

Aaron Darling
Jens Stoye (Eds.)

LNBI 8126

Algorithms in Bioinformatics

13th International Workshop, WABI 2013
Sophia Antipolis, France, September 2013
Proceedings

 Springer

Lecture Notes in Bioinformatics

8126

Edited by S. Istrail, P. Pevzner, and M. Waterman

Editorial Board: A. Apostolico S. Brunak M. Gelfand

T. Lengauer S. Miyano G. Myers M.-F. Sagot D. Sankoff

R. Shamir T. Speed M. Vingron W. Wong

Subseries of Lecture Notes in Computer Science

Aaron Darling Jens Stoye (Eds.)

Algorithms in Bioinformatics

13th International Workshop, WABI 2013
Sophia Antipolis, France, September 2-4, 2013
Proceedings



Springer

Volume Editors

Aaron Darling
University of Technology Sydney
ithree institute
Ultimo, NSW 2007, Australia
E-mail: aaron.darling@uts.edu.au

Jens Stoye
Bielefeld University
Faculty of Technology
Universitätsstraße 25
33615 Bielefeld, Germany
E-mail: jens.stoye@uni-bielefeld.de

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-40452-8

e-ISBN 978-3-642-40453-5

DOI 10.1007/978-3-642-40453-5

Springer Heidelberg New York Dordrecht London

Library of Congress Control Number:2013945447

CR Subject Classification (1998): F.2, J.3, G.2.2, F.1, I.2.6, H.2.8, G.1.2

LNCS Sublibrary: SL 8 – Bioinformatics

© Springer-Verlag Berlin Heidelberg 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

It is our great pleasure to present the proceedings of the 13th Workshop on Algorithms in Bioinformatics (WABI 2013), held at Sophia Antipolis, France, September 2–4, 2013. The WABI 2013 workshop was part of the seven ALGO 2013 conference meetings, which included ESA, IPEC, WAOA, ALGOSENSORS, MASSIVE, and ATMOS. WABI 2013 was hosted by INRIA and Campus SophiaTech, and sponsored by the European Association for Theoretical Computer Science (EATCS) and the International Society for Computational Biology (ISCB). See <http://algo2013.inria.fr/wabi.shtml> for more details.

The Workshop on Algorithms in Bioinformatics highlights research in algorithmic work for bioinformatics, computational biology and systems biology. The emphasis is mainly on 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 goal is to present recent research results, including significant work-in-progress, and to identify and explore directions of future research.

Original research papers (including significant work-in-progress) or state-of-the-art surveys were solicited for WABI 2013 in all aspects of algorithms in bioinformatics, computational biology and systems biology. In response to our call, we received 61 submissions for papers and 27 were accepted. In addition, WABI 2013 hosted a poster session and a distinguished lecture by Bernard Moret, of École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. Prof. Moret founded the WABI series 13 years ago and provided leadership to expand the meeting over the past 13 years. Last year Prof. Moret stepped down from his position of steering the meeting and formed a Steering Committee to facilitate the workshop's continued success.

We would like to sincerely thank the authors of all submitted papers and the conference participants. We also thank the Program Committee and their sub-referees for their hard work in reviewing, discussing, and selecting papers for this year's workshop.

July 2013

Aaron Darling
Jens Stoye

Organization

Steering Committee

Bernard Moret	EPFL, Switzerland
Vincent Moulton	University of East Anglia, UK
Jens Stoye	Bielefeld University, Germany
Tandy Warnow	The University of Texas at Austin, USA

Program Committee

Mohamed Abouelhoda	Cairo University, Egypt
Tatsuya Akutsu	Kyoto University, Japan
Anne Bergeron	University of Quebec at Montreal, Canada
Paola Bonizzoni	Università di Milano-Bicocca, Italy
Guillaume Bourque	McGill University, Canada
Marilia Braga	Inmetro - Ditel, Brazil
C. Titus Brown	Michigan State University, USA
Daniel Brown	University of Waterloo, Canada
David Bryant	University of Otago, New Zealand
Philipp Bucher	Swiss Institute for Experimental Cancer Research, Switzerland
Sebastian Böcker	Friedrich Schiller University Jena, Germany
Rita Casadio	UNIBO, Italy
Cedric Chauve	Simon Fraser University, Canada
Benny Chor	Tel Aviv University, Israel
Lachlan Coin	The University of Queensland, Australia
Lenore Cowen	Tufts University, USA
Keith Crandall	George Washington University, USA
Aaron Darling	University of Technology Sydney, Australia
Nadia El-Mabrouk	University of Montreal, Canada
Eleazar Eskin	University of California, Los Angeles, USA
Liliana Florea	Johns Hopkins University, USA
Martin Frith	CBRC, AIST, Japan
Anna Gambin	Warsaw University, Poland
Olivier Gascuel	LIRMM, CNRS - University of Montpellier 2, France
Nicholas Hamilton	The University of Queensland, Australia
Barbara Holland	University of Tasmania, Australia
Katharina Huber	University of East Anglia, UK
Carl Kingsford	Carnegie Mellon University, USA
Jinyan Li	University of Technology Sydney, Australia

VIII Organization

Zsuzsanna Lipták	University of Verona, Italy
Stefano Lonardi	UC Riverside, USA
Ion Mandoiu	University of Connecticut, USA
Giovanni Manzini	University of Eastern Piedmont, Italy
Paul Medvedev	Pennsylvania State University, USA
Joao Meidanis	University of Campinas / Scylla Bioinformatics, Brazil
Istvan Miklos	Renyi Institute, Hungary
Satoru Miyano	University of Tokyo, Japan
Bernard Moret	EPFL, Switzerland
Burkhard Morgenstern	University of Göttingen, Germany
Vincent Moulton	University of East Anglia, UK
Gene Myers	MPI Cell Biology and Genetics, Germany
Veli Mäkinen	University of Helsinki, Finland
Luay Nakhleh	Rice University, USA
Nadia Pisanti	University of Pisa, Italy and Leiden University, The Netherlands
Teresa Przytycka	NIH, USA
Sven Rahmann	University of Duisburg-Essen, Germany
Ben Raphael	Brown University, USA
Knut Reinert	FU Berlin, Germany
Marie-France Sagot	INRIA Grenoble Rhône-Alpes and Université de Lyon 1, Villeurbanne, France
S. Cenk Sahinalp	Simon Fraser University, Canada
David Sankoff	University of Ottawa, Canada
Russell Schwartz	Carnegie Mellon University, USA
Joao Setubal	University of São Paulo, Brazil
Saurabh Sinha	University of Illinois, USA
Jens Stoye	Bielefeld University, Germany
Krister Swenson	Université de Montréal / McGill University, Canada
Jijun Tang	University of South Carolina, USA
Eric Tannier	INRIA, France
Glenn Tesler	University of California, San Diego, USA
Lusheng Wang	City University of Hong Kong, China
Yuzhen Ye	Indiana University, USA
Louxin Zhang	National University of Singapore, Singapore
Michal Ziv-Ukelson	Ben-Gurion University of the Negev, Israel

Additional Reviewers

Aiche, Stephan	Cicalese, Ferdinando
Badr, Ghada	Conrad, Tim
Biller, Priscila	D'Addario, Marianna
Chateau, Annie	Dao, Phuong

Dittwald, Piotr
Donmez, Nilgun
Drori, Hagai
Duggal, Geet
Duma, Denisa
Goldberg, Tatyana
Gorecki, Pawel
Hazelhurst, Scott
Higashi, Susan
Hormozdiari, Farhad
Hufsky, Franziska
Kluge, Bogusław
Kocsis, Levente
Lacroix, Vincent
Leoncini, Mauro
Marino, Andrea
McPherson, Andrew
Mirebrahim, Seyed
Oesper, Layla
Pardi, Fabio
Pardini, Giovanni
Patterson, Murray
Pinhas, Tamar
Pizzi, Cinzia
Polishko, Anton
Rizzi, Raffaella
Sadakane, Kunihiko
Scheubert, Kerstin
Scornavacca, Celine
Sheridan, Paul
Sinimeri, Blerina
Snir, Sagi
Startek, Michał
Tomescu, Alexandru
Wang, Zhanyong
White, Walton Timothy James
Wohlers, Inken
Yakhini, Zohar
Yasuda, Tomohiro
Yue, Feng
Zakov, Shay
Zanetti, João Paulo Pereira
Łacki, Mateusz

Table of Contents

Extending the Reach of Phylogenetic Inference	1
<i>Bernard M.E. Moret</i>	
Protein (Multi-)Location Prediction: Using Location Inter-dependencies in a Probabilistic Framework	3
<i>Ramanuja Simha and Hagit Shatkay</i>	
Towards Reliable Automatic Protein Structure Alignment	18
<i>Xuefeng Cui, Shuai Cheng Li, Dongbo Bu, and Ming Li</i>	
A Minimum-Labeling Approach for Reconstructing Protein Networks across Multiple Conditions	33
<i>Arnon Mazza, Irit Gat-Viks, Hesso Farhan, and Roded Sharan</i>	
Faster Mass Decomposition	45
<i>Kai Dührkop, Marcus Ludwig, Marvin Meusel, and Sebastian Böcker</i>	
On NP-Hardness of the Paired de Bruijn Sound Cycle Problem	59
<i>Evgeny Kapun and Fedor Tsarev</i>	
Accurate Decoding of Pooled Sequenced Data Using Compressed Sensing	70
<i>Denisa Duma, Mary Wootters, Anna C. Gilbert, Hung Q. Ngo, Atri Rudra, Matthew Alpert, Timothy J. Close, Gianfranco Ciardo, and Stefano Lonardi</i>	
A Novel Combinatorial Method for Estimating Transcript Expression with RNA-Seq: Bounding the Number of Paths	85
<i>Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen</i>	
A Polynomial Delay Algorithm for the Enumeration of Bubbles with Length Constraints in Directed Graphs and Its Application to the Detection of Alternative Splicing in RNA-seq Data	99
<i>Gustavo Sacomoto, Vincent Lacroix, and Marie-France Sagot</i>	
Distribution of Graph-Distances in Boltzmann Ensembles of RNA Secondary Structures	112
<i>Rolf Backofen, Markus Fricke, Manja Marz, Jing Qin, and Peter F. Stadler</i>	

Faster Algorithms for RNA-Folding Using the Four-Russians Method . . .	126
<i>Balaji Venkatachalam, Dan Gusfield, and Yelena Frid</i>	
Algorithms for the Majority Rule (+) Consensus Tree and the Frequency Difference Consensus Tree	141
<i>Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung</i>	
The Generalized Robinson-Foulds Metric	156
<i>Sebastian Böcker, Stefan Canzar, and Gunnar W. Klau</i>	
Computing the Skewness of the Phylogenetic Mean Pairwise Distance in Linear Time	170
<i>Constantinos Tsirogiannis and Brody Sandel</i>	
Characterizing Compatibility and Agreement of Unrooted Trees via Cuts in Graphs	185
<i>Sudheer Vakati and David Fernández-Baca</i>	
Unifying Parsimonious Tree Reconciliation	200
<i>Nicolas Wieseke, Matthias Bernt, and Martin Middendorf</i>	
Sibelia: A Scalable and Comprehensive Synteny Block Generation Tool for Closely Related Microbial Genomes	215
<i>Ilya Minkin, Anand Patel, Mikhail Kolmogorov, Nikolay Vyahhi, and Son Pham</i>	
On the Matrix Median Problem	230
<i>João Paulo Pereira Zanetti, Priscila Biller, and João Meidanis</i>	
A Fixed-Parameter Algorithm for Minimum Common String Partition with Few Duplications	244
<i>Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz, and Irena Rusa</i>	
MSARC: Multiple Sequence Alignment by Residue Clustering	259
<i>Michał Modzelewski and Norbert Dojer</i>	
Mutual Enrichment in Ranked Lists and the Statistical Assessment of Position Weight Matrix Motifs	273
<i>Limor Leibovich and Zohar Yakhini</i>	
Probabilistic Approaches to Alignment with Tandem Repeats	287
<i>Michal Nánási, Tomáš Vinař, and Broňa Brejová</i>	
Multiscale Identification of Topological Domains in Chromatin	300
<i>Darya Filippova, Rob Patro, Geet Duggal, and Carl Kingsford</i>	
Modeling Intratumor Gene Copy Number Heterogeneity Using Fluorescence in Situ Hybridization Data	313
<i>Charalampos E. Tsourakakis</i>	

