizzi. Packing cycles in undirected graphs. *J. of Algorithms*, 48(1):239–256, 2003.

6 G. R. Galvão and Z. Dias. Approximation algorithms for sorting by signed short reversals. In *Proc. of the 5th ACM Conf. on Bioinformatics, Comp. Biology, and Health Informatics*, pages 360–369. ACM, 2014.

7 Ian Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

8 J.-F. Lefebvre, N. El-Mabrouk, E. R. M. Tillier, and D. Sankoff. Detection and validation of single gene inversions. In *Proc. 11th Int'l Conf. on Intelligent Systems for Mol. Biol. (ISMB'03)*, volume 19 of *Bioinformatics*, pages i190–i196. Oxford U. Press, 2003.

9 Erez Lieberman-Aiden, Nynke L. van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R. Lajoie, Peter J. Sabo, Michael O. Dorschner, Richard Sandstrom, Bradley Bernstein, M. A. Bender, Mark Groudine, Andreas Gnirke,

John Stamatoyannopoulos, Leonid A. Mirny, Eric S. Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, Oct 2009.

10 Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2):3336 – 3341, 2009.

11 R. Y. Pinter and S. Skiena. Genomic sorting with length-weighted reversals. *Genome Informatics*, 13:103–111, 2002.

12 Sylvain Pulicani, Pijus Simonaitis, and Krister M. Swenson. Rearrangement scenarios guided by chromatin structure. *Uploaded to bioRxiv*, 2017.

13 Tom Sexton, Eitan Yaffe, Ephraim Kenigsberg, Frédéric Bantignies, Benjamin Leblanc, Michael Hoichman, Hugues Parrinello, Amos Tanay, and Giacomo Cavalli. Three-dimensional folding and functional organization principles of the drosophila genome. *Cell*, 148(3):458–472, Feb 2012.

14 Krister M. Swenson, Pijus Simonaitis, and Mathieu Blanchette. Models and algorithms for genome rearrangement with positional constraints. *Algorithms for Molecular Biology*, 11(1):13, 2016.

15 S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

A Proof of Lemma 15

Proof. We will demonstrate with a help of an example how to obtain such $\hat{A} \in A_+$ and $\hat{B} \in B_+$. We will augment $AG(A,B)$ with adjacencies to obtain a graph AG' which at the end will turn out to be $AG(\hat{A},\hat{B})$. Our working example will consist of the following graphs:

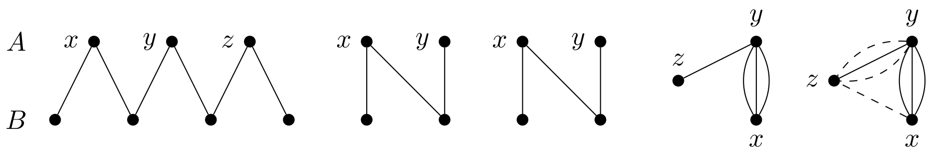
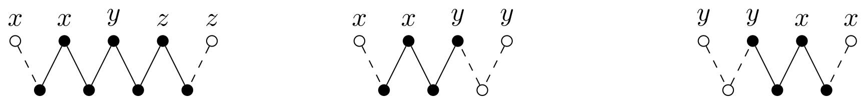
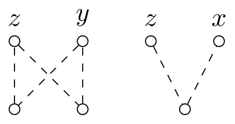


Figure 1 $AG(A,B)$, $J(A,B,col) = J$ and J'.

We first include into AG' every cycle of $AG(A,B)$. Other connected components of $AG(A,B)$ are paths and to these we add new adjacencies at their end points copying their colors to obtain the paths for AG'. In our example $AG(A,B)$ has no cycles and its three paths give paths for AG'



For graphs $G' = (V, E \cup E')$ and $G = (V,E)$ we will note $G' - G = (V,E')$. We take an Eulerian subgraph H of $J' - J$ such that $F = (J' - J) - H$ is a forest. For H we create a union of cycles in AG' giving H as its junction graph. F can be partitioned into paths joining the vertices of an odd degree and for each path in F we create a path consisting of new adjacencies of corresponding colors in AG'. In our example H is a cycle (z,y,z) and F has a single path (z,x) and these add a cycle and a path to AG'.



The vertices of J' have even degrees as it is an Eulerian graph. This guarantees that for every color the number of the pairs added at the ends of the paths in AG' is even. We can group these pairs at the ends of the paths into monochromatic couples and merging these couples we obtain an AG' which is an Eulerian extension of $AG(A, B)$ giving a junction graph loop-equal to J'. A possible grouping of the added end points into monochromatic couples and their merge leads to AG'

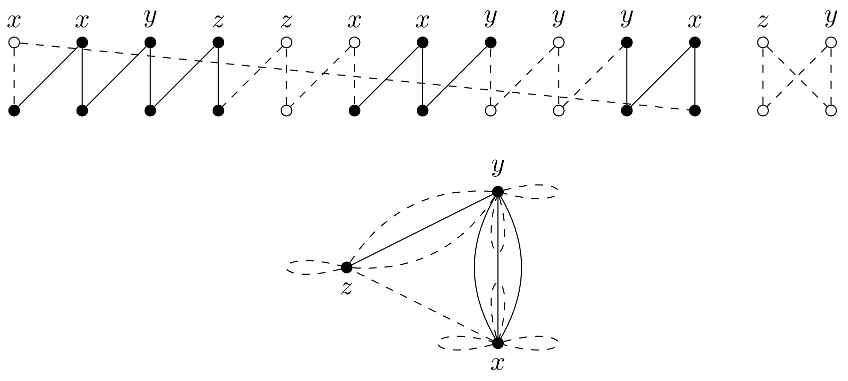


Figure 2 A junction graph obtained from AG' is loop-equal to J'.

Now it is easy to reconstruct $\hat{B} \in B_+$, $\hat{A} \in A_+$ and its coloring $\hat{col}$ such that $AG' = AG(\hat{A}, \hat{B}, \hat{col})$, which guarantees that $J(\hat{A}, \hat{B}, \hat{col})$ is loop-equal to J'. ◀

B Proof of Theorem 18

Proof. For a cycle packing C, we denote the number of loops in it by $L(C)$, the number of the cycles of length 2 by $B(C)$ and the number of longer cycles by $R(C)$. We start by proving Lemma 19

▶ **Lemma 19.** *For every Eulerian graph G*

$$w(G) \geq \frac{2}{3}E(G) - \frac{1}{3}B(G) - \frac{2}{3}L(G).$$

Proof. Using Theorem 8 we obtain a cycle packing C of G such that $w(G) = E(G) - |C|$. We have $|C| = L(C) + B(C) + R(C)$ and $E(G) \geq L(C) + 2B(C) + 3R(C)$ and from this we get

$$w(G) - \frac{2}{3}E(G) = \frac{1}{3}E(G) - |C| \geq -\frac{1}{3}B(C) - \frac{2}{3}L(C), \text{ so}$$
$$w(G) \geq \frac{2}{3}E(G) - \frac{1}{3}B(C) - \frac{2}{3}L(C) \geq \frac{2}{3}E(G) - \frac{1}{3}B(G) - \frac{2}{3}L(G). \quad ◀$$

Now we take a MECP C of J. It covers an Eulerian subgraph J' of J. Using Theorem 16 we have $w_{\text{MLS}} = E(J) - |C|$ and by counting edges and using Theorem 8 we obtain

$$w_{\text{MLS}} = E(J) - |C| = E(J') - |C| + E(J) - E(J') = w(J') + E(J) - E(J')$$

from which using Lemma 19 and a simple counting argument we obtain

$$w_{\text{MLS}} \geq \frac{2}{3}E(J') - \frac{1}{3}B(J') - \frac{2}{3}L(J') + E(J) - E(J') \geq \frac{2}{3}E(J) - \frac{1}{3}B(J) - \frac{2}{3}L(J). \quad ◀$$

Seed-driven Learning of Position Probability Matrices from Large Sequence Sets*

Jarkko Toivonen[1], **Jussi Taipale**[2], **and Esko Ukkonen**[1]

1 **Department of Computer Science, University of Helsinki, Helsinki, Finland**
`jarkko.toivonen@cs.helsinki.fi`

2 **Department of Biosciences and Nutrition, Karolinska Institutet, Stockholm, Sweden**
`jussi.taipale@ki.se`

3 **Department of Computer Science, University of Helsinki, Helsinki, Finland**
`esko.ukkonen@cs.helsinki.fi`

Abstract

We formulate and analyze a novel seed-driven algorithm SeedHam for PPM learning. To learn a PPM of length ℓ, the algorithm uses the most frequent ℓ-mer of the training data as a seed, and then restricts the learning into the ℓ-mers of training data that belong to a Hamming neighbourhood of the seed. The PPM is constructed from background corrected counts of such ℓ-mers using an algorithm that estimates a product of ℓ categorical distributions from a (non-uniform) Hamming sample. The SeedHam method is intended for PPM learning from large sequence sets (up to hundreds of Mbases) containing enriched motif instances. A variant of the method is introduced that decreases contamination from artefact instances of the motif and thereby allows using larger Hamming neighbourhoods. To partially solve the motif orientation problem in two-stranded DNA we propose a novel seed finding rule, based on analysis of the palindromic structure of sequences. Test experiments are reported, that illustrate the relative strengths of different variants of our methods, and show that our algorithm outperforms two popular earlier methods.

Availability and implementation: A C++ implementation of the method is available from `https://github.com/jttoivon/seedham/`

Contact: `jarkko.toivonen@cs.helsinki.fi`

1998 ACM Subject Classification I.2.6 Learning, G.2.1 Combinatorics, I.5.1 Models, J.3 Life and Medical Sciences

Keywords and phrases motif finding, transcription factor binding site, sequence analysis, Hamming distance, seed

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.25

1 Introduction

Position probability matrix (PPM), introduced by Stormo et al [13, 12], is a simple probabilistic model for motifs in biological sequences. PPM represents a product of mutually independent categorical variables, and it is currently the most popular representation, for example, of the DNA motifs for binding sites of transcription factors (collected in motif databases such as Transfac [18] and Jaspar [11]) as well as of motifs in RNA and in protein sequences. Besides PPMs, several other representations of sequence motifs have been

* This work was supported by EU FP7 project SYSCOL (UE7-SYSCOL-258236).

proposed, most of them being various generalizations of the consensus sequence of a motif. Motif representations and their discovery from sequence data is surveyed, e.g., in [9, 14].

In this paper we formulate and analyze a seed-driven algorithm SeedHam for PPM learning. Seed-driven means that a selected seed sequence is used as a starting point of a search that constructs the PPM by analyzing the segments of training data which are similar to the seed. This is in contrast with the well-known alignment method to learn a PPM of length ℓ. This method takes a collection of ℓ-mers, that are supposed to be (somehow verified) instances of the motif, and aligns the ℓ-mers which gives ℓ columns of bases A, C, G, and T. The frequency of each base on each column is counted which gives a position frequency matrix of size $4 \times \ell$. When normalized column-wise, this matrix gives the probability matrix θ for the motif. Assuming that the collection of ℓ-mers is an unbiased sample from the distribution of the motif instances and assuming mutual independence of different positions of the motif, this very simple procedure gives an unbiased estimate of motif distribution.

While verified samples of motif instances are not easily available, there is currently lots of sequence data in which instances of motif(s) are enriched within longer background sequences. Such sequence sets are produced, for example, by high-throughput SELEX [5, 8, 15] or by variants of ChIP-seq [10]. It is possible to learn PPMs from such sequences, by analyzing their over-represented ℓ-mers that are considered instances of the motif. This is what we do in our seed-driven approach to learning PPMs.

We assume that the training data D for learning a PPM is a collection of one or several DNA sequences that contain a relatively high number of instances of the target motif X. Different instance variants should be present in D according to the probability distribution to be learned but the exact locations of the instances within D are not known. The rest of D outside the motif instances is assumed neutral background (although in practice it may contain instances of some other motifs).

Our method locates plausible motif instances in D using the following rule. A most frequent ℓ-mer s of D is taken as the *seed*. Here the motif length ℓ is a user-given constant. Then s as well as the ℓ-mers of D within a short Hamming distance d from s are taken as instances of the target motif X. Using background corrected counts of such ℓ-mers we estimate a PPM that represents X. The aforementioned simple alignment method cannot be used as such because the restriction to a Hamming neighbourhood yields a non-uniform sample of the target distribution.

Our algorithm, called SeedHam, does not contain an iterative search. It is therefore very fast and can learn from large high-throughput sequence sets. An early version of SeedHam method was used for learning PPMs from HT-SELEX sequence sets [5]. Here we present a complete formal definition of a general version of the algorithm as well as extensions for correcting self-overlaps and how to decide the orientation of the motif in two-stranded case. The 'seed-and-wobble' procedure [2] independently developed for the analysis of protein microarray data is analogous to (restricted) SeedHam.

Learning a PPM is complicated by two issues of combinatorial nature. First, if motif X is strongly self-similar, then D may contain lots of artefact instances that overlap the true instances of X [3]. We give an instance elimination technique and associated background correction that decrease contamination from artefacts. Second, in two-stranded DNA the orientation of instances has to be decided. If the Hamming neighbourhoods with radius d of s and its reverse complement $\overline{s}$ do not intersect, we get a heuristic rule for deciding the orientation. We show that if the so-called *palindromic index* of the seed is large enough, then the neighbourhoods become separate for a given d. Algorithm SeedHam+ finds a seed that has high-enough palindromic index at the expense of having sub-maximal count.

The experimental tests illustrate relative strengths of different variants of our methods. We demonstrate the algorithms' capacity to relearn the PPM from simulated data that contains implanted instances of the motif represented by the PPM. The experiments show that the accuracy depends on the Hamming radius d such that the optimal d increases when motif length ℓ increases but decreases when the number of training instances increases.

We compared our methods experimentally with two earlier algorithms, DREME [1] and DECOD [4]. Both are seed-driven, discriminative PPM learning methods that start from a seed and make a heuristic search (beam search in DREME and hill-climbing search in DECOD) to find a PPM that maximizes the discriminative power of the motif to separate between positive and negative training data sets. In a large majority of cases, SeedHam and SeedHam+ relearned the PPM from generated data more accurately than the two other methods.

The paper is organized as follows. After preliminaries in Section 2, Section 3 gives the basic SeedHam algorithm and its artefact eliminating variant. Section 4 introduces palindromic index and its application in selecting orientation, implemented in algorithm SeedHam+. Experimental tests are reported in Section 5.

2 Preliminaries of PPM models

A PPM representing a motif of length ℓ in the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ is a $4 \times \ell$ matrix $\theta = (\theta_{aj})_{a\in\Sigma, j=1,\ldots,\ell}$ such that θ_{aj} gives the occurrence probability of base $a \in \Sigma$ in position j of the motif. Hence each column $\theta_j = (\theta_{\cdot j})$ of θ defines a categorical distribution $\mathrm{Cat}(\theta_j)$ of Σ. The entire matrix θ represents a random variable $X = X(\theta)$ which is a product of ℓ mutually independent categorical random variables X_j:

$$X = X_1 \times X_2 \times \ldots \times X_\ell,$$

where $X_j \sim \mathrm{Cat}(\theta_j)$, and the values of X are in Σ^ℓ.

As the component variables are assumed mutually independent, the probability $P(u) = P_\theta(u)$ of $u = u_1 \ldots u_\ell \in \Sigma^\ell$ of X is

$$P(X = u_1 \ldots u_\ell) = P(X_1 = u_1)P(X_2 = u_2) \cdots P(X_\ell = u_\ell) = \theta_{u_1,1}\theta_{u_2,2} \ldots \theta_{u_\ell,\ell}.$$

Learning of PPMs from DNA sequence data is complicated by the two-stranded structure of DNA. The reverse complement $\overline{u}$ of a sequence $u = u_1 \ldots u_\ell \in \Sigma^\ell$ is sequence $\overline{u_\ell} \ldots \overline{u_1}$ where $\overline{u_j}$ denotes the complementary base of base u_j. Sequence u is *palindromic* if $u_j = \overline{u_{\ell-j+1}}$ for $j = 1, \ldots, \lceil \ell/2 \rceil$. Similarly, a PPM θ is palindromic if $\theta_{aj} = \theta_{\overline{a},\ell-j+1}$ for all (a, j). Note that $P(u) = P(\overline{u})$ if θ is palindromic.

3 SeedHam Algorithm

3.1 Finding a seed and locating motif instances from training data

Let D be the training data (a collection of sequences in Σ^*) that contains enriched amounts of instances of a target PPM motif X of length ℓ, and let s be a sequence of length ℓ. Let $H_d(s) = \{u \in \Sigma^\ell | h(s, u) \le d\}$ be the *Hamming d-neighbourhood* of the sequence s. Here $h(s, u)$ is the Hamming distance of (equal-length) sequences s and u, and the radius d is an integer $\le \ell$. If the given training data is of two-stranded origin, we always denote by D the original data and its reverse complement combined.

Seed-driven motif discovery then proceeds in the following general steps.

1. $s \leftarrow$ a most frequent ℓ-mer of D. Sequence s is selected as the *seed*.
2. For each ℓ-mer $u \in H_d(s)$, $\text{count}(u) \leftarrow$ number of occurrences of u in D.
3. Estimate PPM θ for motif X from the sequences $u \in H_d(s)$ and their counts $\text{count}(u)$.

Detailed implementation of above steps 1 and 2 is possible using elementary techniques that often are fast enough in practice. For big D or ℓ more elaborate implementation techniques from string algorithmics may be needed. We will describe such methods in Subsection 3.3.

The more interesting step 3 is the topic of the next subsection.

3.2 Learning PPM in Hamming neighbourhoods

As the target motif $X(\theta)$ is a product of mutually independent categorical variables $X_1, \ldots, X_\ell$, it follows that, for any $1 \le j \le \ell$,

$$P(X_j) = P(X|X_1 \cdots X_{j-1}X_{j+1} \cdots X_\ell). \tag{1}$$

Consider now the multiset of all ℓ-mers of D. This multiset is a mixture sample of ℓ-mers, some of which are coming from X and the rest come, fully or partially, from the background. Our goal is to learn column θ_j of θ, i.e., we want to estimate the parameters of X_j, for some fixed j. It follows from Equation (1), that by conditioning X on ℓ-mer positions other than j, we get a sample of X_j, possibly contaminated by the background. Let a *j-condition* be any $w = (w_L, w_R)$ such that $w_L \in \Sigma^{j-1}$ and $w_R \in \Sigma^{\ell-j}$. We let $\text{count}_j(a, w)$ denote the number of ℓ-mers $w_L a w_R$ in the data, that is, the number times the symbol $a \in \Sigma$ occurs in context (w_L, w_R) in D.

Then, omitting for a moment the correction for background, we could estimate

$$\theta_{aj} \approx \frac{\text{count}_j(a, w)}{\Sigma_{c \in \Sigma} \text{count}_j(c, w)}. \tag{2}$$

This immediately generalizes to a set of j-conditions, i.e., to several different w combined. We will use conditions taken from a Hamming neighbourhood of the seed s. To minimize contamination from background noise, we restrict the learning to ℓ-mers of D that belong to a small Hamming neighbourhood of s. By the properties of products of categorical variables, such ℓ-mers are likely to have a relatively high count which comes mostly from X and not only from the background as they are small variants of s whose count is the highest.

We restrict the learning of θ to a Hamming neighbourhood $H_d(s)$ of s in D, by using the implied set of j-conditions. For learning θ_j, the set of j-conditions becomes

$$W_{j,d}(s) = \{(w_L, w_R) \in \Sigma^{j-1} \times \Sigma^{\ell-j} \mid h(s, w_L c w_R) \le d \text{ for all } c \in \Sigma\}.$$

As samples for different j-conditions can be combined, (2) becomes

$$\theta_{aj} \approx \frac{\text{count}_j(a)}{\Sigma_{c \in \Sigma} \text{count}_j(c)}, \tag{3}$$

where we have written $\text{count}_j(c) = \Sigma_{w \in W_{j,d}(s)} \text{count}_j(c, w)$.