Phylogenetic Analysis of Cell Types Using Histone Modifications	326
<i>Nishanth Ulhas Nair, Yu Lin, Philipp Bucher, and Bernard M.E. Moret</i>	
Detecting Superbubbles in Assembly Graphs	338
<i>Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya</i>	
Cerulean: A Hybrid Assembly Using High Throughput Short and Long Reads	349
<i>Viraj Deshpande, Eric D.K. Fung, Son Pham, and Vineet Bafna</i>	
Using Cascading Bloom Filters to Improve the Memory Usage for de Bruijn Graphs	364
<i>Kamil Salikhov, Gustavo Sacomoto, and Gregory Kucherov</i>	
Author Index	377

Extending the Reach of Phylogenetic Inference

Bernard M.E. Moret

Laboratory for Computational Biology and Bioinformatics, EPFL, Lausanne, Switzerland
bernard.moret@epfl.ch

One of the most cited articles in biology is a 1973 piece by Theodosius Dobzhansky in the periodical *The American Biology Teacher* entitled “Nothing in biology makes sense except in the light of evolution.” It was also around that time that the development of computational approaches for the inference of phylogenies (evolutionary histories) started. Since then, phylogenetic inference has grown to become one of the standard research tools throughout biological and biomedical research. Today, phylogenetic tools receive over 10,000 citations every year. Concurrently, many groups are engaged in fundamental research in phylogenetic methods and in the design and study of computationally oriented models of evolution for systems ranging from simple genetic sequences through entire genomes to interaction networks. Yet, in spite of the fame of Dobzhansky’s article and the spread of phylogenetic methods beyond the original applications to systematics, the use of methods grounded in evolutionary biology is not as pervasive as it could be.

In this talk, we illustrate some of the algorithmic problems raised by current research and some of the potential new applications of phylogenetic approaches through several projects carried out in our laboratory. The problems arise from combinatorial and algorithmic questions about models of evolution and approaches to the analysis of whole genomes. The new approaches include an extension of the time-tested and universally used comparative method, as well as applications of phylogenetic approaches to genomic transcripts and cell types, objects not typically studied through the lens of evolution.

Comparing the complete genomes of vertebrates is a daunting problem. Not only does each genome have billions of nucleotides, but almost nothing is known for 90% of even the best studied of these genomes. The standard approach today partitions the genomes into syntenic blocks, contiguous intervals along the genome that are viewed as homologous—as descending from the same contiguous interval in the genome of the last common ancestor (LCA). Since mutations, rearrangements, insertions, and other evolutionary events have transformed the LCA genome in different ways along each evolutionary path, one cannot expect to find high levels of similarity between the sequences defined by these intervals. Instead, one looks for markers, nearly perfectly conserved short sequences that are nevertheless long enough to make accidental conservation highly improbable. Homologous blocks should share (most of) their markers and have few, if any, shared markers with non-homologous blocks. Under most reasonable formulations, this problem is NP-hard and solutions to date are mostly *ad hoc*.

The issue of genomic evolution “in the large,” that is, at the scale of markers, genes, or blocks and through rearrangements, duplications, and losses, has been intensely studied for nearly 20 years now, with a number of remarkable algorithmic results. Every new algorithmic result, however, has served mostly to raise interest in more complete

or more sophisticated models, or to motivate new and harder problems. Combining large-scale duplication events with rearrangements events, for instance, appears crucial to the understanding of genomic evolution, but remains poorly solved to date, even though researchers even now attempt to recreate “ancestral” genomes for families of organisms. Some recent observations on the bench suggest that results that appeared to be artifacts of mathematical models (such as small circular chromosomes created in the process of cutting and regluing chromosomes) may in fact arise in nature, confirming that even abstract research in models and algorithms may lead to scientific breakthroughs in biology.

The comparative method (also known as “guilt by association”) attempts to transfer knowledge from one well studied system to another by establishing correspondences. Perhaps the best known example is the transfer of gene annotation from one organism to another using gene homologies. Naturally, such an approach requires a high degree of similarity for the transfer to be successful. We developed a new approach, phylogenetic transfer of knowledge, which leverages known phylogenetic relationships to improve the inference of data about modern systems. We have successfully applied our ProPhyC tool to the refinement of regulatory networks and are currently using it for the refinement and prediction of protein contact networks in protein complexes.

Phylogenetic inference assumes that the systems under study are the product of evolution and share a common ancestor. A phylogeny is simply a tree, with the (unknown) common ancestor at the root and data about the modern systems at the leaves. Evolution, in the form of various events that affect the data used to represent the systems (e.g., sequence data for genomes or directed graphs for regulatory networks), is responsible for the divergence from the common ancestor to the modern forms. Such a model does not apply directly to structures that are influenced by evolution in a less direct, or more complicated manner. Two such are transcripts in genomes exhibiting alternative splicing and cell differentiation in a single organism (or a collection of closely related ones). Transcript evolution takes place at two distinct levels—changes in the underlying gene sequence and changes in the splicing variants. We have developed a two-level framework for inference and used our TrEvoR tool on the entire ASPIC database of alternative transcripts, resulting in much enhanced accuracy in the classification of transcripts.

Cell differentiation is a direct product of development, not evolution, in cells belong to a lineage, not to a clade (group). However, cell types are consistent across individuals of the same species and their characteristic adaptations (e.g., a blood cell, a motor neuron, a red muscle cell, etc.) are the product of long-term evolution, even though in one individual all of these cells are descendants of the same undifferentiated cell in the fertilized egg. Using epigenomic data, it is possible to reconstruct a phylogeny of cell types, that is, a tree in which the children of a node represent various refinements of the single type at the parent. We have carried out such an analysis on human cells of many different types using ChIP-Seq data from the ENCODE project, with robust results that match findings and predictions from developmental biologists.

Our contention is that these are but a few of the numerous further applications of phylogenetic methods in the life sciences, applications that will require both modelling and algorithmic research.

Protein (Multi-)Location Prediction: Using Location Inter-dependencies in a Probabilistic Framework

Ramanuja Simha¹ and Hagit Shatkay^{1,2,3}

¹ Department of Computer and Information Sciences,
University of Delaware, Newark, DE, USA

² Center for Bioinformatics and Computational Biology, DBI,
University of Delaware, Newark, DE, USA

³ School of Computing, Queen's University, Kingston, ON, Canada

Abstract. Knowing the location of a protein within the cell is important for understanding its function, role in biological processes, and potential use as a drug target. Much progress has been made in developing computational methods that predict single locations for proteins, assuming that proteins localize to a single location. However, it has been shown that proteins localize to multiple locations. While a few recent systems have attempted to predict multiple locations of proteins, they typically treat locations as independent or capture inter-dependencies by treating each locations-combination present in the training set as an individual location-class. We present a new method and a preliminary system we have developed that directly incorporates inter-dependencies among locations into the multiple-location-prediction process, using a collection of Bayesian network classifiers. We evaluate our system on a dataset of single- and multi-localized proteins. Our results, obtained by incorporating inter-dependencies are significantly higher than those obtained by classifiers that do not use inter-dependencies. The performance of our system on multi-localized proteins is comparable to a top performing system (YLoc⁺), without restricting predictions to be based only on location-combinations present in the training set.

1 Introduction

Knowing the location of a protein within the cell is essential for understanding its function, its role in biological processes, as well as its potential role as a drug target [1–3]. Experimental methods for protein localization such as those based on mass spectrometry [4] or green fluorescence detection [5, 6], although often used in practice, are time consuming and typically not cost-effective for high-throughput localization. Hence, an ongoing effort is put into developing high-throughput computational methods [7–11] to obtain proteome-wide location predictions.

Over the last decade, there has been significant progress in the development of computational methods that predict a *single* location per protein. The focus on

single-location prediction is driven both by the data available in public databases such as UniProt, where proteins are typically assigned a single location, as well as by an (over-)simplifying assumption that proteins indeed localize to a single location. However, proteins do localize to multiple compartments in the cell [12–15], and translocate from one location to another [16]. Identifying the multiple locations of a protein is important because translocation can serve some unique functions. For instance, GLUT4, an insulin-regulated glucose transporter, which is stored in the intracellular vesicles of adipocytes, translocates to the plasma membrane in response to insulin [17, 18]. As proteins do not localize at random and translocations happen between designated inter-dependent locations, we hypothesize that modeling such inter-dependencies can help in predicting protein locations. Thus, we aim to identify associations or *inter-dependencies* among locations and leverage them in the process of predicting locations for proteins.

Several methods have been recently suggested for predicting multiple locations for proteins. ngLOC [19] uses a Naïve Bayes classifier to obtain *independent predictions* for each single location and combines these individual predictions to obtain a multi-location prediction. Li et al. [20] construct multiple binary classifiers, each using an ensemble of k -nearest neighbor and SVM, where each binary classifier distinguishes between a pair of locations. The predictions from all the classifiers are combined to obtain a multi-location prediction. iLoc-Euk [21] uses a multi-label k -nearest neighbor classifier to predict multiple locations for proteins. Similar methods were used for localizing subsets of eukaryotic proteins [22, 23], virus proteins [24], and bacterial proteins [25, 26]. In contrast to the machine learning-based approaches listed above, KnowPred [27] uses sequence similarity to associate proteins with multiple locations.

Notably, none of the above methods for predicting multiple locations utilizes inter-dependencies among locations in the prediction process. All the above models independently predict each single location and thus do not take into account predictions for other locations. IMMML [28] attempts to make use of *correlation* among pairs of locations, a simple type of dependency, when predicting multiple locations for proteins. This system does not account for more complex inter-dependencies and was not tested on any extensive protein multi-localization dataset. YLoc⁺ [29], a comprehensive system for protein location prediction, uses a naïve Bayes classifier (see e.g. [30]) and captures protein localization to multiple locations by explicitly *introducing a new class for each combination of locations supported by the training set* (i.e. having proteins localized to the combination). Thus, each prediction performed by the naïve Bayes classifier can assign a protein to only those combinations of locations included in the training data. To produce its output, YLoc⁺ transforms the prediction into a multinomial distribution over the individual locations. We also note that as the number of possible location-combinations is exponential in the number of locations, training the naïve Bayes classifier in this manner does not provide a practical model in the general case of multi-localized proteins, beyond the training set. The performance of YLoc⁺ was evaluated using an extensive dataset [29] and is the highest among current multi-location predictors.

In this paper, we present a new method that directly models inter-dependencies among locations and incorporates them into the process of predicting locations for proteins. Our system is based on a collection of Bayesian network classifiers [31]. Each Bayesian Network (BN) related to each classifier corresponds to a single location L . Each such network is used to assign a probability for a protein to be found at location L , given both the protein’s features and *information regarding the protein’s other possible locations*. Learning each BN involves learning the dependencies among the other locations that are primarily related to proteins localizing to location L . For each Bayesian network classifier, its corresponding BN is learnt with the goal to improve the classifier’s prediction quality. The formulation of multi-location prediction as classification via Bayesian networks, as well as the network model are presented in Section 2. Notably, our system does not assume that *all* proteins it classifies are multi-localized, but rather more realistically, that proteins may be assigned to one or more locations.

We train and test our preliminary system on a dataset containing single- and multi-localized proteins previously used in the development and testing of the YLoc⁺ system [29], which includes the most comprehensive collection of multi-localized proteins currently available, derived from the DBMLoc dataset [13]. As done in other studies [10, 11, 29, 32], we use multiple runs of 5-fold cross-validation. The results clearly demonstrate the advantage of using location inter-dependencies. The F_1 score of 81% and overall accuracy of 76% obtained by incorporating inter-dependencies are significantly higher than the corresponding values obtained by classifiers that do not use inter-dependencies. Also, while our system retains a level of performance comparable to that of YLoc⁺ on the same dataset, we note that unlike YLoc⁺, by training the individual classifiers to predict individual – although inter-dependent – locations, the training of our system is not restricted to only those combinations of locations present in the dataset, thus our system is generalizable to multi-locations beyond those included in the training set.

The rest of the paper proceeds as follows: Section 2 formulates the problem of protein subcellular multi-location prediction and briefly provides background on Bayesian networks and relevant notations. Section 3 discusses the structure, parameters, and inter-dependencies comprising our Bayesian network collection, and introduces the learning procedure used for finding them. Section 4 presents details of the dataset, the performance evaluation measures, and experimental results. Section 5 summarizes our findings and outlines future directions.

2 Problem Formulation

As is commonly done in the context of classification, and protein-location classification in particular [8, 11, 29, 33], we represent each protein, P , as a weighted feature vector, $\mathbf{f}^P = \langle f_1^P, \dots, f_d^P \rangle$, where d is the number of features. We view each feature as a random variable F_i representing a characteristic of a protein, such as the presence or absence of a short amino acid motif [8, 32], the relative abundance of a certain amino acid as part of amino-acid composition [19], or

the annotation by a Gene Ontology (GO) term [34]. Each vector-entry, f_i^P , corresponds to the value taken by feature F_i with respect to protein P . In the experiments described here, we use the exact same representation used by Briesemeister et al. [29] as explained in Section 4.1.

We next introduce notations relevant to the representation of a protein’s localization. Let $S = \{s_1, \dots, s_q\}$ be the set of q possible subcellular components in the cell. For each protein P , we represent its location(s) as a vector of 0/1 values indicating the protein’s absence/presence, respectively, in each subcellular component. The location-indicator vector for protein P is thus a vector of the form: $\mathbf{l}^P = \langle l_1^P, \dots, l_q^P \rangle$ where $l_i^P = 1$ if P localizes to s_i and $l_i^P = 0$ otherwise. As with the feature values, each location value, l_i^P is viewed as the value taken by a random variable, where for each location, s_i , the corresponding random variable is denoted by L_i . Given a dataset consisting of m proteins along with their location vectors, we denote the dataset as: $D = \{(P_j, \mathbf{l}^{P_j}) \mid 1 \leq j \leq m\}$. We thus view the task of protein subcellular multi-location prediction as that of developing a classifier (typically learned from a dataset D of proteins whose locations are known) that given a protein P outputs a q -dimensional location-indicator vector that represents P ’s localization.

As described in Section 1, most recent approaches that extend location-prediction beyond a single location (e.g. KnowPred [27], and Euk-mPLoc 2.0 [35]), do not consider inter-dependencies among locations. YLoc⁺ [29] indirectly considers these inter-dependencies by creating a class for each location-combination. Our underlying hypothesis, which is supported by the experiments and the results presented here, is that capturing location inter-dependencies directly can form the basis for a generalizable approach for location-prediction. The training of a classifier for protein multi-location prediction involves learning these inter-dependencies so that the classifier can leverage them in the prediction process. We use Bayesian networks to model such inter-dependencies.

In order to develop a protein subcellular multi-location predictor, we propose to develop a collection of classifiers, C_1, \dots, C_q , where the classifier C_i is viewed as an “expert” responsible for predicting the 0/1 value, l_i^P , indicating P ’s non-localization or localization to s_i . In order to make use of location inter-dependencies, each C_i uses estimates of location indicators of P , \hat{l}_j^P (for all other locations j , where $j \neq i$), along with the feature-values of P , in order to calculate a prediction. We use support vector machines (SVMs) (see e.g. [30]) to compute these estimates. The output of C_i for a protein P is given by

$$C_i(P) = \begin{cases} 1 & \text{If } \Pr(l_i^P = 1 \mid P, \hat{l}_1^P, \dots, \hat{l}_{i-1}^P, \hat{l}_{i+1}^P, \dots, \hat{l}_q^P) > 0.5; \\ 0 & \text{Otherwise.} \end{cases} \quad (1)$$

Further details about the estimation procedure itself are provided in Section 3.2.

Bayesian networks have been used before in many biological applications (e.g. [36–38]). In this paper, we use them to model inter-dependencies among subcellular locations, as well as among protein-features and locations. We briefly introduce Bayesian networks here, along with the relevant notations (see [39] for more details). A Bayesian network consists of a directed acyclic graph G ,

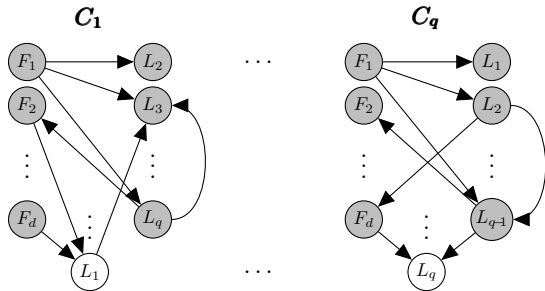


Fig. 1. An example of a collection of Bayesian network classifiers we learn. The collection consists of several classifiers C_1, \dots, C_q , one for each of the q subcellular locations. Directed edges represent dependencies between the connected nodes. We note that there are edges among location variables (L_1, \dots, L_q), as well as between feature variables (F_1, \dots, F_d) and location variables (L_1, \dots, L_q), but not among the feature variables. The latter indicates independencies among features, as well as conditional independencies among features given the locations.

whose nodes are random variables, which in our case represent features, denoted F_1, \dots, F_d , and location indicators, denoted L_1, \dots, L_q . We assume here that all the feature values are discrete. To ensure that, we use the recursive minimal entropy partitioning technique [40] to discretize the features; this technique was also used in the development of YLoc⁺[29].

Directed edges in the graph indicate inter-dependencies among the random variables. Thus, as demonstrated in Figure 1, edges are allowed to appear between feature- and location-nodes, as well as between pairs of location-nodes in the graph. Edges between location-nodes directly capture the inter-dependencies among locations. We note that there are no edges between feature-nodes in our model, which reflects an assumption that features are either independent of each other or conditionally independent given the locations. This simplifying assumption helps speed up the process of learning the network structure from the data, while the other allowed inter-dependencies still enable much of the structure of the problem to be captured (as demonstrated in the results). Further details about the learning procedure itself are provided in Section 3.1.

To complete the Bayesian network framework, each node $v \in \{F_1, \dots, F_d, L_1, \dots, L_q\}$ in the graph is associated with a conditional probability table, θ_v , containing the conditional probabilities of the values the node takes given its parents' values, $\Pr(v \mid Pa(v))$. We denote by Θ the set of all conditional probability tables, and the Bayesian network is the pair (G, Θ) . A consequence of using the Bayesian network structure, is that it represents certain conditional independencies among non-neighboring nodes [39], such that the joint distribution of the set of network variables can be simply calculated as:

$$\Pr(F_1, \dots, F_d, L_1, \dots, L_q) = \prod_{i=1}^d \Pr(F_i \mid Pa(F_i)) \prod_{j=1}^q \Pr(L_j \mid Pa(L_j)). \quad (2)$$

Figure 1 shows an example of a collection of Bayesian network classifiers. The collection consists of Bayesian network classifiers C_1, \dots, C_q , one for each of the

q subcellular locations s_1, \dots, s_q , where each classifier C_i consists of the graph G_i and its set of parameters Θ_i . In each classifier C_i , the location indicator variable L_i is the variable we need to predict and is therefore viewed as *unobserved*, and is shown as an unshaded node in the figure. The feature variables F_1, \dots, F_d are given for each protein and as such are viewed as known or *observed*, shown as shaded nodes in the figure. Finally, the values for the location indicator variables for all locations except for L_i , $\{L_1, \dots, L_q\} - \{L_i\}$, are needed for calculating the predicted value for L_i in the classifier C_i . As such, they are viewed by the classifier as though they are *observed*. Notably, the values of these variables are not known and still need to be estimated.

Thus, the structure and parameters of the network for each classifier C_i (learnt as described in Section 3.1), are used to predict the value of each unobserved variable, L_i . The task of each classifier C_i , is to predict the value of the variable L_i given the values of all other variables F_1, \dots, F_d , and $\{L_1, \dots, L_q\} - \{L_i\}$. Since, as noted above, the values of the location indicator variables L_j ($j \neq i$) are unknown at the point when L_i needs to be calculated, we *estimate* their values, using simple SVM classifiers as described in Section 3.1. We note that other methods, such as expectation maximization, can be used to estimate all the hidden parameters, which we shall do in the future.

3 Methods

As our goal is to assign locations (possibly multiple) to proteins, we use a collection of Bayesian network classifiers, where each classifier C_i , predicts the value (0 or 1) of a single location variable L_i – while using estimates of all the other location variables L_j ($j \neq i$), which are assumed to be known, as far as the classifier C_i is concerned. The estimates of the location values L_j are calculated using SVM classifiers as described in Section 3.1. The individual predictions from all the classifiers are then combined to produce a multi-location prediction. For each location s_i , a Bayesian network classifier C_i must be learned from training data before it can be used. As described in Section 2, each classifier C_i consists of a graph structure G_i and a set of conditional probability parameters, Θ_i , that is: $C_i = (G_i, \Theta_i)$. Thus, our first task is to learn the individual classifiers, i.e. their respective Bayesian network structures and parameters. The individual networks can then be used to predict a protein’s localization to each location.

Given a protein P , each classifier C_i needs to accurately predict the location indicator value l_i^P , given the feature-values of P and estimates of all the other location indicator values \hat{l}_j^P (where $j \neq i$). That is, each classifier C_i in the collection assumes that the estimates of the location-indicator values, \hat{l}_j^P for all other locations s_j (where $j \neq i$) are already known, and is responsible for predicting only the indicator value l_i^P for location s_i , given all the other indicator values. For a Bayesian network classifier this means calculating the conditional probability

$$\Pr(l_i^P = 1 \mid P, \hat{l}_1^P, \dots, \hat{l}_{i-1}^P, \hat{l}_{i+1}^P, \dots, \hat{l}_q^P), \quad (3)$$

under classifier C_i , where $\hat{l}_1^P, \dots, \hat{l}_{i-1}^P, \hat{l}_{i+1}^P, \dots, \hat{l}_q^P$ are all estimated using simple SVM classifiers. The classifiers C_1, \dots, C_q are each learned by directly optimizing

an objective function that is based on such conditional probabilities, calculated with respect to the training data as explained in Section 3.1.

The procedures used for learning the Bayesian network classifiers and to combine the individual network predictions are described throughout the rest of the section.