We have to evaluate (3) for all $j = 1, \ldots, \ell$. It is convenient to organize this such that the contribution of each $u \in H_d(s)$ to the counts is accumulated in one pass that scans through u. Recall that $\text{count}(u)$ denotes the number of occurrences of $u = u_1 \cdots u_\ell \in H_d(s)$ in D. Then it is not difficult to see that the rule for accumulating variables $\text{count}_j(c)$ becomes as follows: if $h(s, u) < d$, then u contributes $\text{count}(u)$ to $\text{count}_j(u_j)$ for all j; if $h(s, u) = d$, then it contributes $\text{count}(u)$ to $\text{count}_j(u_j)$ only for j such that $u_j \ne s_j$.

As data D is only partially covered by ℓ-mers from signal X, and the rest is background, we have at first to correct the counts $\text{count}(u)$ by subtracting estimated contribution from the background. We use here a simple 0-order background model $q = (q_A, q_C, q_G, q_T)$ where each q_c is the frequency of c in D. The background probability of a ℓ-mer $u = u_1 \cdots u_\ell$ is $P_q(u) = \prod_j q_{u_j}$, and the expected number of occurrences of u in a random dataset of the same size as D is $\text{Ecount}(u) = NP_q(u)$, where N is the number of ℓ-mer positions in D.

So we get the following PPM learning algorithm. We say that this algorithm uses *basic counting* of ℓ-mer occurrences, to separate from the counting used in the algorithm variant to be given in Section 3.4.

Algorithm SeedHam

Input: Set of sequences (with reverse complements) D, length of PPM ℓ, Hamming radius d
Output: PPM θ

```
1. $s \leftarrow$ a most frequent $\ell$-mer of $D$
2. for all $u \in H_d(s)$ do $\text{count}(u) \leftarrow \max(0,$ number of occurrences of $u$ in $D - \text{Ecount}(u))$
3. for $j \leftarrow 1, \ldots, \ell$ and $a \in \Sigma$ do $\text{count}_j(a) \leftarrow 0$
4. for all $u = u_1 \cdots u_\ell \in H_d(s)$ do
       if $h(s,u) < d$ then
           for $j \leftarrow 1, \ldots, \ell$ do $\text{count}_j(u_j) \leftarrow \text{count}_j(u_j) + \text{count}(u)$
       else
           for $j \leftarrow 1, \ldots, \ell$ do if $u_j \neq s_j$ then $\text{count}_j(u_j) \leftarrow \text{count}_j(u_j) + \text{count}(u)$
5. for all $j \leftarrow 1, \ldots, \ell$ and $a \in \Sigma$ do $\theta_{aj} \leftarrow \text{count}_j(a) / \sum_{b \in \Sigma} \text{count}_j(b)$
```

Note that SeedHam algorithm differs from the basic alignment method already mentioned in the introduction that aligns the ℓ-mers in the sample, counts the number of occurrences of each element of Σ on each column, and normalizes these counts to get θ. That this algorithm would not estimate θ correctly in a Hamming neighbourhood can be seen, for example, by considering data D that has no motif embedded, the data being background only. A seed s can still be found, but then the standard algorithm applied on, say, $H_1(s)$ would produce a PPM that gives for s a clearly higher probability than for the other ℓ-mers while the correct model should give uniform distribution. Algorithm SeedHam produces such a uniform model in this case; an illustration is given in Figs 1a and 1b. The standard algorithm does so only if the Hamming neighbourhood does not leave any data out, that is, if $H_\ell(s)$ is used.

3.3 Implementation and complexity

A most frequent ℓ-mer s as well as the counts $\text{count}(u)$ for ℓ-mers $u \in H_d(s)$ can be found in linear time $O(|D|)$ using for example suffix-trees (or suffix arrays): First, construct the suffix-tree of D, and associate with each node x of the tree the number $S(x)$ of leaves in the subtree of x and the length $L(x)$ of the sequence represented by the path from the root to x. This can be done in linear time using well-known algorithms [17, 7, 16]. Second, find from the suffix-tree the most frequent ℓ-mer s of D. This can be done by finding the node x such that $L(x) \geq \ell$ and $S(x)$ is largest possible. Then s is the prefix of length ℓ of the sequence represented by the path from the root to x. This again takes linear time. Third, find the counts of ℓ-mers $u \in H_d(s)$ by a depth-first traversal of the tree. Each branch is followed until the Hamming distance between s and the sequence spelled out by the current depth-first search node is $> d$, or a ℓ-mer $u \in H_d(s)$ is found. Then $\text{count}(u) = S(x)$ where x is the node corresponding to u.

Obviously, this search finds all members of $H_d(s)$ that occur in D. Then the learning part (Steps 3, 4) of Algorithm SeedHam can be performed. The search and learning takes time proportional to the total length of different ℓ-mers of D. Hence the total time requirement becomes $O(|D| + \ell \cdot \min(|D|, \sum_{h=0}^{d} \binom{\ell}{h}(|\Sigma| - 1)^h))$.

As the overhead of suffix-tree algorithms may be large, a straightforward tabulating algorithm, possibly with hashing techniques, can be used for D of modest size to find counts $\text{count}(u)$ and the seed s, after which the learning part of Algorithm SeedHam can be performed. Again, the running time becomes $O(\ell|D|)$.

3.4 Elimination of artefact instances

Here we make a more accurate analysis of the mixing of instances of X and the background on training data D. The multiset of ℓ-mers of D consists of three types of ℓ-mers: (i) ℓ-mers that are instances of X; (ii) ℓ-mers that are completely outside the instances of X; (iii) ℓ-mers that overlap both an instance of X and background.

When X has strong self-overlaps, D has tendency to have artefact instances of X in category (iii). As an extreme example, consider X = AAAAAAAA, i.e, each position of X has A with probability 1. Then D has lots of 8-mers AAAAAAAA and, by one-symbol shift, lots of 8-mers CAAAAAAA, GAAAAAAA, and TAAAAAAA. If not eliminated, such artefact instances at distance 1 from the seed would leak to the learned X such that C, G, and T will get clearly non-zero probability at position 1.

To avoid counting self-overlapping artefact instances, we introduce a dominance relation of ℓ-mers. Let g be a positive integer giving maximum shift in a self-overlap that is still considered significant (note that if a shift is large and hence the self-overlap is short, then the implied bias in counts rapidly gets very small). We say that the ℓ-mer u in position i of D *dominates*, if $h(s, u) < h(s, v)$ for all ℓ-mers v that are located in $\pm g$ proximity of i in D, that is, in positions $i - g, \ldots, i - 1, i + 1, \ldots, i + g$ of D.

For an example of dominance, consider a single-stranded data in the setting $\ell = 4$, s = AACG, $g = 2$, and $d = 2$. When counting the dominating occurrences of the 4-mer AGTG $\in H_d(s)$, we need to take the context of the occurrence into account. If we see a string TTAGTGAA in the data, we count this occurrence of AGTG as it dominates here: all the other 4-mers of this string obviously have Hamming distance from the seed greater than 2. But in the context TAAGTGAA the 4-mer AGTG is not counted, because, for instance, $h(\text{TAAG}, s) = 2$ as well.

We then slightly modify the SeedHam algorithm: for each ℓ-mer u, include into its occurrence count only the dominating occurrences of u in D. This way of counting is called *dominance counting*, and the resulting count of an ℓ-mer u is denoted as $\text{count}_{\text{dominant}}(u)$. Intuitively, this rule means that the true instances of X are assumed to locate in D more than g positions apart and to dominate in their $\pm g$ proximity.

Dominance counting needs an accordingly modified background correction. We denote by $\lambda \in [0, 1]$ the relative abundance of the motif instances in D. Hence λN of the N ℓ-mer sites of D are from X. Then the expected artefact count of an ℓ-mer u can be written as the sum of occurrence counts in ℓ-mer categories (ii) and (iii):

$$\text{Ecount}_{\text{dominant},\lambda}(u) = (1 - \lambda(2(\ell + g) - 1))N \cdot P_q(u) + \lambda N \sum_{\substack{j=-(\ell+g-1) \\ j \neq 0}}^{j=\ell+g-1} P_{T_j,\text{dominant}}(u),$$

where $\ell^* = \ell + 2g$ and T_j is a $4 \times \ell^*$ PPM built from θ and q as described below. Note that $j = 0$, which defines the category (i), is excluded. For any ℓ-mer u and a $4 \times \ell^*$ PPM T, $P_{T,\text{dominant}}(u)$ is defined as

$$P_{T,\text{dominant}}(u) = \sum_{v \in \Sigma^{\ell *}} [v(0) = u] \prod_{\substack{j=-g \\ j \neq 0}}^{g} \prod_{k \in \{-1,1\}} [h(v(j), s^k) > h(u, s)] P_T(v),$$

where $[\cdot]$ are Iverson's brackets, s^k is the seed in direction k, and $v(j)$ is the ℓ-mer that starts from position j of v. The PPMs $T_j, j = -(\ell + g - 1), \ldots, \ell + g - 1$, are models for the alternative ways how a string of length ℓ^* can overlap in D the boundary between motif instance and background. PPM T_0 has θ in the middle, with g columns of q before and after it. PPM T_j for $j > 0$ has θ shifted j positions to the left from the center and for $j < 0$, j positions to the right. Note that our technique here is similar to DECOD [4].

As θ is still unknown when the background correction has to be made, we use uncorrected θ to build models T_j. We solve the parameter λ numerically from the equation

$$\text{count}(u) = \text{Ecount}_\lambda(u) := (1 - \lambda(2\ell - 1))N \cdot P_q(u) + \lambda N \sum_{j=-(\ell-1)}^{j=\ell-1} P_{R_j,\lambda}(u),$$

where $R_{j,\lambda}$ is the background corrected PPM generated from corrected counts

$$\text{count}_{\text{dominant}}(u) - \text{Ecount}_{\text{dominant},\lambda}(u).$$

The running time of background correction is exponential in $\ell^* = \ell + 2g$. Therefore in our implementation we do the correction only for $\ell \leq 10$. Since for longer motifs the effect of background is very small for typical data sizes, this restriction has no big effect on the accuracy of the learned models. We have used $g = 4$ as a default.

Figs 1c and 1d give an example of the effect of dominance counting: for a PPM AAAAAAAA, 10 000 instances were generated to create a total data of length 400 000 bp (hence $\lambda = 0.025$). The PPM relearned by basic SeedHam shows clear contamination from artifact instances while SeedHam with dominance counting removes it.

4 Motif Orientation in Two-Stranded Case: SeedHam+ algorithm

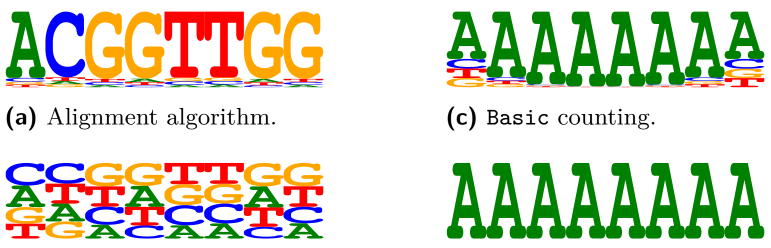
PPM discovery is in practice complicated by the two-strandedness of DNA. Although the motif itself may have direction (for example, a transcription factor binds to DNA in a specific orientation), it is not possible to infer the direction from motif instances that may occur equally in both strands of the DNA. Moreover, an instance in one strand means that we see the reverse complement of it in the other strand. In fact, the counts of a ℓ-mer u and its reverse complement $\overline{u}$ are always equal if the counts are taken along both strands of DNA.

This symmetry should be broken such that we use in the PPM learning only ℓ-mers that have the same direction with respect to the underlying motif. Otherwise we would always get palindromic PPMs.

4.1 Selection of the orientation

The following heuristic could be used for resolving the orientations: To select between u and $\overline{u}$, take the one whose Hamming distance to the seed s is shorter, that is, if $h(s, u) < h(s, \overline{u})$ then take u, and if $h(s, \overline{u}) < h(s, u)$ then take $\overline{u}$.

For this rule to work it is necessary that $h(s, u) \neq h(s, \overline{u})$, which means that s may not be a palindrome. For palindromic seeds we have the following observation.



(a) Alignment algorithm.

(b) SeedHam algorithm.

(c) `Basic` counting.

(d) `Dominance` counting.

Figure 1 (a&b) Learning a PPM from training data that has only uniform background but no motif. Using a seed ACGGTTGG, the alignment algorithm finds a PPM in which the seed dominates while SeedHam correctly finds a PPM that represents the uniform background. **(c&d)** Contamination of the learned model due to artefact occurrences. SeedHam algorithm with basic counting (above) and with dominance counting (below) was used. Both methods used Hamming radius one, and the training data consisted of a single sequence that had the single motif instance AAAAAAAA implanted at every 40th position 10 000 times. Other positions were filled with random uniform background. Dominance counting effectively removes the contamination due to artefact occurrences.

▶ **Theorem 1.** *If the seed s in Algorithm SeedHam is a palindrome, then the resulting PPM θ is palindromic.*

Proof. Let s be a palindrome. Then $H_d(s) = H_d(\overline{s})$, and a sequence $u = u_1 \dots u_\ell$ is in $H_d(s)$ iff $\overline{u}$ is in $H_d(s)$. Then $u = u_1 \dots u_j \dots u_\ell$ contributes 1 to the count of $\theta_{u_j,j}$ iff $\overline{u} = \overline{u_\ell} \dots \overline{u_j} \dots \overline{u_1}$ contributes 1 to the count of $\theta_{\overline{u_j},\ell-j+1}$ which is the element of θ that is palindrome symmetric to $\theta_{u_j,j}$. This is because $h(s_{\neg j}, u_{\neg j}) = h(\overline{s_{\neg j}}, \overline{u_{\neg j}})$, and hence $u_{\neg j}$ is used as a condition iff $\overline{u_{\neg j}}$ is used, where $u_{\neg j}$ denotes the sequence $u_1 \dots u_{j-1} u_{j+1} \dots u_\ell$. Palindrome symmetric elements of θ get equal counts which proves our claim. ◀

Palindromic index $\mathrm{Pi}(u)$ of a sequence $u \in \Sigma^\ell$ is defined as $\mathrm{Pi}(u) = h(u, \overline{u})$. If $|u| = \ell$ is odd then $\mathrm{Pi}(u) \geq 1$ as the symbols in the center position of u and $\overline{u}$ are always different. For even $|u|$, $\mathrm{Pi}(u)$ has an even value $0, 2, \dots, |u|$. For odd $|u|$, $\mathrm{Pi}(u)$ has an odd value $1, 3, \dots, |u|$.

If $\mathrm{Pi}(u) = 0$ (and hence $|u|$ is even), then u is a (DNA) palindrome.

We call a set $Q \subseteq \Sigma^\ell$ *conflict-free* if $Q \cap \overline{Q}$ is empty, i.e., if $u \in Q$ then $\overline{u} \notin Q$. So if $H_d(s)$ is conflict-free then all $u \in H_d(s)$ are such that $h(s, u) < h(s, \overline{u})$. Hence Algorithm SeedHam with such an $H_d(s)$ implicitly applies our rule for choosing between u and $\overline{u}$.

▶ **Theorem 2.**

(a) *If $d < \mathrm{Pi}(s)/2$ then $H_d(s)$ is conflict-free.*

(b) *If $\mathrm{Pi}(s) = 2$ and $d = 1$, then $H_d(s) \cap H_d(\overline{s})$ contains exactly two sequences and these sequences are palindromes.*

Proof. (a) To derive a contradiction, let $d < \mathrm{Pi}(s)/2$ and assume that there is a sequence u in $H_d(s) \cap H_d(\overline{s})$. Then $\mathrm{Pi}(s) = h(s, \overline{s}) \leq h(s, u) + h(u, \overline{s}) \leq 2d$ which contradicts the assumption that $d < \mathrm{Pi}(s)/2$.

(b) $\mathrm{Pi}(s) = 2$ implies that ℓ must be even and that $s_j = \overline{s_{\ell-j+1}}$ except for one value $j \leq \ell/2$. Then $s_j \neq \overline{s_{\ell-j+1}}$ and hence $\overline{s_j} \neq s_{\ell-j+1}$. Then sequences $s(s_j|\overline{s_{\ell-j+1}})$ and $s(s_{\ell-j+1}|\overline{s_j})$ are different palindromes and have distance 1 from both s and $\overline{s}$.

A sequence in $H_1(s) \cap H_1(\overline{s})$ should be the intermediate sequence on the two step path from s to $\overline{s}$ that transforms s to $\overline{s}$ using one symbol changes. There are exactly two such paths, one changing first s_j and then $s_{\ell-j+1}$, and the other changing first $s_{\ell-j+1}$ and then s_j. This gives the above two sequences. ◀

According to Theorem 2 (a), a conflict-free $H_d(s)$ for Hamming radius $d \geq 3$ requires a seed s such that $\mathrm{Pi}(s) \geq 2d+1$. To achieve this, one could use in Algorithm SeedHam as seed s the ℓ-mer that has the largest count among the ℓ-mers whose palindromic index is $\geq 2d+1$, provided that the count of such a ℓ-mer is sufficiently large. Note that increasing the palindromic index of s may make the count of s smaller, which means that Algorithm SeedHam would utilize a smaller fraction of the data.

For a large data D already Hamming radius $d = 1$ can give an accurate estimate of the PPM. Then, by Theorem 2 (b), we only need a seed s such that $\mathrm{Pi}(s) \geq 2$. However, $H_1(s)$ is not conflict-free as $H_1(s) \cap H_1(\bar{s})$ is not empty but contains palindromes. Fortunately, deciding their orientation is not needed, as palindromes are symmetric and hence there is only one orientation for them. One only has to divide their observed counts by two as each occurrence of a palindrome also appears on the opposite DNA strand and hence is counted twice.

To conclude the above discussion we give Algorithm SeedHam modified such that it uses a seed with high-enough palindromic index to avoid orientation conflicts. Only step 1 needs changes.

Algorithm SeedHam+

Input: Set of sequences (with reverse complements) D, length of PPM ℓ, Hamming radius d, count threshold $m = \max(20, N4^{-\ell} + 2\sqrt{N4^{-\ell}(1-4^{-\ell})})$, i.e., two standard deviations more than expected by uniform background and at least 20.
Output: PPM θ

```
1. $r \leftarrow$ if $d = 1$ then 2 else $2d+1$.
   $s \leftarrow$ $\ell$-mer $u$ such that $\mathrm{Pi}(u) = r$ and the count of $u$ in $D$ is largest possible. However, if
   this count is $< m$ then $s \leftarrow$ $\ell$-mer $u$ with largest palindromic index among $\ell$-mers whose
   count is $\geq m$.
2.-5. As Algorithm SeedHam.
```

5 Experimental evaluation

To compare the performance of algorithms SeedHam and SeedHam+ we randomly selected from the article of Jolma *et al* [6] altogether 24 PPMs, eight PPMs of each of lengths $\ell = 8$, 13, and 18. For each PPM, four data sets were randomly generated: number of motif occurrences being either 100 or 10 000 and occurrences oriented either in single direction or in both directions. The occurrences were placed starting at every 40th base in a single sequence. Between motif occurrences uniform random background was used. This long sequence was then used as training data D for SeedHam and SeedHam+ using either basic or dominance counting. The learned PPMs were compared against the originals, and the learning error was measured using the maximum norm (i.e., maximum absolute value of the difference of the corresponding entries of the original and the learned PPM; a shift of -1, 0, +1 columns was allowed and minimum of the max-norm over these shifts was taken as the learning error). Hamming radii $d \in \{1, \ldots, 5\}$ were used in the experiments.

The results from these experiments are reported in Fig. 2. When comparing Figs 2a and 2b, the difficulty of learning the model in the two-directional case is clearly visible.

Table 1 Changes of palindromic index in the SeedHam+ experiments on generated data. For each length, eight factors and two data set sizes and five Hamming radii give 80 experiments in total. The table shows that the required palindromic index is difficult to reach. It is easier for $d = 1$ but rapidly gets difficult with larger d.

	Length 8	Length 13	Length 18
Need for increase	52/80	28/80	24/80
No increase possible	42/52	12/28	12/24
Did not increase enough	4/52	8/28	8/24
Increased enough	6/52	8/28	4/24

Introducing the seed with optimized palindromic index (Fig. 2c) does help a little bit, but with the motif length and dataset sizes we used, it was not always possible to optimize the seed, see Table 1. This is because moving far away (in Hamming distance) from the most common ℓ-mer, decreases the count of seed s (and counts in $H_d(s)$) too much, and hence gives inaccurate estimate of the motif.