3.1 Structure and Parameter Learning of Bayesian Network Classifiers

Given a dataset D , consisting of a set of m proteins $\{P_1, \dots, P_m\}$ and their respective location vectors $\{\mathbf{l}^{P_1}, \dots, \mathbf{l}^{P_m}\}$, each classifier C_i is trained so as to produce the “best” prediction possible for the value of the location indicator l_i^P (for location s_i), for any given protein P and a set of estimates of location indicators for all other locations (as shown in Equation 3 above). Based on this aim and on the available training data, we use the *Conditional Log Likelihood (CLL)* as the objective function to be optimized when learning each classifier C_i . Classifiers whose structures were learnt by optimizing this objective function were found to perform better than classifiers that used other structures [31]. This objective function is defined as:

$$CLL(C_i | D) = \sum_{j=1}^m \log \Pr(L_i = l_i^{P_j} | \mathbf{f}^{P_j}, \hat{l}_1^{P_j}, \dots, \hat{l}_{i-1}^{P_j}, \hat{l}_{i+1}^{P_j}, \dots, \hat{l}_q^{P_j}).$$

Each P_j is a protein in the training set and each probability term is the conditional probability of protein P_j to have the indicator value $l_i^{P_j}$ (for location s_i), given its feature vector \mathbf{f}^{P_j} and the current estimates for all the other location indicators are $\hat{l}_k^{P_j}$ (where $k \neq i$), under the Bayesian network structure G_i for the classifier C_i that governs the joint distribution of all the variables in the network (see Equation 2).

To learn a Bayesian network classifier that optimizes this objective function, we use a greedy hill climbing search (see [31, 41] for details). While Grossman and Domingos [31] propose a heuristic method that modifies the basic search depicted by Heckerman et al. [41], we do not employ it in this preliminary study, but rather use the basic search, as it does not prove to be prohibitively time consuming. To find estimates for the location indicator values $\hat{l}_k^{P_j}$, we compute a one-time estimate for each indicator $l_i^{P_j}$ from the feature-values of the protein \mathbf{f}^{P_j} by using an SVM classifier (e.g. [30]). We use the SVM implementation provided by the Scikit-learn library [42] with a Radial Basis Function kernel. We employ q such SVMs, SVM_1, \dots, SVM_q , where each SVM classifier is trained to distinguish one location indicator from the rest, as done in the Binary Relevance approach [43]. The rest of the network parameters are estimated as follows: For each Bayesian network classifier C_i , we use the maximum likelihood estimates calculated from frequency counts in the training dataset, D , to estimate the network parameters (see [31]). To avoid overfitting of the parameters, we apply standard smoothing by adding pseudo-counts for all the events that have zero counts (see [44] for details).

To summarize, at the end of the learning process we have q Bayesian network classifiers, C_1, \dots, C_q , like the ones depicted in Figure 1, and q SVMs, SVM_1, \dots, SVM_q , used for obtaining initial estimates for each location variable for any given protein. We next describe how these classifiers are used to predict the multi-location of a protein P .

3.2 Multiple Location Prediction

Given a protein P , whose locations we would like to predict, we first use the SVMs to obtain preliminary estimates for each of its location indicator values $\hat{l}_1^P, \dots, \hat{l}_q^P$. We then use each of the learned classifiers C_i , and the preliminary values obtained from the SVMs to predict the value of the location indicator l_i^P . The classifier outputs a value of either a 0 or a 1 by thresholding, as shown in Equation 1. The conditional probability of l_i^P given the feature-values of the protein P and the estimates of the location indicator values \hat{l}_j^P (where $j \neq i$) is first calculated as:

$$\Pr(l_i^P = 1 \mid \mathbf{f}^P, \hat{l}_1^P, \dots, \hat{l}_{i-1}^P, \hat{l}_{i+1}^P, \dots, \hat{l}_q^P) = \frac{\Pr(l_i^P = 1, \mathbf{f}^P, \hat{l}_1^P, \dots, \hat{l}_{i-1}^P, \hat{l}_{i+1}^P, \dots, \hat{l}_q^P)}{\sum_{z \in \{0,1\}} \Pr(l_i^P = z, \mathbf{f}^P, \hat{l}_1^P, \dots, \hat{l}_{i-1}^P, \hat{l}_{i+1}^P, \dots, \hat{l}_q^P)}. \quad (4)$$

The joint probabilities in the numerator and the denominator of Equation 4 above are factorized into conditional probabilities using the Bayesian network structure, G_i (see Equation 2). The 0/1 prediction for each l_i^P obtained from each C_i becomes the value of the i 'th position in the location-indicator vector $\langle l_1^P, \dots, l_q^P \rangle$ for protein P . This is the total multi-location prediction for protein P .

In the next section, we describe our experiments using the Bayesian network framework for predicting protein multi-location and the results obtained.

4 Experiments and Results

We implemented our algorithms for learning and using a collection of Bayesian network classifiers as described above using Python and the machine learning library Scikit-learn [42]. We have applied it to a dataset containing single- and multi-localized proteins, previously used for training YLoc⁺ [29]. Below we describe the dataset, the experiments, the evaluation methods we use, and the multiple location prediction results obtained on the proteins from this dataset.

4.1 Data Preparation

In our experiments we use a dataset containing 5447 single localized proteins (originally published as the Höglund dataset [32]) and 3056 multi-localized proteins (originally published as part of the DBMLoc set [13] that is no longer publicly available). The combined dataset was previously used by Briesemeister et al. [29] in

their extensive comparison of multi-localization prediction systems. We report results obtained over the multi-localized proteins for comparing our system to other published systems, since the results for these systems are only available for this subset [29]. For all other experiments described here, we report results obtained over the combined set of single- and multi-localized proteins. We use the exact same representation of a 30-dimensional feature vector as used in YLoc⁺ [29]. The features include sequence-based features, e.g. amino acid composition and those based on PROSITE patterns, as well as on GO annotations. (See [29] for details on the pre-processing, feature construction, and feature selection). The single localized proteins are from the following locations (abbreviations and number of proteins per location is given in parentheses): cytoplasm (*cyt*, 1411 proteins); endoplasmic reticulum (*ER*, 198), extra cellular space (*ex*, 843), golgi apparatus (*gol*, 150), lysosomal (*lys*, 103), mitochondrion (*mi*, 510), nucleus (*nuc*, 837), membrane (*mem*, 1238), and peroxisomal (*per*, 157). The multi-localized proteins are from the following pairs of locations: *cyt_nuc* (1882 proteins), *ex_mem* (334), *cyt_mem* (252), *cyt_mi* (240), *nuc_mi* (120), *ER_ex* (115), and *ex_nuc* (113). Note that all the multi-location subsets used have over 100 representative proteins.

4.2 Experimental Setting and Performance Measures

To compare the performance of our system to that of other systems (YLoc⁺ [29], Euk-mPLOC [45], WoLF PSORT [46], and KnowPred [27]), whose performance on a large set of multi-localized proteins was described in a previously published comprehensive study [29], we use the exact same dataset, employing the commonly used stratified 5-fold cross-validation. As the information about the exact 5-way splits used before is not available, we ran five complete runs of 5-fold-cross-validation (i.e. 25 runs in total), where each complete run of 5-fold cross-validation uses a different 5-way split. The use of multiple runs with different splits helps validate the stability and the significance of the results. To ensure that the results obtained by using our 5-way splits for cross-validation can be fairly compared with those reported before [29], we replicated the YLoc⁺ runs using our 5-way splits, and obtained results that closely match those originally reported by Briestmeister et al [29]. (The replicated F_1 -label score is 0.69 with standard deviation of ± 0.01 , compared to YLoc⁺ reported F_1 -label score of 0.68, and the replicated accuracy is 0.65 with standard deviation of ± 0.01 , compared to YLoc⁺ reported accuracy of 0.64). The total training time for our system is about 11 hours (wall-clock), when running on a standard Dell Poweredge machine with 32 AMD Opteron 6276 processors. Notably, no optimization or heuristics for improving run time were employed, as this is a one-time training. For the experiments described here, we ran 25 training experiments, through 5 times 5-fold cross validation, where the total run time was about 75 hours (wall clock).

We use in our evaluation the *adapted* measures of *accuracy* and F_1 *score* proposed by Tsoumakas [43] for evaluating multi-label classification. Some of these measures have also been previously used for multi-location evaluation [28, 29].

To formally define these measures, let D be a dataset containing m proteins. For a given a protein P , let $M^P = \{s_i \mid l^{P_i} = 1, \text{ where } 1 \leq i \leq q\}$ be the set of locations to which protein P localizes, and let $\hat{M}^P = \{s_i \mid \hat{l}^{P_i} = 1, \text{ where } 1 \leq i \leq q\}$ be the set of locations that a classifier predicts for protein P , where \hat{l}^{P_i} is the 0/1 prediction obtained (as described in Section 3). The multi-label accuracy and the multi-label F_1 score are defined as:

$$Acc = \frac{1}{m} \sum_{j=1}^m \frac{|M^j \cap \hat{M}^j|}{|M^j \cup \hat{M}^j|} \text{ and } F_1 = \frac{1}{m} \sum_{j=1}^m \frac{2|M^j \cap \hat{M}^j|}{|M^j| + |\hat{M}^j|}.$$

Adapted measures of Precision and Recall, denoted Pre_{s_i} and Rec_{s_i} are used to evaluate how well our system classifies proteins as localized or not localized to any single location s_i [29]. The *Multilabel-Precision* is:

$$Pre_{s_i} = \frac{1}{|\{P \in D \mid s_i \in \hat{M}^P\}|} \sum_{P \in D \mid s_i \in \hat{M}^P} \frac{|M^P \cap \hat{M}^P|}{|\hat{M}^P|},$$

and the *Multilabel-Recall* is:

$$Rec_{s_i} = \frac{1}{|\{P \in D \mid s_i \in M^P\}|} \sum_{P \in D \mid s_i \in M^P} \frac{|M^P \cap \hat{M}^P|}{|M^P|}.$$

Note that Pre_{s_i} captures the ratio of the number of correctly predicted multiple locations to the total number of multiple locations predicted, and Rec_{s_i} captures the ratio of the number of correctly predicted multiple locations to the number of original multiple locations, for all the proteins that co-localize to location s_i . Therefore, high values of these measures for proteins that co-localize to the location s_i indicate that the sets of predicted locations that include location s_i are predicted correctly. Additionally, the F_1 -label score used by Briesemeister et al. [29] to evaluate the performance of multi-location predictors is computed as follows:

$$F_1\text{-label} = \frac{1}{|S|} \sum_{s_i \in S} \frac{2 \times Pre_{s_i} \times Rec_{s_i}}{Pre_{s_i} + Rec_{s_i}}.$$

Finally, to evaluate the correctness of predictions made for each location s_i , we use the *standard precision* and *recall* measures, denoted by $Pre\text{-}Std_{s_i}$ and $Rec\text{-}Std_{s_i}$ (e.g. [10]) and defined as:

$$Pre\text{-}Std_{s_i} = \frac{TP}{TP + FP} \text{ and } Rec\text{-}Std_{s_i} = \frac{TP}{TP + FN},$$

where TP (*true positives*) denotes the number of proteins that localize to s_i and are predicted to localize to s_i , FP (*false positives*) denotes the number of proteins that do not localize to s_i but are predicted to localize to s_i , and FN (*false negatives*) denotes the number of proteins that localize to s_i but are not predicted to localize to s_i .

Table 1. Multi-location prediction results, averaged over 25 runs of 5-fold cross-validation, for multi-localized proteins only, using our system, YLoc⁺ [29], Euk-mPloc [45], WoLF PSORT [46], and KnowPred [27]. The F_1 -label score and Acc measures shown for all the systems except for ours are taken directly from Table 3 in the paper by Briesemeister et al. [29]. Standard deviations are provided for our system (not available for other systems).

	Our system	YLoc ⁺ [29]	Euk-mPloc [45]	WoLF PSORT [46]	KnowPred [27]
F_1 -label	0.66 (\pm 0.02)	0.68	0.44	0.53	0.66
Acc	0.63 (\pm 0.01)	0.64	0.41	0.43	0.63

Table 2. Multi-location prediction results, averaged over 25 runs of 5-fold cross-validation, for the combined set of single- and multi-localized proteins, using our system. The table shows the F_1 score, the F_1 -label score, and the accuracy (Acc) obtained for SVMs without using location inter-dependencies and for our system which uses location inter-dependencies. Standard deviations are shown in parentheses.

	F_1	F_1 -label	Acc
SVMs (without using dependencies)	0.77 (\pm 0.01)	0.67 (\pm 0.02)	0.72 (\pm 0.01)
Our system (using dependencies)	0.81 (\pm 0.01)	0.76 (\pm 0.02)	0.76 (\pm 0.01)

Table 3. Multi-location prediction results, per location, averaged over 25 runs of 5-fold cross-validation, for the combined set of single- and multi-localized proteins. Results are shown for the five locations s_i that have the largest number of associated proteins (the number of proteins per location is given in parenthesis): cytoplasm (cyt), extracellular space (ex), nucleus (nuc), membrane (mem), and mitochondrion (mi). The table shows the measures (*standard precision* ($Pre-Std_{s_i}$) and *recall* ($Rec-Std_{s_i}$), and *Multilabel-Precision* (Pre_{s_i}) and *Multilabel-Recall* (Rec_{s_i})), obtained for SVMs without using location inter-dependencies and for our system by using location inter-dependencies. The highest values between the two methods are shown in boldface. Standard deviations are shown in parentheses.

	cyt (3785)	ex (1405)	nuc (2952)	mem (1824)	mi (870)
$Pre-Std_{s_i}$ (SVMs)	0.84 (\pm 0.01)	0.87 (\pm 0.02)	0.79 (\pm 0.02)	0.93 (\pm 0.01)	0.90 (\pm 0.03)
$Pre-Std_{s_i}$ (Our system)	0.84 (\pm 0.01)	0.91 (\pm 0.02)	0.79 (\pm 0.03)	0.90 (\pm 0.01)	0.87 (\pm 0.03)
$Rec-Std_{s_i}$ (SVMs)	0.85 (\pm 0.01)	0.64 (\pm 0.02)	0.72 (\pm 0.02)	0.79 (\pm 0.02)	0.62 (\pm 0.03)
$Rec-Std_{s_i}$ (Our system)	0.86 (\pm 0.01)	0.65 (\pm 0.02)	0.74 (\pm 0.03)	0.80 (\pm 0.02)	0.66 (\pm 0.03)
Pre_{s_i} (SVMs)	0.82 (\pm 0.01)	0.89 (\pm 0.02)	0.83 (\pm 0.01)	0.92 (\pm 0.01)	0.87 (\pm 0.03)
Pre_{s_i} (Our system)	0.81 (\pm 0.02)	0.91 (\pm 0.02)	0.83 (\pm 0.01)	0.90 (\pm 0.01)	0.89 (\pm 0.02)
Rec_{s_i} (SVMs)	0.78 (\pm 0.01)	0.72 (\pm 0.02)	0.77 (\pm 0.01)	0.76 (\pm 0.01)	0.68 (\pm 0.02)
Rec_{s_i} (Our system)	0.80 (\pm 0.01)	0.74 (\pm 0.02)	0.78 (\pm 0.02)	0.78 (\pm 0.01)	0.73 (\pm 0.02)

4.3 Classification Results

Table 1 shows the F_1 -label score and the accuracy for our system in comparison to those obtained by other predictors (as reported by Briesemeister et al. [29], Table 3 there, using the same set of multi-localized proteins and evaluation measures. While the table shows that our system has a slightly lower performance than YLoc⁺, the differences in the values are not statistically significant, and the overall performance level is comparable. Thus our approach performs as

effectively as current top-systems, while having the advantage of directly capturing inter-dependencies among locations in a generalizable manner (that is, without introducing a new location-class for each new location-combination).

Table 2 shows the F_1 score, the F_1 -label score, and the accuracy obtained by the individual SVM classifiers (used for computing estimates of location indicators) without using location inter-dependencies compared with the corresponding values obtained by our system by using location inter-dependencies, on the combined dataset of both single- and multi-localized proteins. All the scores obtained by using inter-dependencies are significantly higher than those obtained by using SVMs alone without utilizing inter-dependencies. These differences are highly statistically significant ($p \ll 0.001$), as measured using the 2-sample t-test [47].

Table 3 shows the prediction results obtained by our system for the five locations that have the largest number of associated proteins: cytoplasm (cyt), extracellular space (ex), nucleus (nu), membrane (mem), and mi (mitochondrion), on the combined dataset of both single- and multi-localized proteins. For each location s_i , we show the *standard precision* ($Pre-Std_{s_i}$) and *recall* ($Rec-Std_{s_i}$) as well as the *Multilabel-Precision* (Pre_{s_i}) and *Multilabel-Recall* (Rec_{s_i}). The table shows values for each of the measures obtained by SVMs without using location inter-dependencies and by our system using location inter-dependencies. When using inter-dependencies, we note that for all locations the *Multilabel-Recall* (Rec_{s_i}) increases (in some cases statistically significantly); while for a few locations (such as cytoplasm and membrane) the *Multilabel-Precision* (Pre_{s_i}) decreases, the decrease is not statistically significant. For instance, when classifying using SVMs without using inter-dependencies Rec_{cyt} is 0.78 and Rec_{mem} is 0.76, while when incorporating the inter-dependencies the recall is 0.80 and 0.78, respectively. Even for locations with fewer associated proteins, e.g. peroxisome, (157 proteins), the *Multilabel-Recall* increases from 0.37 using simple SVMs to 0.65 using our classifier. This demonstrates the advantage of using location inter-dependencies for predicting protein locations, not just for locations that have a large number of associated proteins but also for locations that have relatively few associated proteins.

5 Discussion and Future Work

We presented a new way to use a collection of Bayesian network classifiers taking advantage of location inter-dependencies to provide a generalizable method for predicting possible multiple locations of proteins. The results demonstrate that the performance of our preliminary system is comparable to the best current multi-location predictor YLoc⁺[29], which indirectly addresses dependencies by creating a class for each multi-location combination. Our results also show that utilizing inter-dependencies significantly improves the performance of the location prediction system, with respect to SVM classifiers that do not use any inter-dependencies.

In most biological applications that have used Bayesian networks so far (e.g. [36–38]), the variable-space typically corresponds to genes or SNPs which is

a very large space and necessitates the use of strong simplifying assumptions and many heuristics. In contrast, we note that predicting multiple locations for proteins involves a significantly smaller number of variables (as the number of subcellular components and the number of features for representing proteins are relatively small), making this task ideally suitable for the use of Bayesian networks.

The study presented here is a first investigation into the benefit of directly modeling and using location inter-dependencies. In order to obtain initial estimates for location values, we used a simple SVM classifier, and location inter-dependencies were only learned based on these values. While the results have already shown much improvement with respect to the baseline SVM classifiers, we believe that a better approach would be to simultaneously learn a Bayesian network while estimating the location values using methods such as expectation maximization.

We note that although the dataset we use contains the most extensive available collection of multi-localized proteins, several subcellular locations are not represented in the dataset at all due to the low number of proteins associated with them. Similarly, there is not enough data pertaining to proteins that are localized to more than two locations. We are in the process of constructing a set of multi-localized proteins that will be used in future work to test the performance of our system on novel, and more complex, combinations. We also plan to develop improved approaches for learning models of location inter-dependencies from the available data.

Acknowledgments. We are grateful to S. Briesemeister for so readily providing us with information about the implementation and testing of YLoc⁺.

References

1. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: *Molecular Biology of the Cell*, vol. 4. Garland Science (2002)
2. Rost, B., Liu, J., Nair, R., Wrzeszczynski, K., Ofra, Y.: Automatic prediction of protein function. *Cellular and Molecular Life Sciences* 60(12), 2637–2650 (2003)
3. Bakheet, T., Doig, A.: Properties and identification of human protein drug targets. *Bioinformatics* 25(4), 451–457 (2009)
4. Dreger, M.: Proteome analysis at the level of subcellular structures. *Eur. J. Biochem.* 270, 2083–2092 (2003)
5. Simpson, J., Wellenreuther, R., Poustka, A., Pepperkok, R., Wiemann, S.: Systematic subcellular localization of novel proteins identified by large-scale cDNA sequencing. *EMBO Rep.* 1, 287–292 (2000)
6. Hanson, M., Kohler, R.: Gfp imaging: methodology and application to investigate cellular compartmentation in plants. *J. Exp. Bot.* 52, 529–539 (2001)
7. Nakai, K., Kanehisa, M.: Expert system for predicting protein localization sites in gram-negative bacteria. *Proteins* 11(2), 95–110 (1991)
8. Emanuelsson, O., Nielsen, H., Brunak, S., von Heijne, G.: Predicting subcellular localization of proteins based on their n-terminal amino acid sequence. *J. Mol. Biol.* 300(4), 1005–1016 (2000)

9. Rey, S., Gardy, J., Brinkman, F.: Assessing the precision of high-throughput computational and laboratory approaches for the genome-wide identification of protein subcellular localization in bacteria. *BMC Genomics* 6, 162 (2005)
10. Shatkay, H., Höglund, A., Brady, S., Blum, T., Dönnies, P., Kohlbacher, O.: Sherloc: high-accuracy prediction of protein subcellular localization by integrating text and protein sequence data. *Bioinformatics* 23, 1410–1417 (2007)
11. Blum, T., Briesemeister, S., Kohlbacher, O.: Multiloc2: integrating phylogeny and gene ontology terms improves subcellular protein localization prediction. *BMC Bioinformatics* 10, 274 (2009)
12. Foster, L., de Hoog, C., Zhang, Y., Zhang, Y., Xie, X., Mootha, V., Mann, M.: A mammalian organelle map by protein correlation profiling. *Cell* 125, 187–199 (2006)
13. Zhang, S., Xia, X., Shen, J., Zhou, Y., Sun, Z.: Dbmloc: a database of proteins with multiple subcellular localizations. *BMC Bioinformatics* 9, 127 (2008)
14. Millar, A., Carrie, C., Pogson, B., Whelan, J.: Exploring the function-location nexus: using multiple lines of evidence in defining the subcellular location of plant proteins. *Plant Cell* 21(6), 1625–1631 (2009)
15. Murphy, R.: Communicating subcellular distributions. *Cytometry A* 77(7), 686–692 (2010)
16. Pohlschroder, M., Hartmann, E., Hand, N., Dilks, K., Haddad, A.: Diversity and evolution of protein translocation. *Annu. Rev. Microbiol.* 59, 91–111 (2005)
17. Rea, S., James, D.: Moving glut4: The biogenesis and trafficking of glut4 storage vesicles. *Diabetes* 46(11), 1667–1677 (1997)
18. Russell, R., Bergeron, R., Shulman, G., Young, H.: Translocation of myocardial glut-4 and increased glucose uptake through activation of ampk by aicar. *Am. J. Physiol.* 9, H643–H649 (1997)
19. King, B., Guda, C.: ngloc: an n-gram-based bayesian method for estimating the subcellular proteomes of eukaryotes. *Genome Biology* 8, 3963–3969 (2007)
20. Li, L., Zhang, Y., Zou, L., Zhou, Y., Zheng, X.: Prediction of protein subcellular multi-localization based on the general form of chou’s pseudo amino acid composition. *Protein Pept. Lett.* 19(4), 375–387 (2012)
21. Chou, K., Wu, Z., Xiao, X.: iloc-euk: A multi-label classifier for predicting the subcellular localization of singleplex and multiplex eukaryotic proteins. *PLoS ONE* 6(3), e18258 (2011)
22. Chou, K., Wu, Z., Xiao, X.: iloc-hum: using the accumulation-label scale to predict subcellular locations of human proteins with both single and multiple sites. *Mol. Biosyst.* 8(2), 629–641 (2012)
23. Wu, Z., Xiao, X., Chou, K.: iloc-plant: a multi-label classifier for predicting the subcellular localization of plant proteins with both single and multiple sites. *Mol. Biosyst.* 7(12), 3287–3297 (2011)
24. Xiao, X., Wu, Z., Chou, K.: iloc-virus: a multi-label learning classifier for identifying the subcellular localization of virus proteins with both single and multiple sites. *J. Th. Bio.* 284, 42–51 (2011)
25. Xiao, X., Wu, Z., Chou, K.: A multi-label classifier for predicting the subcellular localization of gram-negative bacterial proteins with both single and multiple sites. *PLoS ONE* 6, e20592 (2011)
26. Wu, Z., Xiao, X., Chou, K.: iloc-gpos: a multi-layer classifier for predicting the subcellular localization of singleplex and multiplex gram-positive bacterial proteins. *Protein Pept. Lett.* 19, 4–14 (2012)