Algorithms are fast in practice. On average it took less than a second of CPU time for the basic counting method and 7 minutes of CPU time for the dominance counting method per PPM, when SeedHam was run on a generated data set with 10 000 occurrences (400 000 bp). The longer running time of dominance counting is due to the relatively slow background correction. For motifs of length larger than 10 this correction can be lifted. Performance testing was done using a single thread on Intel Xeon CPU X7350 running at 2.93 GHz.

We also applied SeedHam to real SELEX data. We used data sets from [6] that were used by Jolma *et al* to obtain the previously mentioned 24 PPMs. SeedHam showed in these experiments consistent and robust performance with quite small differences between different variants of the method.

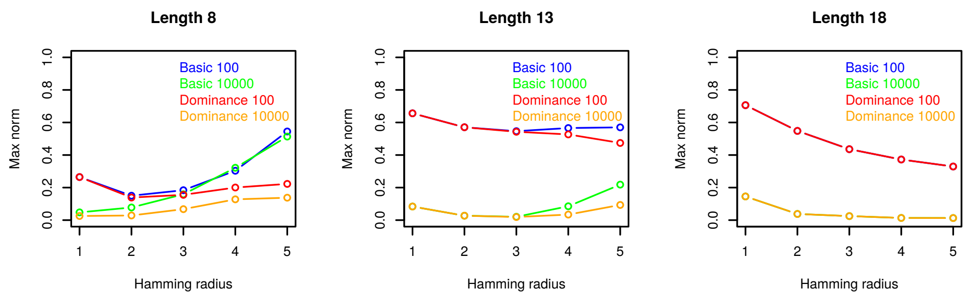
Our experimental evaluation suggests that in practice for large data already Hamming radius $d = 2$ gives accurate results and larger d do not improve too much. For small data this holds true for short motifs but for longer ones using larger d would improve accuracy. For larger d the dominance version of SeedHam clearly improves accuracy.

Results of comparison of SeedHam with earlier algorithms DREME [1] and DECOD [4] are given in Table 2. SeedHam used dominance counting in these experiments. The generated training data contained 1000 instances of the motif, implanted into sequences of length 40. The same motifs were used as in the experiments of Fig. 2. Hamming radius d used in the experiments was 1, 3, and 5 for motif lengths 8, 13, and 18, respectively. SeedHam gives the most accurate result in 21 cases out of 24, often with clear margin. We also tested SeedHam+ with dominance counting. Then the learning accuracy of the LMX1B motif improved to 0.14, making this, too, the best among the compared results.

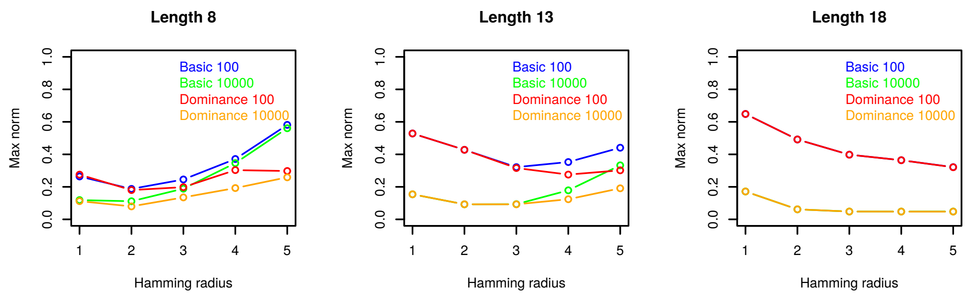
6 Discussion

We gave a general formulation and analyzed the PPM learning algorithm SeedHam that restricts the use of the training data only to a small Hamming neighbourhood $H_d(s)$ of a seed s. We also introduced a dominance counting technique to correct for artifact occurrences of self-similar motifs, as well as a novel seed selection rule, based on the palindromic index, that gives seeds such that the orientation of the motif instances restricted to $H_d(s)$ in two-stranded DNA is unambiguous.

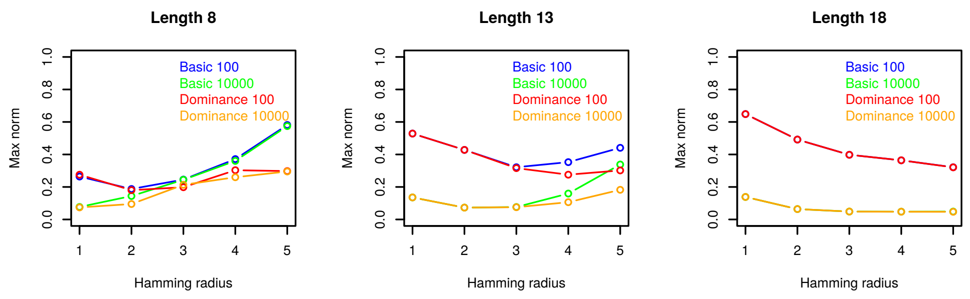
Even if we have chosen a seed with optimal palindromic index, there is still another source of inaccuracy due to the two-strandedness of DNA. When we see a (putative) motif occurrence



(a) Single direction.



(b) Both directions.



(c) Both directions using SeedHam+.

Figure 2 Two data sets were generated for each given PPM: one with 100 random occurrences and another with 10 000 random occurrences of the motif. The occurrences were placed in the same orientation starting at every 40th position. The gaps between the occurrences were filled with uniform random background. The three columns of the figure correspond to experiments with sets of 8 motifs of lengths 8, 13, and 18, respectively. The average maximum error between the original and the relearned models is shown for each Hamming radius, each size of data set, and both methods of counting the number of occurrences of sequences in the Hamming neighbourhood, namely, the basic and dominance counting. Background subtraction was only applied to motif length 8. **(a)** Accuracy of SeedHam algorithm for data with single direction of motif occurrences. For motif lengths 8 and 13 the benefits of the dominance counting over the basic counting becomes visible as the Hamming radius increases. For length 18, the used Hamming radii were not large enough to show any difference between the two counting methods. The accuracy of the learned models is very good. Different motif lengths show different behaviour with respect to the Hamming radius. With every motif length, increasing the Hamming radius initially gives more accurate result, but after some point the increased contamination from the background starts to weaken the result. With factor length 18, however, the dataset would have to be very large for the background to have an effect on the learned motif. **(b)** Accuracy of SeedHam algorithm for data with motif occurrences in random orientation. As expected, the learning error is larger than in the case of single direction; note, however, that for $\ell = 13$, data size 100, the accuracy is here better! **(c)** Accuracy of SeedHam+ algorithm for the same data as in panel (b), i.e., optimization of the palindromic index of the seed enabled. The effect of higher palindromic index is in general minor, except for $\ell = 8, d = 1$ in which case we get here the best accuracy.

Table 2 Comparison of SeedHam to DREME and DECOD. Data sets of 1000 sequences of length 40 were generated using original motifs of TFs indicated below (note that the factor RARA had two distinct motifs). One occurrence of the motif was planted in each sequence. The distances are maximum element-wise distances between the original PPM and the PPM relearned from the generated data. DREME was only run for motifs of length 8 which is its maximum recommended motif length.

Factor	DREME	DECOD	SeedHam dominance
DLX5	0.16	0.26	0.32
GATA3	0.38	0.50	0.10
ISL2	0.22	0.41	0.08
LMX1B	0.44	0.24	0.32
MEIS2	0.16	0.46	0.03
MSX1	0.08	0.35	0.08
NR2F1	0.04	0.55	0.04
OTX1	0.20	0.97	0.06

(a) Motif length 8.

Factor	DECOD	SeedHam dominance
DUXA	0.38	0.07
EOMES	0.55	0.05
LBX2	0.16	0.13
LHX9	0.37	0.36
NFKB1	0.88	0.05
NR2F1	0.50	0.06
PRDM4	1.00	0.01
ZNF238	0.50	0.05

(b) Motif length 13.

Factor	DECOD	SeedHam dominance
CUX1	0.18	0.23
E2F3	0.44	0.03
ETS1	0.22	0.09
KLF13	0.50	0.04
MGA	0.14	0.06
MSX1	0.35	0.06
RARA	0.50	0.02
RARA	0.55	0.01

(c) Motif length 18.

u, that belongs to $H_d(s)$, on a DNA strand, it is not possible to decide, without additional information, whether or not the motif occurrence really is this u or the reverse complement $\overline{u}$ on the opposite strand. In the latter case the occurrence should not be used when building the model. Hence, the occurrence count is necessarily a mixture of counts from u and $\overline{u}$. It is not possible to directly resolve the mixture as the mixing proportions of how much of the count comes from u and how much from $\overline{u}$ are unknown. Mixing proportion depends on the probabilities $P(u)$ and $P(\overline{u})$ of u and $\overline{u}$ to occur as motif instances. A subject for further study is to incorporate into the SeedHam algorithm a maximum likelihood estimator for improved resolution of the mixture.

References

1 Timothy L. Bailey. Dreme: motif discovery in transcription factor chip-seq data. *Bioinformatics*, 27(12):1653, 2011.

2 Michael F. Berger, Anthony A. Philippakis, Aaron M. Qureshi, Fangxue S. He, Preston W. Estep, and Martha L. Bulyk. Compact, universal DNA microarrays to comprehensively determine transcription-factor binding site specificities. *Nature Biotech.*, 24(11):1429–1435, 2006.

3 Mathieu Blanchette and Saurabh Sinha. Separating real motifs from their artifacts. *Bioinformatics*, 17(SUPPL. 1), 2001.

4 Peter Huggins, Shan Zhong, Idit Shiff, Rachel Beckerman, Oleg Laptenko, Carol Prives, Marcel H. Schulz, Itamar Simon, and Ziv Bar-Joseph. DECOD: fast and accurate discriminative DNA motif finding. *Bioinformatics*, 27(17):2361, 2011.

5 Arttu Jolma, Teemu Kivioja, Jarkko Toivonen, et al. Multiplexed massively parallel SELEX for characterization of human transcription factor binding specificities. *Genome Res.*, 20(6):861–873, 2010.

6 Arttu Jolma, Jian Yan, Thomas Whitington, Jarkko Toivonen, et al. DNA-binding specificities of human transcription factors. *Cell*, 152(1–2):327–339, 2013.

7 Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

8 Arnold R. Oliphant, Christopher J. Brandl, and Kevin Struhl. Defining the sequence specificity of DNA-binding proteins by selecting binding sites from random-sequence oligonucleotides: analysis of yeast GCN4 protein. *Mol. Cell. Biol.*, 9(7):2944–2949, 1989.

9 Giulio Pavesi, Giancarlo Mauri, and Graziano Pesole. In silico representation and discovery of transcription factor binding sites. *Brief. Bioinformatics*, 5(3):217–236, 2004.

10 Gordon Robertson, Martin Hirst, Matthew Bainbridge, Misha Bilenky, Yongjun Zhao, Thomas Zeng, Ghia Euskirchen, Bridget Bernier, Richard Varhol, Allen Delaney, Nina Thiessen, Obi L. Griffith, Ann He, Marco Marra, Michael Snyder, and Steven Jones. Genome-wide profiles of STAT1 DNA association using chromatin immunoprecipitation and massively parallel sequencing. *Nat. Methods*, 4(8):651–657, 2007.

11 Albin Sandelin, Wynand Alkema, Par Engstrom, Wyeth W. Wasserman, and Boris Lenhard. JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Res.*, 32(Database issue):D91–94, 2004.

12 Gary D Stormo. DNA binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

13 Gary D. Stormo, Thomas D. Schneider, Larry Gold, and Andrzej Ehrenfeucht. Use of the 'Perceptron' algorithm to distinguish translational initiation sites in E. coli. *Nucleic Acids Res.*, 10(9):2997–3011, 1982.

14 Martin Tompa, Nan Li, Timothy L. Bailey, George M. Church, Bart De Moor, Eleazar Eskin, Alexander V. Favorov, Martin C. Frith, Yutao Fu, W. James Kent, et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature biotechnology*, 23(1):137–144, 2005.

15 Craig Tuerk and Larry Gold. Systematic evolution of ligands by exponential enrichment: RNA ligands to bacteriophage T4 DNA polymerase. *Science*, 249(4968):505–510, 1990.

16 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

17 Peter Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11, 1973.

18 Edgar Wingender. The TRANSFAC project as an example of framework technology that supports the analysis of genomic regulation. *Brief. Bioinformatics*, 9(4):326–332, 2008.

Improved De Novo Peptide Sequencing using LC Retention Time Information

Yves Frank[†1], **Tomas Hruz**[2], **Thomas Tschager**[*3], **and Valentin Venzin**[†4]

1 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
2 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
3 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
4 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**

Abstract

Liquid chromatography combined with tandem mass spectrometry (LC-MS/MS) is an important tool in proteomics for identifying the peptides in a sample. Liquid chromatography temporally separates the peptides and tandem mass spectrometry analyzes the peptides, that elute one after another, by measuring their mass-to-charge ratios and the mass-to-charge ratios of their prefix and suffix fragments. De novo peptide sequencing is the problem of reconstructing the amino acid sequences of the analyzed peptide from this measurement data. While previous approaches solely consider the mass spectrum of the fragments for reconstructing a sequence, we propose to also exploit the information obtained from liquid chromatography. We study the problem of computing a sequence that is not only in accordance with the experimental mass spectrum, but also with the retention time of the separation by liquid chromatography. We consider three models for predicting the retention time of a peptide and develop algorithms for de novo sequencing for each model. An evaluation on experimental data from synthesized peptides for two of these models shows an improved performance compared to not using the chromatographic information.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, J.3 Life and Medical Sciences: Biology and Genetics

Keywords and phrases Computational proteomics, Peptide identification, Mass spectrometry, De novo peptide sequencing, Retention time prediction

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.26

1 Introduction

The amino acid sequences of peptides in a sample can be analyzed with the following tandem mass spectrometry experiment [7]. First, the peptides are separated temporally by liquid chromatography. Then, the mass spectrometer measures the mass-to-charge ratio of a peptide and fragments multiple copies of it at random positions. Finally, the mass spectrometer measures the mass-to-charge ratio of the resulting fragments. The peptide sequencing problem is to reconstruct the amino acid sequence of the peptide from the experimental data. This problem has been extensively studied [5, 18]. Nevertheless, when analyzing unknown peptides the otherwise very successful database search approach is not applicable and de novo sequencing, which is the reconstruction of the whole sequence from scratch, is necessary.

[*] Corresponding author: `thomas.tschager@inf.ethz.ch`.
[†] Preliminary work for this study was carried out during the bachelor theses of Y. Frank and V. Venzin.

Several algorithms for de novo sequencing [2, 1, 6, 10] consider the differences of the peptide's fragment masses to reconstruct the peptide's sequence. Various scoring functions have been proposed that try to exploit as much information as possible from the mass spectrum of the fragments to find a sequence that explains the observed spectrum as well as possible. However, the information obtained from the chromatographic separation in the first step of the experiment is not considered by these scoring functions.

In liquid chromatography, the peptides in a sample have to pass through a column. The time a peptide needs to traverse the column is called *retention time* and depends on the chemical properties of the peptide. This process results in the temporal separation of the peptides in a sample. Predicting the retention time of a peptide from its amino acid sequence is a challenging task [14, 11]. Several studies use retention time prediction models for peptide sequencing as a filtering step after a database search to increase the confidence of identification and to identify false positive identifications [12, 16].

However, to the best of our knowledge, the retention time information has not been considered by de novo peptide sequencing algorithms. This information can be useful, because it allows to reconstruct parts of a sequence that cannot be resolved by mass spectrometry (e.g. amino acids and fragments with equal masses). Moreover, it is available without additional experimental effort. However, simply filtering the solutions of a standard de novo sequencing algorithm by predicted retention time is not an option, as it requires to compute all possible solutions in the worst case to find an optimal solution. We formulate and study a de novo sequencing problem that integrates the retention time as an additional constraint and does not require filtering many candidates. We are interested in a sequence that both matches the experimental spectrum and the measured retention time. We consider three additive retention time prediction models and develop algorithms for each model.

In this study, we do not aim for a replacement for available de novo sequencing tools, but rather explore ways of exploiting the retention time information in de novo sequencing algorithms. We evaluate the performance of two algorithms on experimental measurements from synthesized peptides. In our evaluation, we consider a basic scoring function to clearly expose the impact of using retention time prediction models. We compare our algorithms to DeNovoΔ [4, 17], an algorithm that considers the same symmetric difference scoring model but no retention time information. This scoring model shows improved identification rates compared to the prevalent shared peak count scoring model [1]. For the third prediction model, we present some preliminary results.

Considering the retention time information comes at the cost of higher computational effort and requires additional parameters for retention time prediction (either estimated from suitable datasets or taken from the literature). Yet, we believe that it is useful to exploit retention time information for peptide identification and to further study the integration of retention time information in algorithms for de novo peptide sequencing.

2 Notation and Problem Definition

In this paper, we model amino acids by characters and peptides by strings. We consider an alphabet Σ of characters. A string $\mathtt{S} = \mathtt{a}_1 \ldots \mathtt{a}_n$ is a sequence of characters. The empty string is denoted by $\mathtt{S}_\emptyset$. Every character $\mathtt{a} \in \Sigma$ has a mass $m(\mathtt{a}) \in \mathbb{R}^+$. The mass of a string $\mathtt{S} = \mathtt{a}_1 \ldots \mathtt{a}_n$ is the sum of its character's masses $m(\mathtt{S}) = \sum_{i=1}^{n} m(\mathtt{a}_i)$. The empty string $\mathtt{S}_\emptyset$ has mass 0. A substring of $\mathtt{S}$ is denoted by $\mathtt{S}_{\mathtt{i},\mathtt{j}} = \mathtt{a}_\mathtt{i} \ldots \mathtt{a}_\mathtt{j}$ for $1 \leq i \leq j \leq n$. The prefix set $\mathrm{Pre}(\mathtt{S})$ contains all prefixes of $\mathtt{S}$ including the empty string, i.e. $\mathrm{Pre}(\mathtt{S}) = \Cup_{i=1}^{n} S_{1,i} \cup \{S_\emptyset\}$. The *theoretical spectrum* of $\mathtt{S}$ is the union of all its prefix and suffix masses $\mathrm{TS}(\mathtt{S}) =$

(a) Linear: $t_{\text{lin}}(\mathtt{S}) = t(\mathtt{A}) + t(\mathtt{I}) + t(\mathtt{A}) + t(\mathtt{G}) + t(\mathtt{A}) + t(\mathtt{K})$

(b) Position-dependent: $t_{\text{pos}}(\mathtt{S}) = t_{\text{pre}}(\mathtt{A}, 1) + t_{\text{pre}}(\mathtt{I}, 2) + t(\mathtt{A}) + t(\mathtt{G}) + t_{\text{suf}}(\mathtt{A}, 2) + t_{\text{suf}}(\mathtt{K}, 1)$

(c) Neighborhood-based: $t_{\text{nei}}(\mathtt{S}) = t(-, \mathtt{A}) + t(\mathtt{A}, \mathtt{I}) + t(\mathtt{I}, \mathtt{A}) + t(\mathtt{A}, \mathtt{G}) + t(\mathtt{G}, \mathtt{A}) + t(\mathtt{A}, \mathtt{K}) + t(\mathtt{K}, -)$

Figure 1 Retention time prediction for string $\mathtt{S} = \mathtt{AIAGAK}$. (a) In the linear model, the retention time of a string is the sum of its character's coefficients. (b) In the position-dependent model (with $\gamma = 2$), the position of the first and the last two characters is considered additionally. (c) The neighborhood-based model considers all pairs of consecutive characters in a string. The first and the last character have additional coefficients, as they only have one adjacent character.

$\{m(\mathtt{T}), m(\mathtt{S}) - m(\mathtt{T}) \mid \mathtt{T} \in \text{Pre}(\mathtt{S})\}$. Note that for every prefix $\mathtt{T} \in \text{Pre}(\mathtt{S})$ the string $\mathtt{S}$ has a complementary suffix of mass $m(\mathtt{S}) - m(\mathtt{T})$. We say a mass m is *explained* by $\mathtt{S}$ if $m \in \text{TS}(\mathtt{S})$.