27. Lin, H., Chen, C., Sung, T., Ho, S., Hsu, W.: Protein subcellular localization prediction of eukaryotes using a knowledge-based approach. *BMC Bioinformatics* 10, 8 (2009)
28. He, J., Gu, H., Liu, W.: Imbalanced multi-modal multi-label learning for subcellular localization prediction of human proteins with both single and multiple sites. *PLoS ONE* 7, e37155 (2012)
29. Briesemeister, S., Rahnenfuhrer, J., Kohlbacher, O.: Going from where to why - interpretable prediction of protein subcellular localization. *Bioinformatics* 26, 1232–1238 (2010)
30. Mitchell, T.: *Machine Learning*, 1st edn. McGraw-Hill, Inc., New York (1997)
31. Grossman, D., Domingos, P.: Learning bayesian network classifiers by maximizing conditional likelihood. In: *ICML*, pp. 361–368. ACM (2004)
32. Höglund, A., Dönnies, P., Blum, T., Adolph, H., Kohlbacher, O.: Multiloc: prediction of protein subcellular localization using n-terminal targeting sequences, sequence motifs, and amino acid composition. *Bioinformatics* 22, 1158–1165 (2006)
33. Garg, A., Raghava, G.: Eslpred2: improved method for predicting subcellular localization of eukaryotic proteins. *BMC Bioinformatics* 9(1), 503 (2008)
34. Huang, W., Tung, C., Ho, S., Hwang, S., Ho, S.: Proloc-go: Utilizing informative gene ontology terms for sequence-based prediction of protein subcellular localization. *BMC Bioinformatics* 9 (2008)
35. Chou, K., Shen, H.: A new method for predicting the subcellular localization of eukaryotic proteins with both single and multiple sites: Euk-mploc 2.0. *PLoS ONE* 5, e9931 (2010)
36. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using bayesian networks to analyze expression data. *J. Comput. Biol.* 7(3-4), 601–620 (2000)
37. Segal, E., Taskar, B., Gasch, A., Friedman, N., Koller, D.: Rich probabilistic models for gene expression. *Bioinformatics* 17(suppl. 1), S243–S252 (2001)
38. Lee, P., Shatkay, H.: Bntagger: improved tagging snp selection using bayesian networks. *Bioinformatics* 22(14), e211–e219 (2006)
39. Jensen, F., Nielsen, T.: *Bayesian Networks and Decision Graphs*, 2nd edn. Springer Publishing Company, Incorporated (2007)
40. Fayyad, U., Irani, K.: Multi-interval discretization of continuous-valued attributes for classification learning. In: *IJCAI*, pp. 1022–1029 (1993)
41. Heckerman, D., Chickering, D.: *Learning Bayesian networks: The combination of knowledge and statistical data*. Kluwer Academic Publishers, Boston (1995)
42. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, F., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. *Journal of Machine Learning Research* 12, 2825–2830 (2011)
43. Tsoumakas, G., Katakis, I.: Multi-label classification: An overview. *IJDWM* 3, 1–13 (2007)
44. Russell, S., Norvig, P.: *Artificial Intelligence - A Modern Approach*, 3rd edn. Pearson Education (2010)
45. Chou, K., Shen, H.: A fusion classifier for large-scale eukaryotic protein subcellular location prediction by incorporating multiple sites. *J. Proteome Res.* 6, 1728–1734 (2007)
46. Horton, P., Park, K., Obayashi, T., Fujita, N., Harada, H., Adams-Collier, C., Nakai, K.: WoLF PSORT: Protein localization predictor. *Nucleic Acids Research* 35, W585–W587 (2007)
47. DeGroot, M.: *Probability and Statistics*, 2nd edn. Addison-Wesley (1986)

Towards Reliable Automatic Protein Structure Alignment

Xuefeng Cui¹, Shuai Cheng Li², Dongbo Bu³,
and Ming Li^{1,*}

¹ University of Waterloo, Ontario, Canada

² City University of Hong Kong, Hong Kong, China

³ Chinese Academy of Sciences, Beijing, China

mli@cs.uwaterloo.ca

Abstract. A variety of methods have been proposed for structure similarity calculation, which are called structure alignment or superposition. One major shortcoming in current structure alignment algorithms is in their inherent design, which is based on local structure similarity. In this work, we propose a method to incorporate global information in obtaining optimal alignments and superpositions. Our method, when applied to optimizing the TM-score and the GDT score, produces significantly better results than current state-of-the-art protein structure alignment tools. Specifically, if the highest TM-score found by TMalign is lower than 0.6 and the highest TM-score found by one of the tested methods is higher than 0.5, there is a probability of 42% that TMalign failed to find TM-scores higher than 0.5, while the same probability is reduced to 2% if our method is used. This could significantly improve the accuracy of fold detection if the cutoff TM-score of 0.5 is used.

In addition, existing structure alignment algorithms focus on structure similarity alone and simply ignore other important similarities, such as sequence similarity. Our approach has the capacity to incorporate multiple similarities into the scoring function. Results show that sequence similarity aids in finding high quality protein structure alignments that are more consistent with eye-examined alignments in HOMSTRAD. Even when structure similarity itself fails to find alignments with any consistency with eye-examined alignments, our method remains capable of finding alignments highly similar to, or even identical to, eye-examined alignments.

1 Introduction

Proteins function in living organisms as enzymes, antibodies, sensors, and transporters, among myriad other roles. The understanding of protein function has great implications to the study of biological and medical sciences. It is widely accepted that protein function is determined mainly by structure. Protein structures are often aligned for their common substructures, to discover functionally

* Corresponding author.

or evolutionarily meaningful structure units. A very large amount of data is available for such studies; the number of known protein structures (the Protein Data Bank) has exceeded 90,000 [1]. Research in structure alignments has intensified recently to enable efficient searches of such databases.

Protein structures are usually modeled as 3-dimensional coordinates of atoms. Thus, the alignment of two protein structures can be modeled as an optimization problem to minimize the distance between two protein structures after a specific rotation and translation. One problem with such comparisons is that the time complexity is typically high. As a result, current methods for the problem are heuristic in nature [2–12].

For example, TMalign [13] creates an initial alignment through sequence and secondary structure alignments and extracts an initial *rotation and translation* (ROTRAN) accordingly. Then, the ROTRAN is improved iteratively until convergence. This approach suffers from possibly dissatisfactory initial alignments and from a lack of optimality guarantees in the final results. TMalign was improved by the fragment-based approach in fr-TM-align [14], in which local structure alignments are computed and represented by the fragment alignments. A dynamic programming technique is then employed to optimize the score function. However, this method only guarantees the quality of the local alignment rather than of the global alignment.

An alignment of two subsets of residues (or C_α atoms) corresponds to a ROTRAN. Unlike fr-TM-align, we also consider the situation in which the small sets contain remote residues. In addition, to overcome the problem of computational inefficiency, we choose to filter the ROTRANs by clustering rather than by using an exhaustive method.

Experimental results suggest that both local fragments and remote fragment pairs show significant contribution to finding higher TM-scores [15] and to finding higher GDT scores [16], as stated in Sections 3.1 and 3.2, respectively. Specifically, if the highest TM-score found by TMalign [13] is lower than 0.6 and the highest TM-score found by one of the tested methods is higher than 0.5, there is a probability of 42% that TMalign failed to find TM-scores higher than 0.5, while the same probability is reduced to 2% with our method. Our method is also capable of finding alignments with significantly (up to 0.21) higher TM-scores. This could significantly improve the accuracy of fold detection if the cutoff TM-score of 0.5 is used.

Another limitation of current protein structure alignment scoring functions, the TM-score [15] and the LG-score [17], is that only protein structure similarity is taken into consideration, while other important protein similarities, such as sequence similarity, are ignored. It has been observed that many protein structure alignments, based only on protein structure similarity are highly sensitive to conformational changes [18]. Recently, sequence similarity has been incorporated into the scoring function [19, 20]. In this paper we introduce a new scoring function incorporating a variety of protein similarities.

In Section 3.3, we demonstrate that sequence similarity enables discovery of high quality protein structure alignments that are more consistent with

eye-examined alignments. Even when structure similarity itself fails to find alignments with any consistency with eye-examined alignments in HOMSTRAD [21], our method is nevertheless able to find alignments highly similar to, or even identical to, the eye-examined alignments. When the aligned protein structures contain a high percentage of helices, TM-score [15] involving only structure similarity sometimes cannot avoid shifting the HOMSTRAD alignment by a few residues. In our experiment, such shifting tends to be avoided by our scoring function, which involves both structure and sequence similarities.

2 Method

Our protein structure alignment search method can be divided into two parts: the search algorithm and the scoring function. In Section 2.1, we describe our search algorithm, which samples and selects near optimal alignments reliably and efficiently. In Section 2.2, we describe our scoring function for evaluating the quality of an alignment accurately.

2.1 Protein Structure Alignment Search Algorithm

Given a protein structure alignment scoring function, finding the optimal alignment involves finding the optimal ROTRAN that maximizes the alignment score. Assume that there exists a near optimal ROTRAN that minimizes the RMSD of two small sets of C_α atoms. We find the near optimal structure alignment by sampling ROTRANs in four steps: (1) ROTRANs are initially sampled from local fragment alignments and from remote fragment pair alignments; (2) noise ROTRANs are filtered out by clustering; (3) one representative alignment for each ROTRAN cluster is selected based on alignment scores; (4) the selected alignments are refined by random ROTRAN sampling. Steps one through four are discussed in this section and our scoring function is discussed in Section 2.2.

First, an initial set of ROTRANs must be sampled. Here, the primary concern is to have several good candidates, instead of to have a high signal-to-noise ratio, which is addressed in the next step. Finding good candidates is done by calculating the optimal ROTRAN that minimizes RMSD between one or two fragments from each protein structure. In case of a single fragment from each protein structure, we call it local fragment. In case of two fragments from each protein structure, we call them remote fragment pair. Here, we require the pair of remote fragments to be of the same size and to be at least three residues away from each other to avoid modeling information redundant to the local fragments. In practice, a significantly large number of ROTRANs with the lowest RMSDs are kept for the next step, and the actual number of ROTRANs is selected empirically as stated in Section 3.1.

Since the initial set of ROTRANs may contain a great deal of noise, we try to filter out most of the noise with a star-like k-median clustering algorithm in the second step. Assuming that we know the maximum distance ϵ between the

median of a cluster and any member of the same cluster, an approximate clustering is applied using a neighbor graph: each vertex represents a rotation matrix, and two vertices are connected if and only if the distance between them is at most ϵ . For each iteration, the vertex with the highest degree and its neighbors are grouped into a cluster, and are removed from the neighbor graph. The iteration repeats until either there are no vertices of degree higher than one or until the maximum number of clusters is reached. The unclustered ROTRANs are treated as noise. Similar approximate clustering algorithms have been used [22] and studied [23].