We define three simple models for predicting the retention time of a string $\mathtt{S} = \mathtt{a_1} \dots \mathtt{a_n}$ (Figure 1). The first model is a simple additive model with one coefficient for each character in Σ assuming the retention time of a string mainly depends on the composition of its characters [9]. The second model additionally considers the position of the characters at the beginning and the end of the string [9, 8]. The last model uses coefficients for pairs of consecutive characters to model the influence of a character's direct neighborhood [8, 15].

Linear model. Every character $\mathtt{a} \in \Sigma$ has a retention time coefficient $t(\mathtt{a}) \in \mathbb{Z}$. The retention time of a string $\mathtt{S}$ is the sum of the retention time coefficient of its characters,

$$t_{\text{lin}}(\mathtt{S}) = \sum_{i=1}^{n} t(\mathtt{a_i}). \tag{1}$$

Position-dependent model. We define distinct retention time coefficients for the first γ and the last γ positions of a string, where $1 \leq \gamma \leq \lfloor \frac{n}{2} \rfloor$. The retention time coefficient of the i-th character for $i \leq \gamma$ is denoted by $t_{\text{pre}}(\mathtt{a_i}, i) \in \mathbb{Z}$ and the retention time coefficient of the $(n - j + 1)$-th character for $j \leq \gamma$ by $t_{\text{suf}}(\mathtt{a_{n-j+1}}, j) \in \mathbb{Z}$. The retention time of a string $\mathtt{S}$ is the sum of the corresponding retention time coefficients

$$t_{\text{pos}}(\mathtt{S}) = \sum_{i=1}^{\gamma} t_{\text{pre}}(\mathtt{a_i}, i) + \sum_{j=\gamma+1}^{n-\gamma} t(\mathtt{a_j}) + \sum_{k=1}^{\gamma} t_{\text{suf}}(\mathtt{a_{n-k+1}}, k). \tag{2}$$

Neighborhood-based model. We define retention time coefficients $t(\mathtt{a}, \mathtt{b}) \in \mathbb{Z}$ for pairs of consecutive characters $\mathtt{a}, \mathtt{b} \in \Sigma$. The first and the last character $\mathtt{a_1}$ and $\mathtt{a_n}$ of a string $\mathtt{S}$ have additional coefficients $t(-, \mathtt{a_1}), t(\mathtt{a_n}, -) \in \mathbb{Z}$, as these characters have only one adjacent character in $\mathtt{S}$. The retention time of $\mathtt{S}$ is the sum of all these coefficients,

$$t_{\text{nei}}(\mathtt{S}) = t(-, \mathtt{a_1}) + \left(\sum_{i=1}^{n-1} t(\mathtt{a_i}, \mathtt{a_{i+1}}) \right) + t(\mathtt{a_n}, -). \tag{3}$$

The coefficients for all three models need to be estimated based on a training dataset (Sections 4.1 and Appendix Section B) or taken from the available literature.

We recall the de novo peptide sequencing problem with respect to the symmetric difference scoring model [17]: Given a mass M and a set of fragment masses X (measured by the mass spectrometer), find a string $\mathtt{S}$ of mass M that minimizes $|\text{TS}(\mathtt{S}) \triangle X|$. Equivalently to computing a string with mass M that minimizes $|\text{TS}(\mathtt{S}) \triangle X|$, we can compute a string that maximizes $|\text{TS}(\mathtt{S}) \cap X| - |\text{TS}(\mathtt{S}) \setminus X|$, as X is a fixed input and $\mathtt{S}$ can be chosen. Throughout this paper, we assume that $0, M \in X$.

We consider a variant of this problem that also considers the measured retention time and a retention time prediction function $t_* : \Sigma^* \to \mathbb{Z}$. A function $t_*()$ can return negative values, as a substring can have a negative effect on the retention time of a string.

▶ **Problem 1** (De Novo Sequencing Problem). *Let Σ be an alphabet of characters, with a mass $m(\mathtt{a}) \in \mathbb{R}^+$ for each $\mathtt{a} \in \Sigma$. Given a peptide mass $M \in \mathbb{R}^+$, a retention time $T \in \mathbb{N}$, a tolerance parameter $\varepsilon \geq 0$ and a set $X = \{x_i \in \mathbb{R}^+ \mid i = 1, \dots, k\}$, find a string $\mathtt{S}$ of characters in Σ with $m(\mathtt{S}) = M$ and $|t(\mathtt{S}) - T| \leq \varepsilon$ that minimizes $|\mathrm{TS}(\mathtt{S}) \triangle X|$ among all strings with mass M and a retention time $t_*(\mathtt{S}) \in [T - \varepsilon, T + \varepsilon]$.*

2.1 Model Simplifications

The model used in this paper simplifies several aspects of experimental data. First, the peptide molecule contains an H_2O molecule in addition to the amino acid molecules. Therefore, the peptide mass has an offset of 18 Dalton compared to the sum of the amino acid masses. To simplify the description of the algorithms, we do not consider this offset (i.e. the mass M is the sum of only the amino acid masses) and the mass offsets of different ion types. However, we do consider both offsets in the implementation of our algorithms using techniques described in [17]. Moreover, the mass spectrometer measures masses-to-charge ratios. Charge state deconvolution [7] is required as a preparatory step to convert mass-to-charge ratios to masses if multiply charged fragments should be considered. Our model can consider fixed modifications by altering the amino acid masses and variable modifications by adding new characters to the alphabet. Finally, we consider integer values in the description of the algorithm and ignore the mass accuracy of the mass spectrometer. We discuss in the appendix how we account for the mass accuracy and also refer to [17].

3 Algorithms for De Novo Sequencing with Retention Time

We briefly describe the algorithm DeNovoΔ [17] for computing a string of mass M that minimizes $|\mathrm{TS}(\mathtt{S}) \triangle X|$ without considering retention times. We refer to [17] for a detailed description and a proof of correctness. Then, we describe algorithms based on DeNovoΔ for solving the de novo sequencing problem for each considered prediction model.

The search space of DeNovoΔ is modeled by a directed acyclic multigraph $G = (V, E)$ based on the given set X. A vertex in G represents a mass and a path in G represents a string. For every mass $m \in X$ there are two vertices m and $M - m$ in G, i.e. $V = \{m, M - m \mid m \in X\}$. An edge in G is always directed from the smaller to the larger mass. Two vertices v and w are connected by an edge if there exists a string with mass $w - v$. For each such string with mass $w - v$, we add an edge from v to w to the multigraph and label it with this string. That is, if v and w are connected by an edge with label $l(v, w)$, there is also an edge from v to w for every permutation of $l(v, w)$. In practice, we only consider edges with a maximal label length p. We denote the concatenation of the edge labels along a path P by $l(P)$.

Given a path P that starts at vertex 0, every traversed vertex represents the mass of a prefix of the string $l(P)$. If P additionally ends in vertex M, the path label both explains v and $M - v$ for every traversed vertex v. We find a string $\mathtt{S}$ of mass M that minimizes $|\mathrm{TS}(\mathtt{S}) \triangle X|$ by iteratively extending two paths both starting at vertex 0. One path represents a prefix and the other path a reversed suffix. We extend both paths until the sum of their labels' masses is equal to M and then concatenate the prefix and the reversed suffix to a string of mass M.

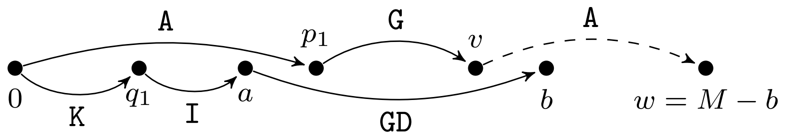


Figure 2 Multigraph G with two paths $P = (0, p_1, v)$ and $Q = (0, q_1, a, b)$. P and Q form a path pair, as there exists a sequence of balanced extensions leading to P and Q. A balanced extension of (P, Q) by (v, w) results in a path pair (P', Q), with $P' = (0, p_1, v, w)$ and $m(l(P')) + m(l(Q)) = M$. The path labels represent a prefix and a reversed suffix and can be combined to a string `AGADGIK`.

▶ **Definition 2** (Balanced extension). Given two paths P and Q, a *balanced extension* extends the path that represents the string of smaller mass by a single edge, unless the resulting paths represent strings with a total mass larger than M. An arbitrary path is extended if both paths represent strings with equal masses.

▶ **Definition 3** (Path pair). A *path pair* is a pair of paths $P = (0, \ldots, v)$ and $Q = (0, \ldots, a, b)$ in G that results from a sequence of balanced extensions starting from two paths $P_0 = (0)$ and $Q_0 = (0)$.

Figure 2 depicts an example for a path pair and a balanced extension. The set of masses that are explained by a path pair (P, Q) is the *partial theoretical spectrum*

$$\mathrm{PTS}(P, Q, M) = \{\ m(\mathtt{T}), M - m(\mathtt{T}) \mid \mathtt{T} \in \mathrm{Pre}(l(P)) \cup \mathrm{Pre}(l(Q))\ \}. \tag{4}$$

The *score* of the path pair is the number of masses explained by (P, Q) that are in X minus the number of explained masses that are not in X, i.e. $|\mathrm{PTS}(P, Q, M) \cap X| - |\mathrm{PTS}(P, Q, M) \setminus X|$. The set of masses explained by an edge (v, w) is denoted by

$$\mathrm{TSe}((v, w), M) = \{\ m(\mathtt{T}) + v,\ \ M - (m(\mathtt{T}) + v) \mid \mathtt{T} \in \mathrm{Pre}(l(v, w)),\ \ m(\mathtt{T}) \neq 0\ \}. \tag{5}$$

▶ **Lemma 4.** *For every path pair $P = (0, \ldots, v)$ and $Q = (0, \ldots, a, b)$ with $v \leq b$ and $v + b \leq M$ it holds that $a \leq v \leq b$. The balanced extension of (P, Q) by an edge (v, w) additionally explains all masses in $N((v, w), (a, b)) = \mathrm{TSe}((v, w), M) \setminus \mathrm{TSe}((a, b), M)$.*

Proof. Assume that there exists a path pair (P, Q) with $v \leq a$. This path pair results by definition from a sequence of balanced extensions. Consider the balanced extension in this sequence, where the last edge (a, b) of Q is added. In this step, either P ended in v or in some vertex $v' < v$. In both cases, a is the larger mass and Q represents the heavier string. Hence, the extension by (a, b) is not a balanced extension and (P, Q) is not a path pair.

Consider a balanced extension of (P, Q) by an edge (v, w). The edge (v, w) explains all masses in $\mathrm{TSe}((v, w), M)$. However, some of these masses might also be explained by (P, Q). We show that $\mathrm{TSe}((v, w), M) \setminus \mathrm{PTS}(P, Q, M) = N((v, w), (a, b))$, i.e. that all masses explained by (v, w) that are also explained by (P, Q), are explained by the last edge (a, b) of Q. We note that all masses in $\mathrm{TSe}((v, w), M)$ are larger than v and smaller than $M - v$. Moreover, all masses in $\mathrm{PTS}(P, Q, M)$ that are larger than v and smaller than $M - v$ are explained by the edge (a, b). Therefore, it follows that the balanced extension with (v, w) additionally explains all masses in $N((v, w), (a, b))$. ◀

Using Lemma 4, the algorithm DeNovoΔ [17] (Algorithm 1) computes a dynamic programming table DP. An entry $DP[v, (a, b)]$ contains the optimal score of a path pair ending at the vertex v, respectively at the edge (a, b). As a base case, we add a loop edge $(0, 0)$ to the

Algorithm 1 DeNovoΔ [17]

```
DP[v,(a,b)] = $-\infty$ for all (a,b)$\in E$ and all v$\in V$
DP[0,(0,0)] = 2
for (v$\in V$ in ascending order):
    for ((a,b)$\in E$ in lexicograph. asc. order with DP[v,(a,b)] $\neq -\infty$ ):
        for ((v,w)$\in E$ with w + b $\leq M$):
            if (w $\leq$ b):
                DP[w,(a,b)] = max(
                    DP[w,(a,b)], DP[v,(a,b)] + gain((v,w),(a,b))
                )
            else:
                DP[b,(v,w)] = max(
                    DP[b,(v,w)], DP[v,(a,b)] + gain((v,w),(a,b))
                )
```

graph and initialize $DP[0,(0,0)] = 2$. Given the optimal score $DP[v,(a,b)]$, the algorithm considers all possible balanced extensions of the corresponding path pair with outgoing edges of v. By Lemma 4, the additionally explained masses of such a balanced extension can be computed only given the last vertex v and the last edge (a,b) of the two paths. The score of the resulting new path pair can be computed by adding

$$\text{gain}((v,w),(a,b)) = |N((v,w),(a,b)) \cap X| - |N((v,w),(a,b)) \setminus X| \tag{6}$$

to the score $DP[v,(a,b)]$. The corresponding entry in the table is updated if the new score exceeds the value stored in this entry at this step of the algorithm. The optimal score for a string of mass M is equal to the maximum value of an entry $DP[M-b,(a,b)]$ among all edges (a,b) in G. The corresponding paths can be reconstructed starting from this entry. The combination of the resulting prefix and reversed suffix then leads to the desired string of mass M. The time complexity of DeNovoΔ is in $\mathcal{O}(|V| \cdot |E| \cdot d \cdot p)$, where d is the maximal out-degree of a vertex in G and p is the maximal length of an edge label [17].

3.1 Linear Prediction Model

In this section, we extend DeNovoΔ for the de novo sequencing problem with the linear retention time prediction model. First, we note that the retention time of a path pair $P = (0,\ldots,v)$ and $Q = (0,\ldots,a,b)$ with $a \leq v \leq b$ is the sum of the retention times of both substrings $t = t_{\text{lin}}(l(P)) + t_{\text{lin}}(l(Q))$. The retention time t' of a path pair obtained from (P,Q) by applying a balanced extension by some edge (v,w) can be computed as $t' = t + t_{\text{lin}}(l(v,w))$. That is, we only need t and the edge label $l(v,w)$ for computing t'.

However, it is not sufficient to only store the optimal score $DP[v,(a,b)]$ of any path pair ending in v, respectively (a,b), and its retention time to reconstruct a solution for our problem. There can be multiple path pairs ending in the same vertex and the same edge with different retention times. If we consider an optimal solution and its sequence of path pairs computed by the algorithm, a path pair $P = (0,\ldots,v)$ and $Q = (0,\ldots,a,b)$ in this sequence does not necessarily have an optimal score among all path pairs ending in v and (a,b). Nevertheless, its score is optimal among all path pairs with the same retention time that end in v and (a,b). Therefore, we need to store for each possible retention time t the optimal score of a path pair ending in vertex v and edge (a,b).

DeNovoΔLin (Algorithm 2) stores for each entry $DP[v,(a,b)]$ an array containing a score for every possible retention time t. $DP[v,(a,b)][t]$ is the optimal score for a path pair ending in v, respectively (a,b), with retention time t. For a given vertex v and an edge (a,b), the algorithm performs balanced extensions by all outgoing edges (v,w) of v. For every balanced extension and every feasible retention time t, the algorithm then computes the new retention time t' and the new score of the resulting path pair and updates the corresponding entry in the table. We can see by an inductive argument that the optimal scores in the table are computed correctly. As the base case, we note that $DP[0,(0,0)][0] = 2$ is correct, as an empty path pair explains the masses $\{0, M\} \subseteq X$ and has retention time 0. As soon as the entry $DP[v,(a,b)]$ is reached in line 7, all optimal scores for path pairs ending in vertex v and edge (a,b) have been computed. This holds by induction, as every possible balanced extension leading to a path pair ending in v and (a,b) has already been considered (given the optimal score of a preceding path pair). Moreover, the array in $DP[v,(a,b)]$ is not further modified as soon as the algorithm reaches the vertex v and the edge (a,b) in line 7. Therefore, the invariant holds that, if the algorithm considers a vertex v and an edge (a,b) in line 7, the corresponding entry $DP[v,(a,b)]$ contains the optimal score for each feasible retention time.

After computing all entries $DP[v,(a,b)]$, we can find the optimal score of a solution by iterating over all entries $DP[M-b,(a,b)][t]$ for $(a,b) \in E$ and all feasible retention times $t \in [T-\varepsilon, T+\varepsilon]$. We can reconstruct a corresponding string starting from this entry.

The running time of DeNovoΔ is in $\mathcal{O}(|V| \cdot |E| \cdot d \cdot p)$ [17], where d is the maximal out-degree of a vertex in G and p is the maximal length of an edge label. The additional overhead of DeNovoΔLin (highlighted lines in Algorithm 2) is to iterate over all feasible retention times t for each entry $DP[v,(a,b)]$ and compute the new retention time t'. The number of scores to be stored varies depending on the entry and the retention time coefficients. For a path pair ending in v, respectively (a,b), we have to consider all retention times in $[rt_{\min} \cdot (v+b), rt_{\max} \cdot (v+b)]$, where $rt_{\min}$ and $rt_{\max}$ are the minimum and the maximum retention time per mass unit. For example, we only store one optimal score in entry $DP[0,(0,0)]$, but up to $\lceil rt_{\max} \cdot M - rt_{\min} \cdot M \rceil$ scores in entries $DP[M-b,(a,b)]$ for $(a,b) \in E$. The time complexity of DeNovoΔLin is in $\mathcal{O}(|V| \cdot |E| \cdot |RT_M| \cdot d \cdot p)$, where $|RT_M|$ denotes the number of possible retention times for a string of mass M. In practice, most entries $DP[v,(a,b)]$ contain only few scores and it is advisable to use a memory-efficient data structure instead of an array to reduces the memory consumption of the algorithm.

3.2 Position-dependent Prediction Model

In the position-dependent prediction model, the retention time of a string $\mathtt{S}$ is not equal to the retention time of all permutations of $\mathtt{S}$. The retention time coefficient of a character in the first and the last γ positions of the string may be different from the coefficient of the same character at another position. To compute the retention time of a path pair $P = (0,\ldots,v)$ and $Q = (0,\ldots,a,b)$ with $a \le v \le b$, we first have to distinguish the prefix and the suffix path. We compute the retention time of (P,Q) by summing the retention times t_P and t_Q of the path labels. Assuming that P is the prefix path and Q the suffix path,

$$t_P = \sum_{\mathtt{a_i} \in\ l(P)} \begin{cases} t_{\text{pre}}(\mathtt{a_i}, i) & i \le \gamma \\ t(\mathtt{a_i}) & i > \gamma \end{cases} \qquad \text{and} \qquad t_Q = \sum_{\mathtt{a_j} \in\ l(Q)} \begin{cases} t_{\text{suf}}(\mathtt{a_j}, j) & j \le \gamma \\ t(\mathtt{a_j}) & j > \gamma. \end{cases} \tag{7}$$

If we want to update the retention time after a balanced extension of (P,Q) by an edge (v,w), we have to compute the retention time of the edge label $l(v,w)$. This retention time depends on whether the edge label contains some of the first or the last γ characters of a

Algorithm 2 DeNovoΔLin – Linear retention time prediction model

```
for ((a,b) $\in E$ and v $\in V$)
    DP[v,(a,b)] = array with entries $-\infty$ for each feasible ret. time
DP[0,(0,0)][0] = 2
for (v $\in V$ in ascending order):
  for ((a,b) $\in E$ in asc. lex. order with a $\leq$ v $\leq$ b):
    for ((v,w) $\in E$ with w + b $\leq M$):
      for (entry t in DP[v,(a,b)]):
        t' = t + $t_{\mathrm{lin}}(l(v,w))$
        if (w $\leq$ b):
          DP[w,(a,b)][t'] = max(
              DP[w,(a,b)][t'], DP[v,(a,b)][t] + gain((v,w),(a,b))
          )
        else:
          DP[b,(v,w)][t'] = max(
              DP[b,(v,w)][t'], DP[v,(a,b)][t] + gain((v,w),(a,b))
          )
```

solution string S of mass M. However, there can be multiple such solution strings resulting from different further balanced extensions of this path pair. Independently of the solution string S, we can decide whether $l(v,w)$ contains some of the first γ characters given the length k of $l(P)$. If $k \geq \gamma$, the edge label clearly does not contain any of the first γ characters of any solution resulting from extending (P,Q). Likewise, we know that $l(v,w)$ contains none of the γ last characters if $l(Q)$ has more than γ characters. However, if $l(Q)$ has less than γ characters, we cannot decide whether $l(v,w)$ contains some of the last γ characters without knowing the length of the solution string. Let us assume for now that $l(v,w)$ does not contain some of the last γ characters of the solution. The retention time of the new path pair resulting the balanced extension of (P,Q) by the edge (v,w) is

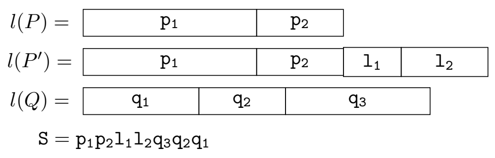
$$t' = t + \sum_{\mathtt{a_i} \in l(v,w)} \begin{cases} t_{\mathrm{pre}}(\mathtt{a_i}, i) & i + k \leq \gamma \\ t(\mathtt{a_i}) & i + k > \gamma. \end{cases} \tag{8}$$

If P would be the suffix path, $t_{\mathrm{pre}}(\mathtt{a_i}, i)$ would be replaced by $t_{\mathrm{suf}}(\mathtt{a_i}, i)$ in the above equation.

It is important that the above assumption holds for every balanced extension leading to a solution string S. Otherwise, the retention time of the new path pair is not computed correctly. We cannot check if our assumption holds for an individual balanced extension. However, given a solution string S and a path pair that represents a prefix and a suffix of S, we can check if either the balanced extension leading to this path pair or a preceding balanced extension did not satisfy the assumption. If so, either the prefix or the suffix path label has at least $n - \gamma$ characters, where n is the length of S. This does also hold for all subsequent path pairs, as we only add characters to path labels in a balanced extension.

When reconstructing a solution from the dynamic programming table, we have to additionally check, if one of the path labels has $n - \gamma$ or more characters, before we combine them to a solution string. If so, the assumption was not fulfilled at some step and we discard this solution, as its retention time was not computed correctly. Note that we cannot consider these strings, unless they can be constructed by another sequence of balanced extensions. It is very unlikely that the assumption is not fulfilled in practice, as we consider small values of γ. We never observed such a situation in our evaluation with $\gamma = 2$.

In our dynamic program, we have to store some additional information to compute a solution with respect to the position-dependent prediction model. First, we have to store



$$t = t_{\text{nei}}\,(P,Q) = t(-,\mathtt{p_1}) + t(\mathtt{p_1},\mathtt{p_2}) + t(\mathtt{q_3},\mathtt{q_2}) + t(\mathtt{q_2},\mathtt{q_1}) + t(\mathtt{q_1},-)$$

$$t' = t_{\text{nei}}\,(P',Q) = t + t(\mathtt{p_2},\mathtt{l_1}) + t(\mathtt{l_1},\mathtt{l_2})$$

$$t_{\text{nei}}(\mathtt{S}) = t_{\text{nei}}\,(P',Q) + t(\mathtt{l_2},\mathtt{q_3})$$

Figure 3 The retention time t of a path pair (P,Q) is the sum of the retention time coefficients up to the last characters $\mathtt{p_2}$ and $\mathtt{q_3}$. The path pair (P',Q) resulting from a balanced extension of (P,Q) by an edge with label $\mathtt{l_1l_2}$ has retention time $t + t(\mathtt{p_2},\mathtt{l_1}) + t(\mathtt{l_1},\mathtt{l_2})$. A path pair (P',Q) with $m(l(P')) + m(l(Q')) = M$ can be combined to a solution string $\mathtt{S}$ by concatenating $l(P')$ and the reversed string of $l(Q')$. The retention time of $\mathtt{S}$ is $t_{\text{nei}}(P',Q') + t(\mathtt{l_2},\mathtt{q_3})$.

whether P is a prefix or a suffix path. Second, we have to store the length of both path labels, unless they are larger than γ. DeNovoΔPos (Algorithm A.1 in the appendix) stores the optimal scores of path pairs ending in v and (a,b) in an array with an entry for every retention time t, the length α and β of the path labels and a Boolean flag *bit* indicating if the path ending in v is the prefix or the suffix path. Given the optimal score of a path pair, the algorithm performs every possible balanced extension with an outgoing edge of v, computes the new score and retention time, and updates the corresponding entries. We reconstruct a solution starting from a path pair ending in some vertex $M-b$ and some edge (a,b) and the algorithm additionally verifies that both the prefix and the suffix path label have more than γ characters. DeNovoΔPos considers at most $\gamma^2 \cdot |RT_M|$ optimal scores for each table entry $DP[v,(a,b)]$, where $|RT_M|$ is the number of possible retention times for a string of mass M. Therefore, the running time is in $\mathcal{O}\left(|V| \cdot |E| \cdot |RT_M| \cdot \gamma^2 \cdot d \cdot p\right)$, where d is the maximal out-degree of a vertex in G and p is the maximal length of an edge label.

3.3 Neighborhood-based Prediction Model

The neighborhood-based model predicts the retention time of a string $\mathtt{S}$ by considering all pairs of consecutive characters. We define the retention time of a prefix of $\mathtt{S}$ as the sum of the retention time coefficients of the pairs of consecutive characters and the additional coefficient of the first character. Note that we consider only one coefficient for the last character in the prefix. The other coefficient depends on the next character in $\mathtt{S}$ that is not part of the prefix. We define the retention time of a suffix analogously and compute the retention time of (P,Q) by summing the retention times of the path labels (Figure 3): We denote the two substrings by $l(P) = \mathtt{p_1},\ldots,\mathtt{p_n}$ and $l(Q) = \mathtt{q_1},\ldots,\mathtt{q_m}$. The retention time of (P,Q) is

$$t_{\text{nei}}(P,Q) = t(-,\mathtt{p_1}) + \left(\sum_{i=1}^{n-1} t(\mathtt{p_i},\mathtt{p_{i+1}})\right) + \left(\sum_{i=m}^{2} t(\mathtt{q_i},\mathtt{q_{i-1}})\right) + t(\mathtt{q_1},-). \tag{9}$$

We can update the retention time after a balanced extensions of (P,Q) as follows. Consider a balanced extension of the prefix path P by an edge (v,w) with $l(v,w) = \mathtt{l_1}\ldots\mathtt{l_k}$. Let $\mathtt{p_n}$ be the last character of $l(P)$. The retention time t' of the new path pair resulting from the balanced extension is $t' = t_{\text{nei}}(P,Q) + t(\mathtt{p_n},\mathtt{l_1}) + \sum_{i=1}^{k-1} t(\mathtt{l_i},\mathtt{l_{i+1}})$. The retention time of a solution $\mathtt{S}$ is not the sum of the retention times of a prefix of $\mathtt{S}$ and its complementary suffix. We have to additionally consider the coefficient of the last character of the prefix and the first character of the suffix, which are consecutive in $\mathtt{S}$. If we combine the path labels of a path pair (P',Q) to a string $\mathtt{S}$ (Figure 3), the retention time of $\mathtt{S}$ is $t_{\text{nei}}(P,Q) + t(\mathtt{p_n},\mathtt{q_m})$, where $\mathtt{p_n}$ and $\mathtt{q_m}$ are the last characters of P and Q.

DeNovoΔNei (Algorithm A.2 in the appendix) extends the algorithm DeNovoΔ and computes a solution with respect to the neighborhood-based prediction model as follows. Instead of storing the optimal score $DP[v,(a,b)]$ of a path pair ending in vertex v and edge (a,b), we distinguish prefix and suffix path and store an optimal score for each retention time t, last character $\mathtt{p}$ of the path ending in v. The algorithm considers at most $|\Sigma| \cdot |RT_M|$ optimal scores for each pair of a vertex v and an edge (a,b), where $|RT_M|$ is the number of possible retention times for a string of mass M and $|\Sigma|$ is the size of the considered alphabet. The running time of DeNovoΔNei is in $\mathcal{O}(|V| \cdot |E| \cdot |RT_M| \cdot |\Sigma| \cdot d \cdot p)$. We refer to the appendix for a more detailed description of the algorithm.

4 Experimental Evaluation and Discussion

In this section, we study the performance of our algorithms for de novo peptide sequencing with retention time prediction. We first describe the considered dataset and a method for estimating the parameters of the three models. Then, we compare the identification rates of the proposed algorithms to the identification rate of DeNovoΔ [17].

4.1 Dataset and Parameter Estimation

We use the SWATH-MS Gold Standard (SGS) dataset (`peptideatlas.org`, identifier PASS00289, [13]). Specifically, we consider the 944 spectra of synthesized peptides from DDA-experiments that have also been considered in [17]. The database search tool Comet [3] identified a sequence for each of these spectra using the very restricted database containing only the 422 synthesized peptides (see [17] for a detailed explanation). We randomly split the dataset into a training set with 80% of the spectra (755 spectra) and a test set with the remaining 20% of the spectra (189 spectra). We use the training set to estimate the retention time coefficients by linear regression and choose the tolerance parameter ε for each model using the test set. We choose the tolerance parameter ε based on the minimum and maximum prediction error (Appendix B) and set $\varepsilon = 1000$ (in seconds) for the linear prediction model and $\varepsilon = 750$ for the position-dependent model. The neighborhood-based prediction model requires many retention time coefficients for each character. Due to the small training dataset, the estimate of some coefficients is based on few observations and some cannot be estimated and are set to 0. A much larger training dataset would be necessary to train this model. We analyze all spectra for the linear and position-dependent model, but limit our evaluation of the neighborhood-based prediction model to some exemplary spectra. It is also be possible to use retention time coefficients reported in the literature (e.g. [9] and references therein), if training data is not available.

4.2 Comparison of DeNovoΔLin and DeNovoΔPos

We analyzed the 944 considered spectra with DeNovoΔLin and DeNovoΔPos. Both algorithms compute all solutions with a score of at least 90% of the optimal score and a predicted retention time within the tolerance range. Figure 4 shows a comparison of the identification rates of DeNovoΔ [17], DeNovoΔLin, and DeNovoΔPos. Without considering the retention time, DeNovoΔ reported the annotated sequence as best-scoring sequence for 586 spectra (62.1%). Considering the linear retention time prediction model, DeNovoΔLin computed the annotated sequence with an optimal score for 610 spectra (64.6%). DeNovoΔPos considers the position-dependent prediction model and achieved the highest identification rate. The annotated sequence was reported as best-scoring sequence for 629 spectra (66.6%). A filtering

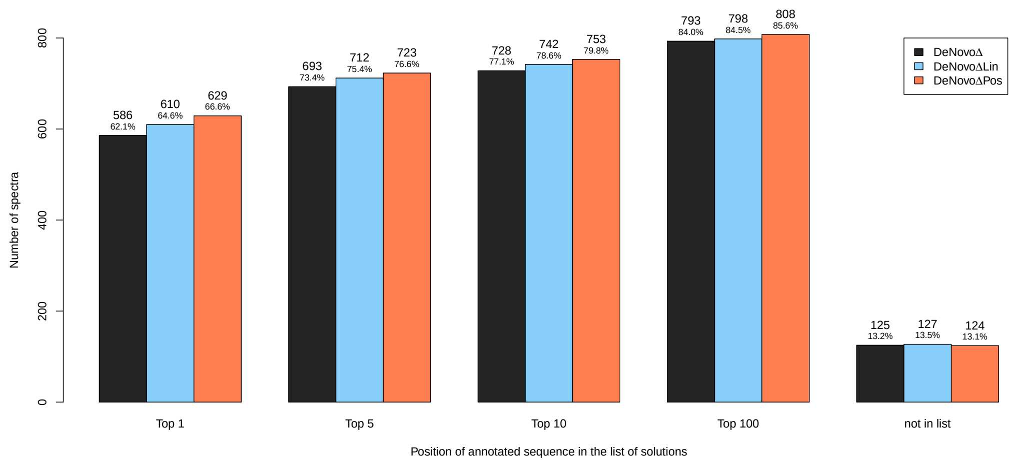


Figure 4 Position of annotated sequence in the list of reported sequences (sorted by score). DeNovoΔ reported the annotated sequence among the top 5 sequences in 73.4% of the spectra, DeNovoΔLin in 75.4% and DeNovoΔPos in 76.6% of the spectra.

approach that considers the top 100 sequences reported by DeNovoΔ, would not be as successful as the proposed algorithms. While the annotated sequence was reported by DeNovoΔ for 793 spectra among the top 100 sequences, DeNovoΔLin reported it in 798 cases and DeNovoΔPos in 808 cases. Even an optimal filtering approach by retention time would miss the sequences that have not been reported by DeNovoΔ. For few spectra, DeNovoΔLin and DeNovoΔPos did not report the annotated sequence, where DeNovoΔ did report it, as the predicted retention time of the annotated sequence was not in the chosen tolerance range.

4.3 Discussion

We develop algorithms for three additive retention time prediction models. However, we did not study the predictive robustness of our models and efficient methods for parameter estimation in detail. In the experimental evaluation, we especially studied the effect of considering the retention time information. We compare the performance of our algorithms to the algorithm DeNovoΔ [17] that uses the same scoring model, but no retention time information. An accurate retention time prediction model is crucial for exploiting the retention time information successfully, as the identification rates of our algorithms depend on the choice of the tolerance parameter ε. Increasing ε diminishes the effect of considering the retention time, while decreasing ε might exclude the correct sequence from the search space. This is especially an issue if the prediction model is not accurate, as for the neighborhood-based retention time model with our small training dataset. To get a glimpse on the performance of DeNovoΔNei, we set $\varepsilon = 500$ (in seconds) and analyzed the spectra from the test set, where the correct sequence was not excluded due to the predictive error. In three cases, the annotated sequence was reported by DeNovoΔNei, but by no other considered algorithm. The position of the annotated sequence improved compared to the position reported by DeNovoΔPos for 12 spectra.

The running time of our prototypical implementations is in some cases not yet practical. DeNovoΔLin needs less than 3 seconds per spectra for half of the considered spectra, but several hours in exceptional cases. In general, DeNovoΔPos is more time-consuming. Half of the spectra were analyzed within about 2 minutes. However, we note that we did not optimize our implementations for speed and memory usage.

5 Conclusion

In this paper, we propose the first algorithms for exploiting the retention time information in de novo peptide sequencing. We study three retention time prediction models and develop algorithms for computing a sequence that matches the experimental mass spectrum as well as possible and is in accordance with the observed retention time. The experimental evaluation of our algorithms shows that identification rates can definitively be improved by exploiting this additional information. Yet, the proposed algorithms score sequences with a very simplistic scoring function that only counts explained and measured masses, but does not consider any other available information. For real-world applications, a more evolved scoring function using all available information needs to be integrated. While [17] introduces a new scoring model, we explore ways of exploiting the retention time information. The proposed algorithms open room for developing new scoring functions that consider both the retention time information and the symmetric difference scoring model.

Acknowledgments. We would like to thank Peter Widmayer, Christian Panse, Witold Wolski, and Ludovic Gillet for helpful discussions. Moreover, we thank the reviewers for their constructive criticism.

References

1 Ting Chen, Ming-Yang Kao, Matthew Tepel, John Rush, and George M. Church. A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 8(3):325–337, 2001. doi:10.1089/10665270152530872.

2 Vlado Dančík, Theresa A. Addona, Karl R. Clauser, James E. Vath, and Pavel A. Pevzner. De novo peptide sequencing via tandem mass spectrometry. *Journal of Computational Biology*, 6(3-4):327–342, 1999. doi:10.1089/106652799318300.

3 Jimmy K. Eng, Tahmina A. Jahan, and Michael R. Hoopmann. Comet: an open-source MS/MS sequence database search tool. *Proteomics*, 13(1):22–24, 2013. doi:10.1002/pmic.201200439.

4 Ludovic Gillet, Simon Rösch, Thomas Tschager, and Peter Widmayer. A better scoring model for de novo peptide sequencing: The symmetric difference between explained and measured masses. In *16th International Workshop on Algorithms in Bioinformatics, WABI 2016*, volume 9838, pages 185–196, 2016. (extended version: [17]). doi:10.1007/978-3-319-43681-4.

5 Christopher Hughes, Bin Ma, and Gilles A. Lajoie. De novo sequencing methods in proteomics. *Proteome Bioinformatics*, 604:105–121, 2010. doi:10.1007/978-1-60761-444-9_8.

6 Kyowon Jeong, Sangtae Kim, and Pavel A. Pevzner. UniNovo: a universal tool for de novo peptide sequencing. *Bioinformatics (Oxford, England)*, 29(16):1953–1962, 2013. doi:10.1093/bioinformatics/btt338.

7 Michael Kinter and Nicholas E. Sherman. *Protein Sequencing and Identification Using Tandem Mass Spectrometry*. Wiley-Interscience, New York, 2000. doi:10.1002/0471721980.

8 Oleg V. Krokhin. Sequence-specific retention calculator. Algorithm for peptide retention prediction in ion-pair RP-HPLC: Application to 300- and 100-A pore size C18 sorbents. *Analytical chemistry*, 78(22):7785–95, 2006. doi:10.1021/ac060777w.

9 Oleg. V Krokhin, Robertson Craig, Vic Spicer, Werner Ens, Kenneth G. Standing, Ronald C. Beavis, and John A. Wilkins. An improved model for prediction of retention times of tryptic peptides in ion pair reversed-phase HPLC: its application to protein peptide mapping by off-line HPLC-MALDI MS. *Molecular & cellular proteomics : MCP*, 3(9):908–19, 2004. doi:10.1074/mcp.M400031-MCP200.

10 Bin Ma. Novor: Real-time peptide de novo sequencing software. *Journal of The American Society for Mass Spectrometry*, 26(11):1885–1894, 2015. `doi:10.1007/s13361-015-1204-0`.

11 Luminita Moruz and Lukas Käll. Peptide retention time prediction. *Mass spectrometry reviews*, 2016. `doi:10.1002/mas.21488`.

12 Magnus Palmblad, Margareta Ramström, Karin E. Markides, Per Håkansson, and Jonas Bergquist. Prediction of chromatographic retention and protein identification in liquid chromatography/mass spectrometry. *Analytical Chemistry*, 74(22):5826–5830, 2002. `doi:10.1021/ac0256890`.

13 Hannes L Röst, George Rosenberger, Pedro Navarro, Ludovic Gillet, Saša M. Miladinović, Olga T. Schubert, Witold Wolski, Ben C Collins, Johan Malmström, Lars Malmström, and Ruedi Aebersold. OpenSWATH enables automated, targeted analysis of data-independent acquisition MS data. *Nature biotechnology*, 32(3):219–223, 2014. `doi:10.1038/nbt.2841`.

14 Kosaku Shinoda, Masahiro Sugimoto, Masaru Tomita, and Yasushi Ishihama. Informatics for peptide retention properties in proteomic LC-MS. *Proteomics*, 8(4):787–98, 2008. `doi:10.1002/pmic.200700692`.

15 Vic Spicer, Marine Grigoryan, Alexander Gotfrid, Kenneth G. Standing, and Oleg V. Krokhin. Predicting retention time shifts associated with variation of the gradient slope in peptide RP-HPLC. *Analytical chemistry*, 82(23):9678–85, 2010. `doi:10.1021/ac102228a`.

16 Eric F. Strittmatter, Lars J. Kangas, Konstantinos Petritis, Heather M. Mottaz, Gordon A. Anderson, Yufeng Shen, Jon M. Jacobs, David G. Camp, and Richard D. Smith. Application of peptide LC retention time information in a discriminant function for peptide identification by tandem mass spectrometry. *Journal of Proteome Research*, 3(4):760–769, 2004. `doi:10.1021/pr049965y`.

17 Thomas Tschager, Simon Rösch, Ludovic Gillet, and Peter Widmayer. A better scoring model for de novo peptide sequencing: The symmetric difference between explained and measured masses. *Algorithms for Molecular Biology*, 12(1), 2017. (extended version of [4]). `doi:10.1186/s13015-017-0104-1`.

18 Susan K. Van Riper, Ebbing P. de Jong, John V. Carlis, and Timothy J Griffin. Mass spectrometry-based proteomics: Basic principles and emerging technologies and directions. *Advances in experimental medicine and biology*, 990:1–35, 2013. `doi:10.1007/978-94-007-5896-4_1`.