To complete the clustering algorithm, we need a distance function between ROTRANs. The Riemannian distance is a widely used distance metric measuring the length of the shortest geodesic curve between two rotation matrices [24]. Since the transition vector can be calculated by the rotation matrix and the weight centers of the aligned residues, we use Riemannian distances between rotation matrices to avoid using redundant information when clustering ROTRANs.

For each cluster, we find the representative alignment defined by the ROTRAN that yields the highest alignment score within the cluster. The alignment score is defined in Section 2.2, and is calculated by the Needleman-Wunsch dynamic programming algorithm [25]. Since dynamic programming is computationally expensive, the number of clusters in the previous step must be carefully determined to avoid wasting computation on clusters of noise. After all alignment scores have been calculated, the top scored alignments are selected for the refinement step.

Finally, we refine the selected representative alignments by random ROTRAN sampling. Specifically, for each alignment to be refined, six aligned residue pairs are randomly selected from the alignment, the ROTRAN that minimizes RMSD of the aligned residue pairs is calculated, the alignment score of the alignment defined by the sampled ROTRAN is also calculated, and the previous steps are repeated until there are no improvements after $l_1 l_2$ iterations, where l_1 and l_2 are the number of residues of the two aligned protein structures.

The example shown in Figure 1 demonstrates the efficiency of our protein structure alignment search algorithm, when aligning SCOP domains d3k2aa₁ and d2cufa1 [26]. In the figure, each coordinate represents a ROTRAN because the coordinate is calculated by applying the rotation matrix of the ROTRAN on the coordinate $(1, 0, 0)$. By looking at the initially sampled ROTRANs shown in Figure 1(a), we can see that the ROTRANs have a non-uniform distribution, and the ROTRANs with a small number of neighbors are potential noise candidates. After clustering, the four largest clusters include 19% of the initially sampled ROTRANs, as shown in Figure 1(b). Note that the optimal ROTRAN that maximizes the alignment score is located in the largest cluster, which includes 13% of the initially sampled ROTRANs. Therefore, our search algorithm is highly efficient because the alignment score calculation (by the computationally expensive dynamic programming algorithm) for noise ROTRANs is mainly

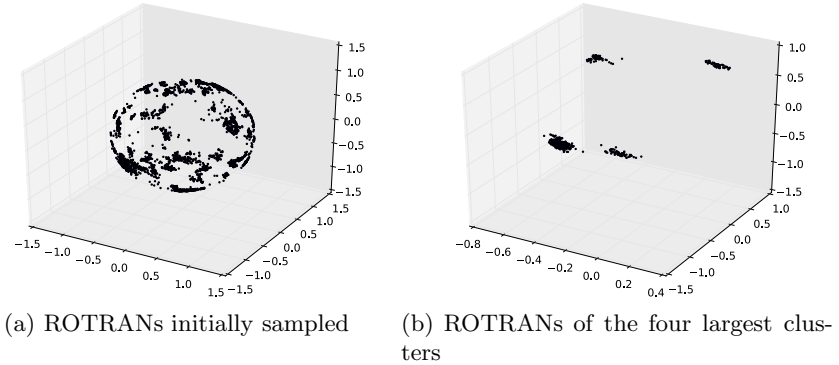


Fig. 1. ROTRANs before and after clustering when aligning SCOP domains d3k2aa_ and d2cufa1: each ROTRAN is represented by a coordinate that is calculated by applying the rotation matrix of the ROTRAN on coordinate (1, 0, 0)

eliminated. It is also possible to trade accuracy for speed by reducing the number of sampled ROTRANs and reducing the number of clusters.

Our search algorithm is both efficient and reliable. Since similar protein structures tend to have many local fragments, or remote fragment pairs with small RMSDs, and similar rotation matrices, these rotation matrices tend to form to a large cluster in our method. Since the rotation matrix space is limited and we assume that the maximum distance between two rotation matrices within a cluster is a constant, the maximum number of clusters within the rotation matrix space is limited. This implies that the number of ROTRANs required to accurately identify large clusters is also limited. Therefore, it is only necessary to sample a limited number of ROTRANs, which is sufficient to identify the large cluster containing near optimal ROTRANs.

2.2 Protein Structure Alignment Scoring Function

TM-score [15], based on LG-score [17], is one of the most successful protein structure alignment scoring functions. However, one limitation of TM-score and LG-score is that they use only protein structure similarity while they ignore other protein similarities, such as the sequence similarity. It has been observed that many protein structure alignments, based only on protein structure similarity, are highly sensitive to conformational changes [18]. This suggests the incorporation of other protein similarities, such as the sequence similarity, in the protein structure alignment scoring function. Here, we introduce a new scoring function incorporating variety kinds of protein similarity as follows:

$$S = \frac{1}{L_r} \sum_{i \leq l} \frac{1}{1 + f_a(D_1(i), D_2(i), \dots, D_n(i))},$$

where L_r is the reference protein size; l is the number of aligned residue pairs of the alignment; f_a is the weighted averaging function (e.g. arithmetic, geometric or harmonic average); $D_k(i)$ is the normalized distance of the i -th aligned residue pair using the k -th distance function; and n is the number of distance functions incorporated. If there is $n = 1$ and $D_1(i) = (d_i/d_0)^2$, where d_i is the distance between the C_α atoms of the i -th aligned residue pair and d_0 is a normalization factor, our scoring function is identical to the LG-score [17]. If there is also $d_0 = 1.24(L_r - 15)^{1/3} - 1.8$, our scoring function is identical to the TM-score [15]. Thus, LG-score and TM-score are two special cases of our scoring function.

As an initial study on our new scoring function, we focus on the geometric average of the normalized C_α distance $D_1(i)$ and the normalized amino acid distance $D_2(i)$ as follows:

$$S = \frac{1}{L_r} \sum_{i \leq l} \frac{1}{1 + \sqrt[1+w]{D_1(i)D_2^w(i)}},$$

where w is a weighting factor. As with TM-score [15], we define the normalized C_α distance as

$$D_1(i) = \left(\frac{d_i}{d_0}\right)^2,$$

where $d_0 = 1.24(L_r - 15)^{1/3} - 1.8$. Based on the popular BLOSUM62 matrix [27, 28], we define the normalized amino acid distance as

$$D_2(i) = 2^{-M(P_i, Q_i)} = 2^{-\lambda \log \frac{P(P_i, Q_i)}{P(P_i)P(Q_i)}} = \left(\frac{P(P_i)P(Q_i)}{P(P_i, Q_i)}\right)^\lambda,$$

where M is the BLOSUM62 matrix, (P_i, Q_i) is the i -th aligned residue pair, λ is a scaling factor, $P(P_i, Q_i)$ is the probability of amino acid P_i aligning to amino acid Q_i , and $P(P_i)$ and $P(Q_i)$ are the probabilities of amino acid P_i and amino acid Q_i , respectively. Instead of using the default scaling factor λ , it is treated here as a parameter to control the rate of mutation.

An appealing property shared between TM-score [29] and our scoring function is that the in-favored protein structure alignments tend to have scores higher than 0.5. If the C_α distance between the i -th aligned residue pair is in-favored, there is $d_i < d_0$ and thus $D_1(i) < 1$. If the amino acid distance between the i -th aligned residue pair is in-favored, there is $P(P_i, Q_i) > P(P_i)P(Q_i)$ and thus $D_2(i) < 1$. Then, for the i -th aligned residue pair, there is $D_1(i)D_2(i) < 1$ and thus $1/(1 + \sqrt[1+w]{D_1(i)D_2^w(i)}) > 0.5$. Therefore, if many in-favored aligned residue pairs occur in the alignment, our protein structure alignment score tends to be higher than 0.5.

3 Result

We included three experiments to demonstrate that the protein structure alignments found by using our method are not only higher scored but are also more

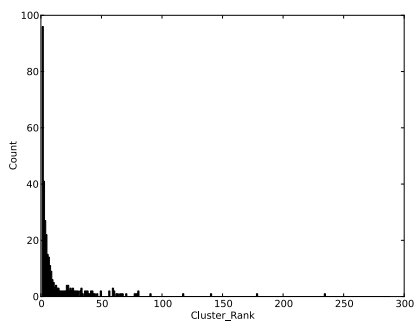
consistent with those alignments examined visually by human-beings. In Section 3.1, we compared our search algorithm to current state-of-the-art search algorithms, TMalign [13] and fr-TM-align [14], to demonstrate that our method tends to find alignments with higher TM-scores [15]. In Section 3.2, we compared our search algorithm to SPalign [30] to demonstrate that our method tends to find alignments with higher GDT scores [16]. In Section 3.3, we compared our scoring function to TM-score [15] to demonstrate that our method tends to find alignments more consistent with the eye-examined alignments in HOMSTRAD [21].

3.1 Search Algorithm Evaluation on TM-Score

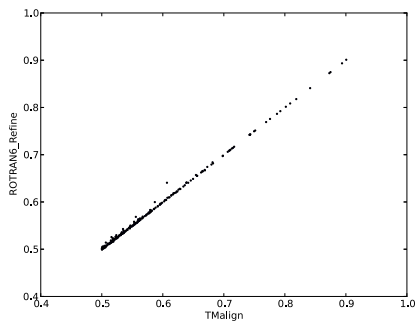
To demonstrate reliability, we repeated the alignment experiment for the 200 non-homologous protein structures, which have sizes of between 46 and 1058, have a sequence identity cutoff of 30%, and are used by TM-align [13]. We compared our results with that of current methods, TM-align and fr-TM-align [14]. Here, we used TM-score [15] normalized by the smaller protein size as the scoring function. Since fr-TM-align does not support normalization by the smaller protein size, TM-score normalized by the smaller protein size is calculated based on the rotation matrix returned by fr-TM-align. Since biologists tend to be more interested in similar protein structures within the same protein fold, and the TM-score of 0.5 is a good approximate threshold for protein fold detection [29], only the 350 protein structure alignments with TM-scores higher than 0.5 (found by at least one of the tested methods) are included in this analysis.

For the experiment settings in the algorithm described in Section 2.1, we used local fragments of size 12, and remote fragment pairs of size 3. Such experiment settings are called L12R3align. To study the contributions of using local fragments and using remote fragment pairs, we simplified our method to two variants: L12align, that used only local fragments of size 12, and R3align, that used only remote fragment pairs of size 3. For consistency, we selected 1,536 local fragments of size 12 and 1,536 remote fragment pairs of size three in the sampling step, used $\epsilon = 10^\circ$ in the clustering step, stopped clustering when 288 clusters were found, and selected eight clusters in the refinement step in all experiments for this section. With L12R3align, the elapsed time required to finish this experiment was approximately 4.5 hours on a computer with dual Intel Xeon X5660 2.8GHz CPUs and dual Nvidia GeForce GTX 670 GPUs. Thus, each pairwise alignment took approximately 0.8 seconds on average.