A Pseudocode of DeNovoΔPos and DeNovoΔNei

DeNovoΔPos (Algorithm A.1) computes the optimal score for a path pair with retention time t, a prefix path with label length α ending in vertex v and a suffix path with label length β ending in edge (a, b). The algorithm distinguishes prefix and suffix path, as the retention time of a string is different to the retention time of its reversed string. We store the length of the path labels only up to length γ, as the exact length is only important as long as the path labels have less than γ characters. If the algorithm reaches an entry $DP[v,(a, b)]$ in line 7, all optimal scores for path pairs ending in vertex v and edge (a, b) have been computed correctly, as all balanced extensions leading to such path pairs have already been considered. The algorithm then considers all balanced extensions by outgoing edges of v and computes the score and the retention time of the resulting path pairs. After computing all entries of the table, the solution can be reconstructed starting from an entry in some array $DP[M - b,(a, b)]$ for $(a, b) \in E$ with optimal score and a feasible retention time. The algorithm additionally has to check if the prefix and the suffix path label of the reconstructed solution have more than γ characters.

DeNovoΔNei (Algorithm A.2) computes and stores the optimal score for a path pair that ends in a given vertex v and a given edge (a, b) with a retention time t, where the path ending in v is a prefix (suffix) and `p` is the last character of the corresponding path label. As

Algorithm A.1 DeNovoΔPos – Position-dependent retention time prediction model

```
for ((a,b) $\in E$ and v $\in V$):
    DP[v,(a,b)] = ($|RT_M| \times \gamma \times \gamma \times 2$)-array initialized with $-\infty$
DP[0,(0,0)][0,0,0,0] = 2
for (v $\in V$ in ascending order):
    for ((a,b) $\in E$ asc. lex. order with a $\leq$ v $\leq$ b):
        for (entry (t,$\alpha$,$\beta$,$bit$) in DP[v,(a,b)]):
            for ((v,w) $\in E$ with w + b $\leq M$):
                t' = retention time of resulting path pair
                if ($bit$ == 1):
                    $\alpha'$ = max($\gamma$, $\alpha + |l(v,w)|$); $\beta'$ = $\beta$
                else:
                    $\alpha'$ = $\alpha$; $\beta'$ = max($\gamma$, $\beta + |l(v,w)|$)

                if (w $\leq$ b):
                    DP[w,(a,b)][t',$\alpha'$,$\beta'$,$bit$] = max(
                        DP[w,(a,b)][t',$\alpha'$,$\beta'$,$bit$],
                        DP[v,(a,b)][t,$\alpha$,$\beta$,$bit$] + gain((v,w),(a,b))
                    )
                else:
                    DP[b,(v,w)][t',$\alpha'$,$\beta'$,$\neg bit$] = max(
                        DP[b,(v,w)][t',$\alpha'$,$\beta'$,$\neg bit$],
                        DP[v,(a,b)][t,$\alpha$,$\beta$,$bit$] + gain((v,w),(a,b))
                    )
```

a base case, the algorithm computes the optimal score for a path pair ending in vertex 0 and the loop edge $(0,0)$ as $DP[0,(0,0)][0,-,0]$. Note that there exists only one such path pair with retention time $t = 0$ and we define the last character as $-$. The algorithm considers the vertices and edges of G in ascending order. Whenever the algorithm reaches a vertex v and an edge (a,b), it has already computed the optimal score of path pairs ending in this vertex and this edge for any combination of retention time t, last character p. Given these scores, the algorithm considers all possible balanced extensions by outgoing edges of v and computes the score and the retention time of the resulting path pair. The algorithm updates the corresponding entries in the table and continues with the next pair of endpoints of a path pair. Finally, the optimal score can be computed by iterating over all entries $DP[M-b,(a,b)]$ and considering the feasible retention time interval and all possible last characters of the path ending in $M-b$. In contrast to DeNovoΔ, which has a running time in $\mathcal{O}(|V| \cdot |E| \cdot d \cdot p)$, where d is the maximal out-degree of a vertex in G and p is the maximal length of an edge label, DeNovoΔNei has to consider for each pair v and (a,b) all possible retention times and all possible last characters. That is, the algorithm considers for each pair v and (a,b) at most $|RT_M| \cdot |\Sigma|$ optimal scores and performs for each optimal score all possible balanced extensions. Therefore, the running DeNovoΔNei is in $\mathcal{O}(|V| \cdot |E| \cdot |RT_M| \cdot |\Sigma| \cdot d \cdot p)$, where $|RT_M|$ is the number of feasible retention times for a string of mass M and $|\Sigma|$ is the size of the alphabet.

B Parameter Estimation

In this work, we are mainly interested in the algorithmic problem of using retention time information for de novo sequencing and do not focus on efficient procedures for estimating

Algorithm A.2 DeNovoΔNei – Neighborhood-based retention time prediction model

```
for ((a,b) $\in E$ and v $\in V$):
  DP[v,(a,b)] = $(|RT_M| \times |\Sigma| \times 2)$-array initialized with $-\infty$
DP[0,(0,0)][0,-,0] = 2
for (v $\in V$ in ascending order):
  for ((a,b) $\in E$ in ascending order of a and b):
    for (entry (t,p,$bit$,$score$) in DP[v,(a,b)]):
      for ((v,w) $\in E$ with w + b $\leq M$):
        t' = retention time of resulting path pair

      if (w $\leq$ b):
        p' = last character of $l(v,w)$
        DP[w,(a,b)][t',p',$bit$] = max(
          DP[w,(a,b)][t',p',$bit$],
          DP[v,(a,b)][t,p,$bit$] + gain((v,w),(a,b))
        )
      else:
        p' = last character of $l(a,b)$
        DP[b,(v,w)][t',p',$\neg bit$] = max(
          DP[b,(v,w)][t',p',$\neg bit$],
          DP[v,(a,b)][t,p,$bit$] + gain((v,w),(a,b))
        )
```

the coefficients of our models. We use linear regression for estimating the coefficients for our three retention time models. Even with these simple models and estimation procedures, our method shows improved identification rates and an increased performance might be achieved by considering a larger dataset.

We consider 944 spectra from the SGS dataset and partition the dataset randomly into a training set containing 80% of the spectra for estimating the retention time coefficients and a test set containing the remaining 20% of the spectra for selecting the tolerance parameter ε. The retention time coefficients are estimated by linear regression. We choose the coefficients such that the sum of the squared loss $\sum_{\mathtt{S_i},T_i}(T_i - t(\mathtt{S_i}))^2$ is minimized, where T_i is the measured retention time, and $t(\mathtt{S_i})$ the predicted retention time of the annotated sequence $\mathtt{S_i}$. For example, for estimating the coefficients of the linear model, we first compute the occurrence vector for each sequence in the dataset. The occurrence vector of a sequence is a vector of length $|\Sigma|$ that indicates how often a character occurs in the sequence; e.g., the occurrence vector of the string `AGA` has value 2 at entry `A`, value 1 at entry `G` and value 0 at all other entries. Then, the retention time of a sequence `S` is the product of its occurrence vector $occ(\mathtt{S})$ and the vector of the retention time coefficients t. Standard software tools for statistical methods can be used to compute t, such that $\sum_i(T_i - t \cdot occ(\mathtt{S}))^2$ is minimized.

In order to choose the tolerance parameter ε, we analyzed the difference between the measured and the predicted retention time of the sequences in the test set. Figure B.1 shows the differences between the predicted and the measured retention times for all three models on the test dataset. Especially for the neighborhood-based prediction model, we have to predict many retention time coefficients for each character. Several coefficients are estimated based on few observations and others cannot be estimated at all. Therefore, we cannot extensively evaluate the identification rates of our algorithm with the neighborhood-based prediction model, as a much larger training dataset for estimating all parameters would be necessary. Our comparison of the identification rates regarding this prediction model is

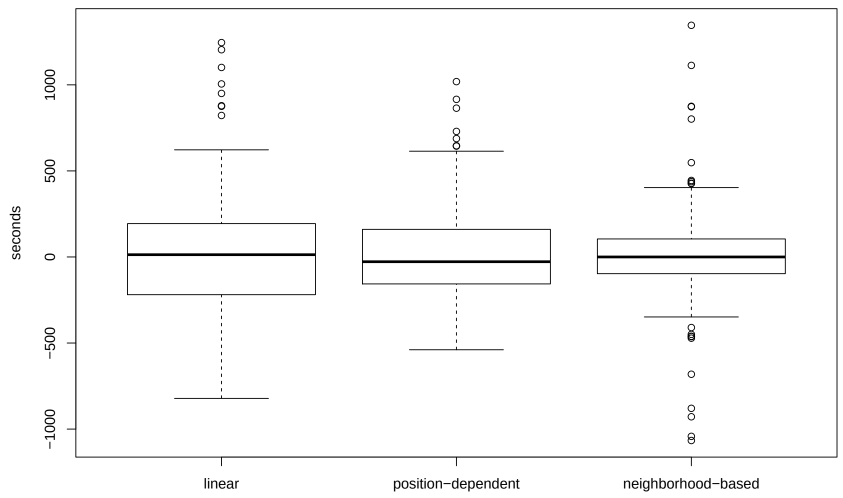


Figure B.1 Retention time prediction models – Difference between predicted and measured retention time of all sequences in the test set with respect to the three prediction model.

limited to some examples, where the prediction model works well.

We chose the tolerance parameter ε independently for each prediction model as half the difference between the maximum error $e_{\max}$ and the minimum error $e_{\min}$, i.e. $\varepsilon = (e_{\max} - e_{\min})/2$. Concretely, we set $\varepsilon = 1000$ for the linear prediction model and $\varepsilon = 750$ for the position-dependent model. The neighborhood-based prediction model has a very large predictive error for several sequences due to the small training dataset. For our limited evaluation, we ignore the 5 largest and the 5 smallest retention time errors when picking the tolerance parameter and use $\varepsilon = 500$.

C Experimental Evaluation – Supplementary Figures

In our experiments, we only considered spectra from peptides with an assumed (precursor) charge state 2 (as reported by Comet). Moreover, we assume that all measured fragment masses are singly charged, i.e. the mass-to-charge ratio is equal to the mass of a fragment. While it is possible to also consider spectra with higher charge states, the analysis of such spectra requires additional data preprocessing to convert the measured mass-to-charge ratios of the fragments to the corresponding masses (charge state deconvolution). Figure C.2 shows a distribution of the number of identified spectra with respect to the length of the corresponding peptide sequence. The position-dependent prediction model improves the identification rates on peptides with less than 15 amino acids, while the linear prediction model is favorable for longer amino acid sequences.

In the description of our algorithms, we only consider integer values and ignore the measurement accuracy. For our evaluation, we consider two masses two be equal if they differ by at most 0.02 Da. Moreover, as described in [17], we use a simple merging algorithm to reduce the size of the graph. We observed a great variation of spectrum graph sizes in our experiments. The spectrum graphs contained roughly 8400 edges on average, whereas the largest observed graph contained 23000 edges. Spectra measured on low resolution lead to denser spectrum graph, i.e. to a larger number of edges, but a lower number of vertices. However, we did not study the performance and runtime of our algorithms on this type of spectra.

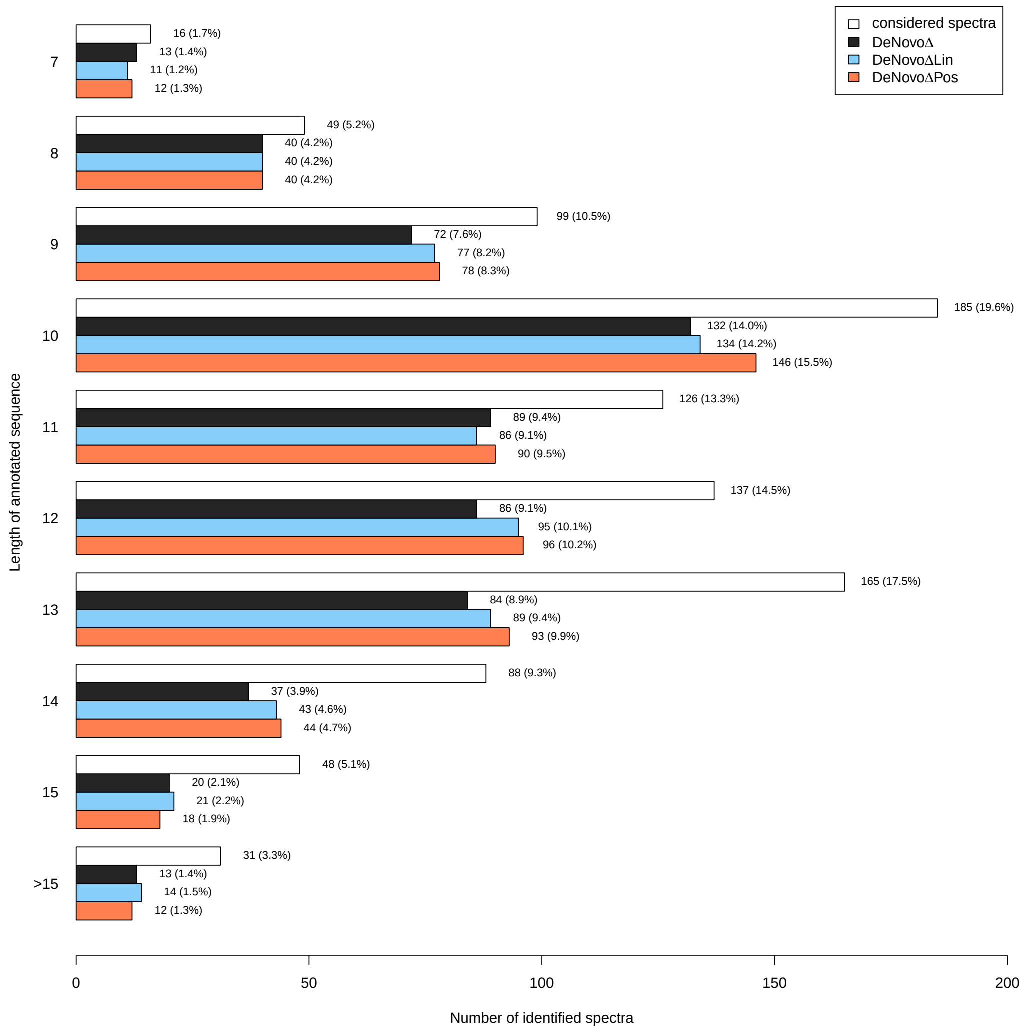


Figure C.2 Identified spectra with respect to the length of the annotated sequence.

Optimal Completion of Incomplete Gene Trees in Polynomial Time Using OCTAL[*]

Sarah Christensen[1], Erin K. Molloy[2], Pranjal Vachaspati[3], and Tandy Warnow[4]

1 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`sac2@illinois.edu`
2 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`emolloy2@illinois.edu`
3 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`vachasp2@illinois.edu`
4 **University of Illinois at Urbana-Champaign, Urbana, IL, USA**
`warnow@illinois.edu`

Abstract

Here we introduce the *Optimal Tree Completion Problem*, a general optimization problem that involves completing an unrooted binary tree (i.e., adding missing leaves) so as to minimize its distance from a reference tree on a superset of the leaves. More formally, given a pair of unrooted binary trees (T, t) where T has leaf set S and t has leaf set $R \subseteq S$, we wish to add all the leaves from $S \setminus R$ to t so as to produce a new tree t' on leaf set S that has the minimum distance to T. We show that when the distance is defined by the Robinson-Foulds (RF) distance, an optimal solution can be found in polynomial time. We also present *OCTAL*, an algorithm that solves this RF Optimal Tree Completion Problem exactly in $O(|S|^2)$ time. We report on a simulation study where we complete estimated gene trees using a reference tree that is based on a species tree estimated from a multi-locus dataset. OCTAL produces completed gene trees that are closer to the true gene trees than an existing heuristic approach, but the accuracy of the completed gene trees computed by OCTAL depends on how topologically similar the estimated species tree is to the true gene tree. Hence, under conditions with relatively low gene tree heterogeneity, OCTAL can be used to provide highly accurate completions of estimated gene trees. We close with a discussion of future research.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory, J.3 Life and Medical Sciences

Keywords and phrases phylogenomics, missing data, coalescent-based species tree estimation, gene trees

Digital Object Identifier 10.4230/LIPIcs.WABI.2017.27

1 Introduction

Species tree estimation from multi-gene datasets is now increasingly common. One challenge is that the evolutionary history for a single locus (called a “gene tree”) may differ from the species phylogeny due to a variety of different biological processes. Some of these processes,

* This work was partially supported by US National Science Foundation Graduate Research Fellowship Program under Grant Number DGE-1144245 to PV and EKM, US National Science Foundation grant CCF-1535977 to TW, and by the University of Illinois.

such as hybridization and horizontal gene transfer, result in non-treelike evolution and so require phylogenetic networks for proper analysis. However, other biological processes, such as gene duplication and loss, incomplete lineage sorting (ILS), and gene flow, produce heterogeneity across the genome but are still properly modeled by a single species tree [8]. In the latter case, species tree estimation methods must account for discordance or heterogeneity across the genome.

Much of the recent focus in the mathematical and statistical phylogenetics literature has been on developing methods for species tree estimation in the presence of incomplete lineage sorting (ILS), which is modelled by the multi-species coalescent (MSC) model [15]. One popular approach for estimating species trees under the MSC model is to estimate trees on individual loci and then combine these gene trees into a species tree. Some of these “summary methods”, such as ASTRAL-II and ASTRID, have been shown to scale well to datasets with many taxa (i.e., >100 species) and provide accurate species tree estimates [12, 20].

A common challenge to species tree estimation methods is that sequence data may not be available for all genes and species of interest, creating conditions with missing data (see discussion in [6, 18]). For example, gene trees can be missing species simply because some species do not contain a copy of a particular gene; in some cases, no common gene will be shared by every species in the set of taxa [7]. Additionally, not all genomes may be fully sequenced and assembled, as this can be operationally difficult and expensive [3, 18].

Although summary methods are statistically consistent under the MSC model [1], the proofs of statistical consistency assume that all gene trees are complete, and so may not apply when the gene trees are missing taxa. Furthermore, multi-gene datasets with missing data can be “phylogenetically indecisive”, meaning more than one tree topology can be optimal [16]. Because of concerns that missing data may reduce accuracy in multi-locus species tree estimation, many phylogenomic studies have restricted their analyses to only include genes with most of the species (see discussion by [6, 13, 18]).

Another way in which missing data impacts phylogenetics is that it is not that obvious how to evaluate the topological similarity between two trees when they are on different sets of species. A common approach is to constrain the two trees to the shared species and then compute the topological distance between the induced subtrees. However, it may be more interesting to ask how close the two trees could be if they were both completed (via the addition of the missing species) so that they are on the same species set.

Therefore, we formulate a class of optimization problems we refer to as Optimal Tree Completion problems, where we seek to add missing species to a tree to minimize the distance (defined in some way) to another tree. A natural version of this problem uses the Robinson-Foulds (RF) [14] distance between two trees, where the RF distance is the total number of unique bipartitions in the two trees. In Section 2, we formalize the RF Optimal Tree Completion problem. The Optimal Completion of incomplete gene Tree ALgorithm (or OCTAL) is a simple algorithm that incrementally adds the missing species one at a time into the tree, and which we prove solves the RF Optimal Tree Completion problem exactly. In Section 3, we present results from a experimental study on simulated datasets comparing OCTAL to a heuristic for tree completion within ASTRAL-II. Finally, we conclude with a discussion of results and future research in Section 6.

2 Optimal Tree Completion

2.1 Terminology

Each edge e in an unrooted phylogenetic tree defines a bipartition π_e on the leaves of the tree induced by the deletion of e (but not its endpoints). Each bipartition is thus a split $A|B$ of the leaf set into two non-empty parts, A and B. The set of bipartitions of a tree T is given by $C(T) = \{\pi_e : e \in E(T)\}$, where $E(T)$ is the set of edges for tree T. When two trees T and T' have the same leaf set, then the *Robinson-Foulds* (RF) distance [14] between T and T', denoted by $\text{RF}(T, T')$, is $|C(T)\Delta C(T')|$. Thus, every bipartition in T or T' is either shared between the two trees or is unique to one tree, and the RF distance counts just the unique bipartitions. When two trees are binary and on the same leaf set, as is the case in this study, the numbers of bipartitions that are unique in each tree are equal, and each is half the RF distance.

Given tree T on leaf set S, T *restricted* to $R \subseteq S$, denoted by $T|_R$, is the minimal subgraph of T that connects all elements of R, suppressing nodes of degree two. Note that if T contains the bipartition $A|B$, $T|_R$ contains the restricted bipartition $(A \cap R)|(B \cap R)$. If T and T' are two trees with R as the intersection of their leaf sets, their *shared edges* are edges whose bipartitions restricted to R are in the set $C(T|_R) \cap C(T'|_R)$. Correspondingly, their *unique edges* are edges whose bipartitions restricted to R are not in the set $C(T|_R) \cap C(T'|_R)$.

2.2 The RF Optimal Tree Completion problem

The problem we address in this paper is the **RF Optimal Tree Completion Problem**, where the distance between trees is defined by the Robinson-Foulds distance, as follows:

- Input: An unrooted binary tree T on the full taxon set S and an unrooted binary tree t on a subset of taxa $R \subseteq S$
- Output: An unrooted binary tree T' on the full taxon set S with two key properties:
  1. T' is a *S-completion* of t (i.e., T' contains all the leaves of S and $T'|_R = t$) and
  2. T' minimizes the RF distance to T among all *S-completions* of t

Note that t and $T|_R$ are both on taxon set R, but need not be identical. In fact, the RF distance between these two trees is a lower bound on the RF distance between T and T'.

2.3 OCTAL: Optimal Completion of incomplete gene trees ALgorithm

The algorithm begins with input tree t and adds leaves one at a time from the set $S \setminus R$ until a tree on the full set of taxa S is computed. To add the first leaf, we choose an arbitrary taxon x to add from the set $S \setminus R$. We root the tree $T|_{R\cup\{x\}}$ (i.e., T restricted to the leaf set of t plus the new leaf being added) at x, and then remove x and the incident edge; this produces a rooted binary tree we will refer to as $T^{(x)}$ that has leaf set R.

We perform a depth-first traversal down $T^{(x)}$ until a shared edge e (i.e., an edge where the clade below it appears in tree t) is found. Since every edge incident with a leaf in $T^{(x)}$ is a shared edge, every path from the root of $T^{(x)}$ to a leaf has a distinct first edge e that is a shared edge. Hence, the other edges on the path from the root to e are unique edges.

After we identify the shared edge e in $T^{(x)}$, we identify the edge e' in t defining the same bipartition, and we add a new node $v(e')$ into t so that we subdivide e'. We then make x adjacent to $v(e')$. Note that since t is binary, the modification t' of t that is produced by adding x is also binary and that $t'|_R = t$. These steps are then repeated until all leaves from $S \setminus R$ are added to t. This process is shown in Fig. 1 and given in pseudocode below.

Algorithm 1: RF Optimal Tree Completion Algorithm (OCTAL)

```
 1: procedure AddLeaf(Taxon $x$, binary tree $T_1$ on taxon set $K$, binary tree $T_2$ on taxon
    set $K \cup \{x\}$, set $E$ of shared edges between $T_1$ and $T_2|_K$)
 2:     Root $T_2$ at $v$, the neighbor of $x$, and delete $x$ to produce a rooted version of $T_2|_K$
 3:     Pick an arbitrary leaf $y$ in $T_2|_K$ and find first edge $e \in E$ on path from $v$ to $y$
 4:     Find $e'$ in $T_1$ defining the same bipartition as $e$
 5:     Attach $x$ to $e'$ in $T_1$ by subdividing $e'$ and making $x$ adjacent to the newly created
    node; call the resulting tree $T_1'$
 6:     return $T_1'$
 7: end procedure
 1: procedure OCTAL(Binary tree $t$ on taxon set $R \subseteq S$, Binary tree $T$ on taxon set $S$)
 2:     if R=S then
 3:         return t
 4:     else
 5:         $E \leftarrow$ Preprocess and initialize set of shared edges between $t$ and $T|_R$
 6:         $R' \leftarrow R$                      ▷ Initialize $R'$ by setting it equal to input $R$
 7:         $t' \leftarrow t$                       ▷ Initialize $t'$ by setting it equal to input $t$
 8:         for $x \in S \setminus R$ do
 9:             $R' \leftarrow R' \cup \{x\}$
10:             $T' \leftarrow T|_{R'}$
11:             $t' \leftarrow$ AddLeaf($x$, $t'$, $T'$, $E$)
12:             $E \leftarrow$ Update shared edges between $t'$ and $T'$
13:         end for
14:         return $t'$
15:     end if
16: end procedure
```

We begin by preprocessing the two trees to identify shared edges; this takes $O(|S|^2)$ time. After this preprocessing is done, it is easy to see that *AddLeaf* takes $O(|S|)$ time to add a single taxon to t. Hence, OCTAL runs in $O(|S|^2)$ time, since there are $O(|S|)$ leaves to add.

2.4 Proof of Correctness

In what follows, let T be an arbitrary binary tree on taxon set S and t be an arbitrary binary tree on taxon set $R \subseteq S$. Let T' denote the tree returned by OCTAL given T and t. We set $r = RF(T|_R, t)$. As we have noted, OCTAL returns a binary tree T' that is an S-completion of t. Hence, to prove that OCTAL solves the RF Optimal Tree Completion problem exactly, we only need to establish that $RF(T, T')$ is the smallest possible of all binary trees on leaf set S that are S-completions of t. While the algorithm works by adding a single leaf at a time, we use two types of subtrees, denoted as *superleaves*, to aid in the proof of correctness.

▶ **Definition 1.** We define the superleaves of T with respect to t as follow (see Figure 2). The set of edges in T that are on a path between two leaves in R define the *backbone*; if this backbone is removed, the remainder of T breaks into pieces. The components of this graph that contain vertices from $S \setminus R$ are the **superleaves**. Each superleaf X is rooted at the node that was incident to one of the edges in the backbone, and is one of two types:

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

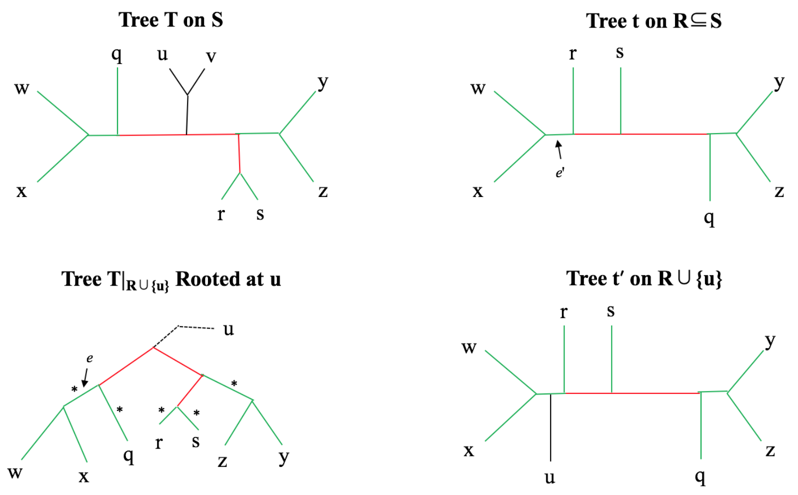


Figure 1 Trees T and t with edges in the backbone (defined to be the edges on paths between nodes in the common leaf set) colored green for shared, and red for unique; all other edges are colored black. After rooting $T|_R$ with respect to u, the edges in $T|_R$ that could be identified by the algorithm for "placement" are indicated with an asterisk (*). Note that any path in $T|_R$ from the root to a leaf will encounter a shared edge, since the edges incident with leaves are always shared. In this scenario, the edge e above the least common ancestor of leaves w and x is selected; this edge defines the same bipartition as edge e' in t. Hence, *AddLeaf* will insert leaf u into t by subdividing edge e', and making u adjacent to the newly added node.

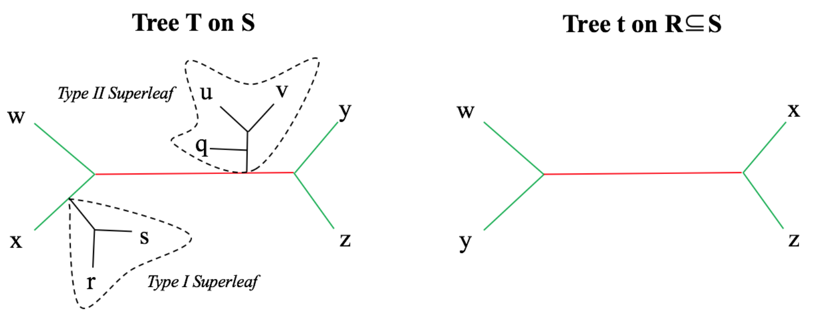


Figure 2 Trees T and t with edges in the backbone (defined to be the edges on paths between nodes in the common leaf set) colored green for shared, and red for unique; all other edges are colored black. The deletion of the backbone edges in T defines the superleaves; one is a Type I superleaf because it is attached to a shared (green) edge and the other is a Type II superleaf because it is attached to a unique (red) edge. The RF distance between t and $T|_R$ is equal to 2, the number of red edges. The Type I superleaf containing leaves r and s can be added to the shared edge incident to leaf x in tree t without increasing the RF distance. However, notice that adding the Type II superleaf to any edge in t creates at least one new unique edge in each tree and therefore increases the RF distance by at least 2, independent of how r and s are placed in t. This example motivates a more general result about elements of Type I and Type II superleaves proved in Section 2.4.

Observe that these intuitive definitions are equivalent to the following more formal definitions we sometimes invoke below. A superleaf X is a Type I superleaf if and only if there exists a bipartition $A|B$ in $C(t) \cap C(T|_R)$ where $A|(B \cup X)$ and $(A \cup X)|B$ are both in $C(T|_{R \cup X})$. Furthermore, a superleaf X is a Type II superleaf if and only if there does not exist such a bipartition in $C(t) \cap C(T|_R)$.

Now we begin our proof by establishing a lower bound on the RF distance to T for all binary, S-completions of t.

▶ **Lemma 2.** *Let Y be a Type II superleaf for the pair (T, t), and let $x \in S \setminus R$. Let t^* be the result of adding x into t arbitrarily (i.e., we do not attempt to minimize the resulting RF distance). If $x \notin Y$, then Y is a Type II superleaf for the pair (T, t^*). Furthermore, if $x \in Y$, then $RF(T|_{R \cup \{x\}}, t^*) \geq RF(T|_R, t) + 2$.*

Proof. It is easy to see that if $x \notin Y$, then Y remains a Type II superleaf after x is added to t. Now suppose $x \in Y$. We will show that we cannot add x into t without increasing the RF distance by at least 2. Since Y is a Type II superleaf, it is attached to a unique edge in $T|_{R \cup Y}$, and this is the same edge that x is attached to in $T|_{R \cup \{x\}}$. So suppose that x is added to t by subdividing an arbitrary edge e' in t with bipartition C|D; note that we do not require that x is added to a shared edge in t. After adding x to t we obtain tree t^* whose bipartition set includes $C|(D \cup \{x\})$ and $(C \cup \{x\})|D$. If C|D corresponds to a unique edge relative to t and $T|_R$, then both of these bipartitions correspond to unique edges relative to t^* and $T|_{R \cup \{x\}}$. If C|D corresponds to a shared edge, then at most one of the two new bipartitions can correspond to a shared edge, as otherwise we can derive that Y is a Type I superleaf. Hence, the number of unique edges in t must increase by at least one no matter how we add x to t, where x belongs to a Type II superleaf. Since t is binary, the tree that is created by adding x is binary, so that $RF(T|_{R \cup \{x\}}, t^*) \geq RF(T|_R, t) + 2$. ◀

▶ **Lemma 3.** *Let T^* be an unrooted binary tree that is a S-completion of t. Then $RF(T^*, T) \geq r + 2m$, where $r = RF(T|_R, t)$ and m is the number of Type II superleaves for the pair (T, t).*

Proof. We note that adding a leaf can never reduce the total RF distance. The proof follows from Lemma 2 by induction. ◀

Now that we have established a lower bound on the best achievable RF distance (i.e., the optimality criterion for the RF Optimal Tree Completion problem), we show OCTAL outputs a tree T' that is guaranteed to achieve this lower bound. We begin by noting that when we add x to t by subdividing some edge e', creating a new tree t', all the edges other than e' in t continue to "exist" in t' although they define new bipartitions. In addition, e' is split into two edges, which can be considered new. Thus, we can consider whether edges that are shared between t and T *remain* shared after x is added to t.

▶ **Lemma 4.** *Let t' be the tree created by AddLeaf given input tree t on leaf set R and tree T on leaf set $R \cup \{x\}$. If x is added to tree t by subdividing edge e' (thus creating tree t'), then all edges in t other than e' that are shared between t and T remain shared between t' and T.*

Proof. Let $T^{(x)}$ be the rooted tree obtained by rooting T at x and then deleting x. Let e be the edge in $T^{(x)}$ corresponding to e', and let $\pi_e = A|B$; without loss of generality assume A is a clade in $T^{(x)}$. Note that $C(T)$ contains bipartition $A|(B \cup \{x\})$ (however, $C(T)$ may not contain $(A \cup \{x\})|B$, unless e is incident with the root of $T^{(x)}$). Furthermore, for subclade $A' \subseteq A$, $A'|(R \setminus A') \in C(T|_R)$ and $A'|(R \setminus A' \cup \{x\}) \in C(T)$. Now suppose e^* in t is a shared edge between t and $T|_R$ that defines bipartition $C|D \neq A|B$. Since $A|B$ and $C|D$

are both bipartitions of t, without loss of generality either $C \subset A$ or $A \subset C$. If $C \subset A$, then C is a clade in $T^{(x)}$, and so e^* defines bipartition $C|(D \cup \{x\})$ within t'. But since $C \subset A$, the previous analysis shows that $C|(D \cup \{x\})$ is also a bipartition of T, and so e^* is shared between T and t'. Alternatively, suppose $A \subset C$. Then within t', e^* defines bipartition $(C \cup \{x\})|D$, which also appears as a bipartition in T. Hence, e^* is also shared between T and t'. Therefore, any edge e^* other than e' that is shared between t and T remains shared between t' and T, for all leaves x added by *AddLeaf*. ◀

▶ **Lemma 5.** *OCTAL(T, t) preserves the topology of superleaves in T.*

Proof. We will show this by induction on the number of leaves added. The lemma is trivially true for the base case when just one leaf is added to t. Let the inductive hypothesis be that the lemma holds for adding up to n leaves to t for some arbitrary $n \in \mathbb{N}^+$. Now consider adding $n + 1$ leaves, and choose an arbitrary subset of n leaves to add to t, creating an intermediate tree t' on leaf set K using the algorithm OCTAL. Let x be the next additional leaf to be added by OCTAL.

If x is the first element of a new superleaf to be added, it is trivially true that the topology of its superleaf is preserved, but we need to show that x will not break the monophyly of an existing superleaf in t'. By the inductive hypothesis, the topology of each superleaf already placed in t' has been preserved. Thus, each superleaf placed in t' has some shared edge in t' and $T|_K$ incident to that superleaf. If x were placed onto an edge contained in some existing superleaf, that edge would change its status from being shared to being unique, which contradicts Lemma 4.

The last case is where x is part of a superleaf for the pair (T, t) that already has been added in part to t. *AddLeaf* roots $T|_{K\cup\{x\}}$ at x and removes the edge incident to x, creating rooted tree $T^{(x)}$. The edge incident to the root in $T^{(x)}$ must be a shared edge by the inductive hypothesis. Thus, OCTAL will add x to this shared edge and preserve the topology of the superleaf. ◀

▶ **Lemma 6.** *OCTAL(T, t) returns binary tree T' such that $RF(T, T') = r + 2m$, where m is the number of Type II superleaves for the pair (T, t) and $r = RF(T|_R, t)$.*

Proof. We will show this by induction on the number of leaves added.

Base Case: Assume $|S \setminus R| = 1$. Let x be the leaf in S$\setminus R$. *AddLeaf* adds x to a shared edge of t corresponding to some bipartition A|B, which also exists in $T^{(x)}$.

1. First we consider what happens to the RF distance on the edge x is attached to.

   If x is a Type I superleaf, the edge incident to the root in $T^{(x)}$ will be a shared edge by the definition of Type I superleaf, so *AddLeaf* adds x to the corresponding edge e' in t. The two new bipartitions that are created when subdividing e' will both exist in T by the definition of Type I superleaf so the RF distance does not change.

   If x is a Type II superleaf, either $(A \cup \{x\})|$B or $A|(B \cup \{x\})$ must not exist in $C(T)$. Since *AddLeaf* adds x to a shared edge, exactly one of those new bipartitions must exist in $C(T)$.
2. Now we consider what happens to the RF distance on the edges x is *not* attached to. Lemma 4 shows that *AddLeaf* (and therefore OCTAL) preserves existing shared edges between t and $T|_R$, possibly excluding the edge where x is added.

Thus, the RF distance will only increase by 2 if x is a Type II superleaf, as claimed.

Inductive Step: Let the inductive hypothesis be that the lemma holds for up to n leaves for some arbitrary $n \in \mathbb{N}^+$. Assume $|S \setminus R| = n + 1$. Now choose an arbitrary subset of leaves $Q \subseteq S \setminus R$, where $|Q| = n$, to add to t, creating an intermediate tree t' using the algorithm OCTAL. By the inductive hypothesis, assume t' is a binary tree with the RF distance between $T|_{Q \cup R}$ and t' equal to $r + 2m$, where m is the number of Type II superleaves in Q. *AddLeaf* adds the remaining leaf $x \in S \setminus R$ to a shared edge of t' and $T|_{Q \cup R}$.

1. Lemma 4 shows that *AddLeaf* (and therefore OCTAL) preserves existing shared edges between t' and $T|_{Q \cup R}$, possibly excluding the edge where x is added.
2. Now we consider what happens to the RF distance on the edge x is attached to. There are three cases: (i) x is not the first element of a superleaf (ii) x is the first element of a Type I superleaf or (iii) x is the first element of a Type II superleaf.

   Case (i): If x is not the first element of a superleaf to be added to t, it directly follows from Lemma 5 that OCTAL will not change the RF distance when adding x.

   Case (ii): If x is the first element of a Type I superleaf to be added, then x is attached to a shared edge in the backbone corresponding to some bipartition $A|B$ existing in both $C(t)$ and $C(T|_R)$. Let e' be the edge in t s.t. $\pi_{e'} = A|B$. Note there must exist an edge e in $T|_{Q \cup R}$ producing $A|B$ when restricted to just R. Hence, the bipartition π_e has the form $M|N$ where $(M \cap R) = A$ and $(N \cap R) = B$. We need to show that $M|N \in C(t')$.

   - By Lemma 4, any leaves from Q not attached to e' by OCTAL will preserve this shared edge in t'.
   - Now consider when leaves from Q are added to e' by OCTAL. We decompose M and N into the subsets of leaves existing in either R or Q: let $M = A \cup W$ and $N = B \cup Z$. OCTAL will not cross a leaf from W with a leaf from Z along e' because this would require crossing the shared edge dividing these two groups: any leaf $w \in W$ has the property that $(A \cup \{w\})|B$ is a shared edge and any leaf $z \in Z$ has the property that $A|(B \cup \{z\})$ is a shared edge. Hence, any leaves added from Q that subdivide e' will always preserve an edge between leaves contained in W and Z on e'.

   Thus, $M|N \in C(t')$. Moreover, $(M \cup \{x\})|N$ and $M|(N \cup \{x\})$ are bipartitions in $C(T)$. *AddLeaf* roots T at x and removes the edge incident to x, creating rooted tree $T^{(x)}$. We have shown that the edge incident to the root in $T^{(x)}$ must be a shared edge, so adding x does not change the RF distance.

   Case (iii): If x is the first element of a Type II superleaf to be added, we have shown in Lemma 2 that the RF distance must increase by at least two. Since *AddLeaf* always attaches x to some shared edge e', the RF distance increases by exactly 2 when subdividing e'.

Thus, OCTAL will only increase the RF distance by 2 if x is a new Type II superleaf. ◀

Combining the above results, we establish our main theorem:

▶ **Theorem 7.** *Given unrooted binary trees t and T with the leaf set of t a subset of the leaf set of T, OCTAL(T, t) returns an unrooted binary tree T' that is a completion of t and that has the smallest possible Robinson-Foulds distance to T. Hence, OCTAL finds an optimal solution to the RF Optimal Tree Completion problem. Furthermore, OCTAL runs in $O(n^2)$ time, where T has n leaves.*

Proof. The running time bound was described above in Section 2.3. To prove that OCTAL solves the RF Optimal Tree Completion problem optimally, we need to establish that OCTAL returns an S-completion of the tree t, and that the RF distance between the output tree T' and the reference tree T is the minimum among all S-completions. Since OCTAL always returns a binary tree and only adds leaves into t, by design it produces a completion of t and so satisfies the first property. By Lemma 6, the tree T' output by OCTAL has an RF score that matches the lower bound established in Lemma 3. Hence, OCTAL returns a tree with the best possible score among all S-completions. ◀

3 Methods

We compare OCTAL to the heuristic used in ASTRAL-II [12] for completing incomplete gene trees, as described in [10], noting however that the ASTRAL-II technique is used to expand the search space explored by ASTRAL-II and does not explicitly attempt to minimize the distance to a reference tree.

Datasets used in this simulation study have 26 species (one outgroup) and 200 genes. The simulation protocol is as follows.

(1) SimPhy [9] was used to simulate a model species tree and a collection of gene trees (with branch lengths deviating from a molecular clock) under the MSC model. Note that we refer to these simulated trees as the "true" gene and species trees. Under this process, the true gene trees differ topologically from the true species tree due only to ILS and not to any other processes.
(2) For each individual true gene tree, INDELible [5] was used to simulate DNA sequences under the GTR+Γ model of evolution without insertions or deletions. Model parameters varied across the gene trees and were determined by drawing from a distribution.
(3) Of the 200 genes, 150 genes were randomly selected to be missing data, created by deleting the sequences corresponding to randomly selected species. The number of species missing varies across gene trees from 2 to 20, and on average the estimated gene trees were missing approximately 60% of the species.
(4) RAxML [17] was then used to estimate gene trees from each (often incomplete) gene alignment under the GTRGAMMA model.
(5) ASTRID [20] was run on the 200 estimated gene trees to get a fast estimate of the species tree to be used as a reference tree by OCTAL.
(6) OCTAL (using the ASTRID tree as a reference) and ASTRAL-II were used to complete the estimated gene trees.
(7) The completed gene trees computed by OCTAL and ASTRAL-II were compared to the true gene trees, and the normalized RF error distance between the 150 completed gene trees and the true gene trees was recorded.

Overall we completed 6000 gene trees using OCTAL and ASTRAL-II for this study (20 replicates for 2 model conditions, and each replicate has 150 genes that are incomplete).

These datasets were originally generated for the ASTRAL-II study [12], and full details of this protocol (Steps 1 and 2) are provided in [12]. Data can be downloaded at [4]. OCTAL was scripted using the Python library DendroPy [19] and can be found at https://github.com/pranjalv123/OCTAL-2.

We explored two model conditions which vary in the degree of gene tree heterogeneity due to ILS. The two different levels of ILS can be characterized by the average normalized topological distance (AD) between true gene trees and the true species tree. The moderate ILS condition has AD of 10%, and the high ILS condition has AD of 35%. There were 20 replicate datasets for each of the two model conditions. We used one-sided paired Wilcoxon Signed-Rank tests to determine whether using OCTAL was significantly better than ASTRAL-II on each replicate dataset (200 genes). As 20 replicate datasets were tested per model condition, a Bonferroni multiple comparison correction was applied (i.e., p-values indicating significance are less than 0.0025).

4 Results

Moderate ILS (10% AD): OCTAL frequently produced more accurate gene trees than ASTRAL-II: the average RF error rate for ASTRAL-II was 0.18 ± 0.10 and the average RF error rate for OCTAL was 0.16 ± 0.09. OCTAL had better accuracy than ASTRAL-II on 1,247 genes, ASTRAL-II had better accuracy on 319, and the methods were tied on the remaining 1,434 genes. The degree of improvement in RF rate varied, but was as great as 20% on some replicates. The improvement obtained by using OCTAL over ASTRAL-II was statistically significant in 18 out of 20 of the replicates (Fig. 3). The degrees of missing data and gene tree error did not impact whether OCTAL improved over ASTRAL-II (Fig. 4).

High ILS (35% AD): OCTAL and ASTRAL-II achieved similar levels of accuracy on high ILS condition: the average RF error rate for ASTRAL-II was 0.39 ± 0.11 and the average RF error rate for OCTAL was 0.38 ± 0.11. OCTAL was more accurate than ASTRAL-II on 945 genes, ASTRAL-II was more accurate on 568 genes, and the methods were tied on the remaining 1,487 genes. OCTAL provided a statistically significant advantage over ASTRAL-II in 8 of the 20 replicates, and the differences between the two methods was not statistically significant on the remaining 12 replicates (Fig. 5). Similar to the moderate ILS condition, whether OCTAL or ASTRAL-II performed best appears to be unrelated to the degree of missing data or gene tree error (Fig. 6).

5 Discussion

These results show that OCTAL can be more accurate than ASTRAL-II at completing gene trees and that the degree and frequency of improvement depends on the level of ILS. Under low to moderate ILS, a reasonably accurate estimate of the species tree will be close to the true gene trees, and hence be useful as a reference tree for OCTAL. However, under higher ILS, estimated species trees will be further from the true gene trees, impairing their utility as reference trees for OCTAL.

Therefore, it may make sense to only use OCTAL in conditions with sufficiently low gene tree heterogeneity (i.e., when ILS levels are at most moderate), so that the computed reference tree is topologically close to the gene trees. However, another strategy is to define a set of reference trees rather than a single reference tree, and complete each gene tree based on an appropriately selected reference tree. Statistical binning [11] and its variant weighted statistical binning [2] are examples of this kind of technique. In statistical binning, the genes are clustered into sets (called “bins”) on the basis of gene tree similarity taking bootstrap support into account. Then, for each bin, a concatenated maximum likelihood tree is computed and used as the new gene tree for the genes in the bin. As shown in [11], statistical binning improves gene tree estimation and leads to improved species tree estimation in downstream analyses using summary methods. The same basic approach could be combined with OCTAL, so that after clustering the genes, a reference tree for the genes in each bin could be computed based only on the genes in the bin, and then the gene trees could be completed using the reference tree for their bin. Given the high accuracy obtained by OCTAL when the gene tree heterogeneity is low, this may well produce improved accuracy when the overall heterogeneity is high.

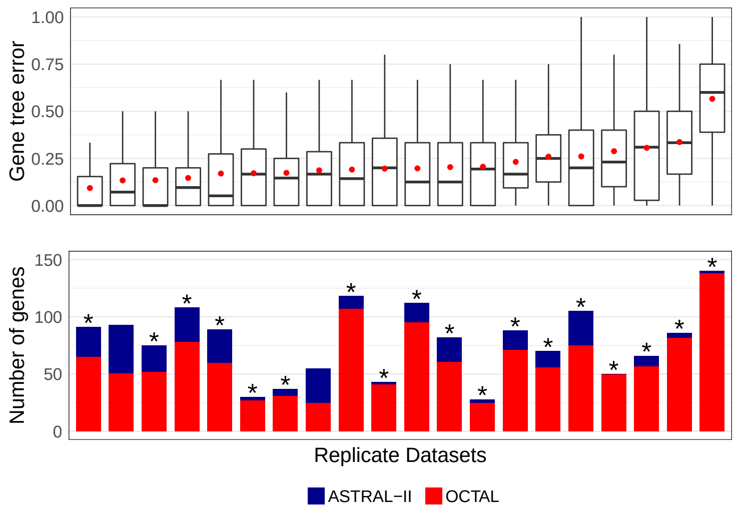


Figure 3 The relative performance of OCTAL and ASTRAL-II under the moderate ILS condition (10% AD) is shown. The top subfigure shows the amount of gene tree estimation error in each of the 150 completed genes for each of the 20 replicates. The bottom subfigure shows the relative performance of OCTAL and ASTRAL-II. The number of gene trees for which OCTAL is better than ASTRAL-II is shown in red, the number of gene trees for which ASTRAL-II is better is shown in blue, and ties are indicated by empty space. OCTAL has a statistically significant improvement over ASTRAL-II on replicates indicated with an asterisk (*).

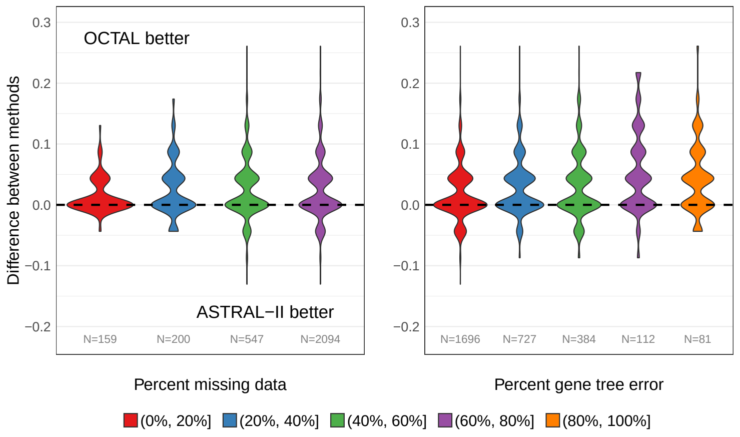


Figure 4 The relative performance of OCTAL and ASTRAL-II under the moderate ILS condition (10% AD) is shown. The y-axis shows the difference in the RF error rate between trees completed using OCTAL and ASTRAL-II. Positive values indicate that OCTAL is better than ASTRAL-II, and negative values indicate that ASTRAL-II is better. The violin plots show that for many genes (indicated by thickness of the violin plot), there is no difference in accuracy between OCTAL and ASTRAL-II. However, when there is a difference between the two methods, OCTAL frequently outperforms ASTRAL-II. This finding holds regardless of the degree of missing data or amount of gene tree estimation error. In the left subfigure, each violin plot includes genes with a certain percent of missing data, e.g., red indicates genes are missing 0-20% of the species. In the right subfigure, each violin plot includes genes with a certain percent gene tree estimation error. The number of genes in each violin plot is provided on the x-axis.

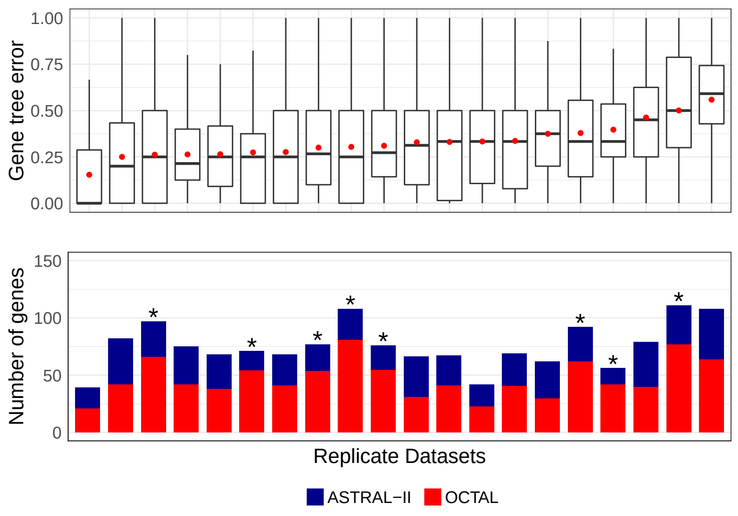


Figure 5 The relative performance of OCTAL and ASTRAL-II under the high ILS condition (35% AD) is shown. The top subfigure shows the amount of gene tree estimation error in each of the 150 completed genes for each of the 20 replicates. The bottom subfigure shows the relative performance of OCTAL and ASTRAL-II. The number of gene trees for which OCTAL is better than ASTRAL-II is shown in red, the number of gene trees for which ASTRAL-II is better is shown in blue, and ties are indicated by empty space. OCTAL has a statistically significant improvement over ASTRAL-II on replicates indicated with an asterisk (*).

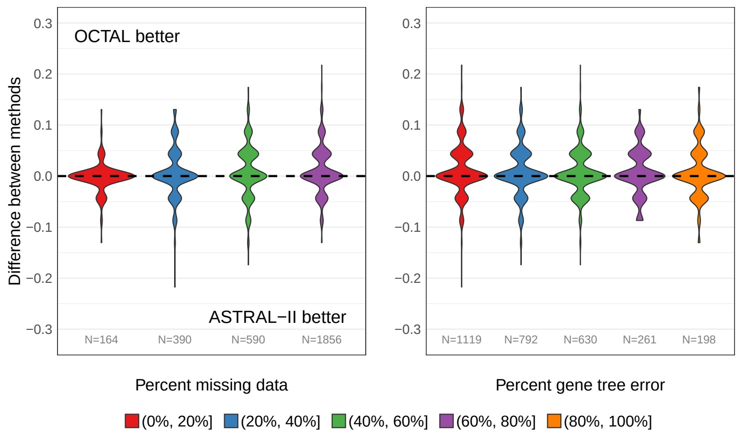


Figure 6 The relative performance of OCTAL and ASTRAL-II under the high ILS condition (35% AD) is shown. The y-axis shows the difference in the RF error rate between the 150 gene trees computed using OCTAL and ASTRAL-II. Positive values indicate that OCTAL is better than ASTRAL-II, and negative values indicate that ASTRAL-II is better. In the left subfigure, each violin plot includes genes with a certain percent of missing data, e.g., red indicates genes are missing 0-20% of the species. In the right subfigure, each violin plot includes genes with a certain percent gene tree estimation error. The number of genes in each violin plot is provided on the x-axis.

6 Conclusions

OCTAL is a simple polynomial time algorithm that can add species into an estimated gene tree and minimize the RF distance with respect to a reference tree. As we saw, OCTAL frequently produces more accurate completed gene trees than ASTRAL-II under both moderate and high ILS conditions; however, the improvement under high ILS conditions is much lower and less frequent than under the low to moderate ILS condition. The results shown here suggest that OCTAL (or some modification of OCTAL) might be useful for coalescent-based species tree estimation using summary methods, especially for those summary methods that are impacted by missing data. As OCTAL only adds missing species and does not provide statistical support for the placements, future work should address this issue. In addition, the current approach assumes that the gene tree is accurate, but typically gene trees have some estimation error; hence, another approach would allow the low support branches in gene trees to be collapsed and then seek a complete gene tree that refines the collapsed gene tree. Finally, this paper addresses tree completion when the distance to be minimized was the Robinson-Foulds distance; yet, many other tree distances have been considered. For example, the Minimize Deep Coalescence (MDC) distance [8] between two trees is a measure of the amount of incomplete lineage sorting, and adding missing species to produce the minimum MDC distance may be more suitable than minimizing the RF distance when datasets have high ILS.

Acknowledgements The authors would like to thank Michael Nute for helpful discussions regarding statistical testing. SC and TW are supported by National Science Foundation Grant Number CCF-1535977. SC is also supported by the Chirag Foundation Graduate Fellowship in Computer Science. EKM and PV are supported by the National Science Foundation Research Fellowship Program under Grant Number DGE-1144245. This research made use of the Illinois Campus Cluster, a computing resource that is operated by the Illinois Campus Cluster Program in conjunction with the National Center for Supercomputing Applications and which is supported by funds from the University of Illinois at Urbana-Champaign.

References

1 Elizabeth S. Allman, James H. Degnan, and John A. Rhodes. Split Probabilities and Species Tree Inference under the Multispecies Coalescent Model. *arXiv:1704.04268*, 2017.

2 Md. Shamsuzzoha Bayzid, Siavash Mirarab, Bastien Boussau, and Tandy Warnow. Weighted statistical binning: enabling statistically consistent genome-scale phylogenetic analyses. *PLOS One*, 10(6):30129183, 2015. doi:10.1371/journal.pone.0129183.

3 J. Gordon Burleigh, Khidir W. Hilu, and Douglas E. Soltis. Inferring Phylogenies with Incomplete Data Sets: A 5-gene, 567-taxon analysis of angiosperms. *BMC Evolutionary Biology*, 9(1):61, 2009. doi:10.1186/1471-2148-9-61.

4 Sarah Christensen, Erin Molloy, Pranjal Vachaspati, and Tandy Warnow. Datasets from the study: Optimal completion of incomplete gene trees in polynomial time using OCTAL, 2017. doi:10.13012/B2IDB-8402610_V1.

5 William Fletcher and Ziheng Yang. INDELible: A Flexible Simulator of Biological Sequence Evolution. *Molecular Biology and Evolution*, 26(8):1879–1888, 2009. doi:10.1093/molbev/msp098.

6 Peter A. Hosner, Brant C. Faircloth, Travis C. Glenn, Edward L. Braun, and Rebecca T. Kimball. Avoiding Missing Data Biases in Phylogenomic Inference: An Empirical Study in the Landfowl (Aves: Galliformes). *Molecular Biology and Evolution*, 33(4):1110–1125, 2016. doi:10.1093/molbev/msv347.

7 Martyn Kennedy and Roderic D. M. Page. Seabird Supertrees: Combining Partial Estimates of Procellariiform Phylogeny. *The Auk*, 119(1):88–108, 2002. doi:10.1642/0004-8038(2002)119[0088:SSCPEO]2.0.CO;2.

8 Wayne Maddison. Gene Trees in Species Trees. *Systematic Biology*, 46(3):523–536, 1997. doi:10.1093/sysbio/46.3.523.

9 Diego Mallo, Leonardo De Oliveira Martins, and David Posada. SimPhy: phylogenomic simulation of gene, locus, and species trees. *Systematic biology*, 65(2):334–344, 2016. doi:doi:10.1093/sysbio/syv082.

10 Siavash Mir arabbaygi (Mirarab). *Novel Scalable Approaches for Multiple Sequence Alignment and Phylogenomic Reconstruction.* PhD thesis, The University of Texas at Austin, 2015. URL: http://hdl.handle.net/2152/31377.

11 Siavash Mirarab, Md. Shamsuzzoha Bayzid, Bastien Boussau, and Tandy Warnow. Statistical binning enables an accurate coalescent-based estimation of the avian tree. *Science*, 346(6215), 2014. doi:10.1126/science.1250463.

12 Siavash Mirarab and Tandy Warnow. ASTRAL-II: Coalescent-based Species Tree Estimation with Many Hundreds of Taxa and Thousands of Genes. *Bioinformatics*, 31(12):i44, 2015. doi:10.1093/bioinformatics/btv234.

13 Erin Molloy and Tandy Warnow. To include or not to include: The impact of gene filtering on species tree estimation methods. *bioRxiv*, 2017. doi:10.1101/149120.

14 David F. Robinson and Leslie R. Foulds. Comparison of Phylogenetic Trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981. doi:10.1016/0025-5564(81)90043-2.

15 Sébastien Roch and Mike Steel. Likelihood-based Tree Reconstruction on a Concatenation of Alignments can be Positively Misleading. *arXiv:1409.2051*, 2014.

16 Michael J. Sanderson, Michelle M. McMahon, and Mike Steel. Phylogenomics with incomplete taxon coverage: the limits to inference. *BMC Evolutionary Biology*, 10, 2010. doi:10.1186/1471-2148-10-155.

17 Alexandros Stamatakis. RAxML Version 8: A tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies. *Bioinformatics*, 30(9), 2014. doi:10.1093/bioinformatics/btu033.

18 Jeffrey W. Streicher, James A. Schulte, II, and John J. Wiens. How Should Genes and Taxa be Sampled for Phylogenomic Analyses with Missing Data? An Empirical Study in Iguanian Lizards. *Systematic Biology*, 65(1):128, 2016. doi:10.1093/sysbio/syv058.

19 Jeet Sukumaran and Mark T. Holder. Dendropy: a Python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571, 2010. doi:10.1093/bioinformatics/btq228.

20 Pranjal Vachaspati and Tandy Warnow. ASTRID: Accurate Species Trees from Internode Distances. *BMC Genomics*, 16(10):S3, 2015. doi:10.1186/1471-2164-16-S10-S3.