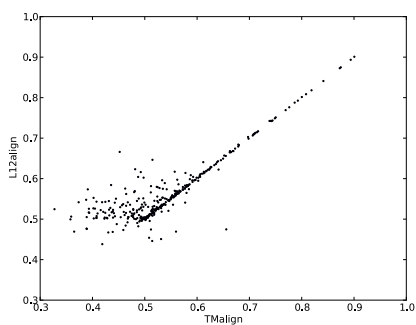
First, we would like to evaluate the ROTRAN filtering step described in Section 2.1. Figure 2(a) shows the cluster rank that contains the optimal ROTRAN with the highest TM-score [15]. Here, we focus on the results of using local fragments because the results of using remote fragment pairs draws similar conclusions. Specifically, 28% of the optimal ROTRANs are from the largest cluster and 72% of the optimal ROTRANs are from the largest ten clusters. Moreover, only 1% of the optimal ROTRANs are not from the largest 100 clusters. This demonstrates that the optimal ROTRAN tends to have many similar ROTRANs that



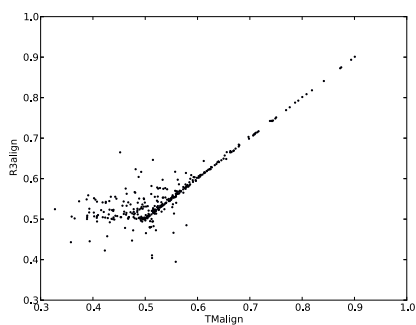
(a) Cluster rank containing the optimal



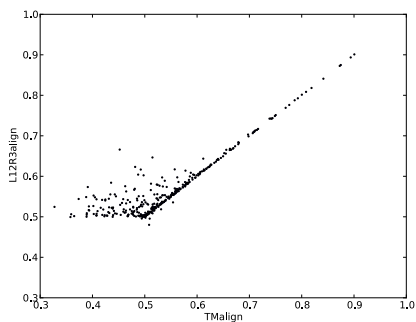
(b) TM-score before and after refinement



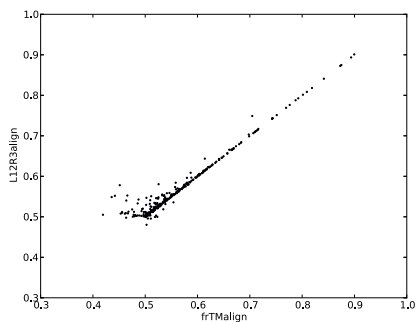
(c) TMalign v.s. L12align



(d) TMalign v.s. R3align



(e) TMalign v.s. L12R3align



(f) fr-TM-align v.s. L12R3align

Fig. 2. Comparisons of the highest TM-scores found by TMalign and by using our method

minimize the RMSD of local fragment alignments, and that these ROTRANs tend to form a large cluster, which can be identified easily by clustering the sampled ROTRANs.

Next, we will demonstrate that our refinement step using randomly selected ROTRANs, as described in Section 2.1, is able to consistently find protein structure alignments with similar or higher TM-scores [15]. Figure 2(b) shows the TM-score before and after refining the optimal alignment found by TMalign [13]. It can be seen that the TM-scores are mostly similar, while our refinement occasionally improves the TM-score by up to 0.10. Specifically, after refinement, all TM-scores are at most 0.0029 lower, while 3% of the TM-scores are at least 0.01 higher. Recall that the random ROTRANs used in the refinement step are generated by finding the ROTRAN that minimizes the RMSD of size randomly selected aligned residue pairs from the alignment. Thus, this result also verifies our assumption that there exists a near optimal ROTRAN that minimizes the RMSD of two small sets of C_α atoms.

To support our choices of local fragment size and of remote fragment pair size, the highest TM-scores found by L12align and R3align are compared to those found by TMalign in Figures 2(c) and 2(d), respectively. For protein structure pairs that have TMalign TM-scores higher than 0.6, both L12align and R3align can reliably find high quality alignments with similar TM-scores. For the other protein structure pairs, both L12align and R3align tend to improve TM-scores, although there may be some reductions of TM-scores. This demonstrates that both L12align and R3align are capable of finding high quality alignments that are comparable to or even better than those found by TMalign. In fact, the local fragment size of 12 has also been used by fr-TM-align [14].

The improvements of TM-scores found by L12R3align over those found by TMalign are shown in Figure 2(e). We see that TM-scores found by L12R3align are mainly higher than those found by TMalign for the 284 protein structure pairs that have TMalign TM-scores lower than 0.6. Specifically, L12R3align improves TM-scores by 0.03 on average and by 0.21 in the best case. Moreover, 14% of the TM-scores are improved by at least 0.1, 30% of the TM-scores are improved by at least 0.05, and only 2% of the TM-scores are reduced by at most 0.03. Comparing to Figures 2(c) and 2(d), the number of TM-scores found by our method that are lower than those found by TMalign is significantly reduced using both local fragments and remote fragment pairs.

If the highest TM-score found by TMalign is lower than 0.6 and the highest TM-score found by one of the tested methods is higher than 0.5, there is a probability of 42% that TMalign failed to find TM-scores higher than 0.5. In such cases, L12R3align tends to discover better protein structure alignments with (possibly significantly) higher TM-scores, with a probability of only 2% that L12R3align failed to find TM-scores higher than 0.5. This could significantly improve fold detection results. Interestingly, L12R3align tends to improve TM-scores more for α -proteins, while never reduces TM-scores for β -proteins.

In addition to comparison with TMalign, the TM-scores found by L12R3align are also compared with those found by fr-TM-align [14] as shown in Figure 2(f).

Note that TM-scores found by L12R3align are also mainly higher than those found by fr-TM-align for protein structure pairs that have fr-TM-align TM-scores lower than 0.6. Specifically, L12R3align improves TM-scores by up to 0.13, while it reduces TM-scores by at most 0.02. Moreover, L12R3align finds 28 more TM-scores that are higher than 0.5.

3.2 Search Algorithm Evaluation on GDT Score

In addition to TM-score [15], GDT [16] score is also one of the most popular protein structure alignment scoring function [31]. Thus, we repeated the experiment in Section 3.1, but compared the GDT scores found by our method to those found by SPalign [30], which is a new protein structure alignment tool that uses a search algorithm similar to that of TMalign. SPalign aims to find one of the highest SP-score, the highest TM-score, or the highest GDT score. If we included SPalign in the previous experiment in Section 3.1, it would perform slightly better than TMalign on average. Thus, SPalign has a effective search algorithm and it should be a candidate for finding the highest GDT score for comparison. Again, only the 339 protein structure alignments with GDT scores higher than 0.5, found by at least one of the tested methods, are included in this analysis.

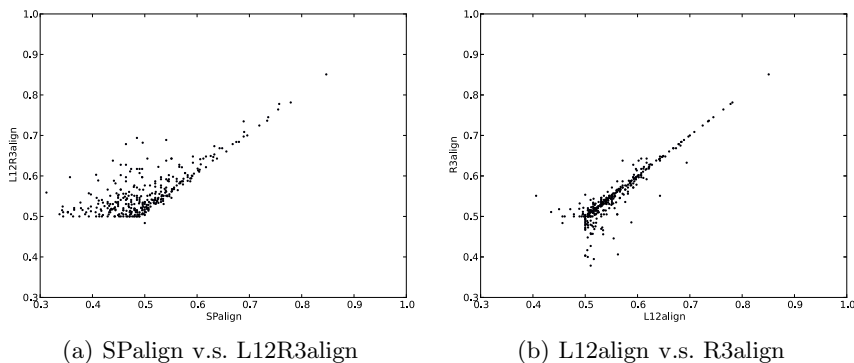


Fig. 3. Comparisons of the highest GDT scores found by SPalign and by using our method

Comparing the GDT scores found by L12R3align and SPalign as shown in Figure 3(a), we find that L12R3align consistently finds similar or higher GDT scores than SPalign. Specifically, L12R3align improves GDT scores by 0.06 on average and by 0.25 in the best case. It is seen that 25% of the GDT scores are improved by at least 0.09 and that 75% of the GDT scores are improved by at least 0.02. Moreover, SPalign finds 145 alignments with GDT scores higher than 0.5, while L12R3align finds 314 alignments with GDT scores higher than 0.5. Thus, 169 more alignments with GDT scores higher than 0.5 are discovered, with an average GDT score improvement of 0.09. These results again supports that

our protein structure alignment search algorithm can reliably find high quality alignments.

To further study the contributions of local fragments and remote fragment pairs to the GDT score improvements of L12R3align over SPalign, the GDT scores found by L12align and R3align are compared in Figure 3(b). It can be seen that both L12align and R3align find similar GDT scores when one of the GDT scores found by L12align and R3align is higher than 0.65. For the remaining protein structure pairs, both L12align and R3align are capable of discovering some better GDT scores than is the other method. Generally, 47% of the GDT scores found by L12align are up to 0.16 higher and 30% of the GDT scores found by R3align are up to 0.14 higher. Therefore, local fragments have a greater contribution in finding the highest GDT scores, while remote fragment pairs still have a significant contribution in finding the highest GDT scores.

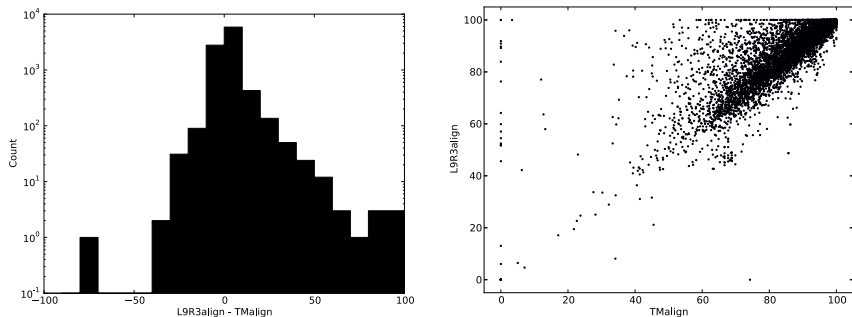
3.3 Scoring Function Evaluation on Consistency with Eye-Examined Alignments

In this experiment, we would like to show that our scoring function is capable of finding protein structure alignments that are significantly more consistent with alignments examined visually by human-beings. Thus, we used protein structure alignments from the HOMSTRAD database [21] as a benchmark and compared the alignment quality of our protein structure alignment with that of TMalign [13]. Here, the quality of the alignment is evaluated by the F-score, the harmonic mean of recall and precision, of aligned residue pairs.

The HOMSTRAD database has been widely used in protein research, including sequence-sequence alignment [32], sequence-structure alignment [33], and structure-structure alignment [34], among others. The database contains structure alignments of 3,454 homologous protein structures from 1,032 protein families [21]. Since different sequences were read from alignment files and from PDB structure files for some proteins, only 9,429 out of 9,535 protein structure alignments from HOMSTRAD were included in this experiment.

For our experiment settings, we chose $\lambda = 0.25$ and $w = 1.9$, empirically. Unlike previous experiment settings, we used local fragments of size 9 and remote fragment pairs of size 3. Such experiment settings are balanced between the accuracy and the speed of our protein structure alignment algorithm because only a minor improvement on accuracy is gained by increasing the sizes, while slowing down the running time. The local fragment size of 9 was previously shown to be the optimal balance between the complexity of the model and the amount of data required to train the model [35, 36]. Other experiment settings remained the same as in the previous experiment.

The F-score differences between L9R3align and TMalign are shown in Figure 4(a). Using L9R3align, 47% of the F-scores are improved, and the average F-score is improved from 88% to 90% compared to using TMalign. Moreover, there are 663 L9R3align F-scores that are at least 10% higher and there are 1,342 L9R3align F-scores that are at least 5% higher than the TMalign F-scores. For comparison, 31% of the TMalign F-scores are higher, and only 124 TMalign



(a) F-score difference between L9R3align and TMalign F-score (b) TMalign F-score v.s. L9R3align F-score

Fig. 4. Comparisons of the F-scores of the aligned residue pairs found by L9R3align and TMalign

F-scores are at least 10% higher. In total, TMalign finds 5,560 protein structure alignments with F-scores higher than 90%, while L9R3align finds 6,114 such alignments. Therefore, the protein structure alignments found by L9R3align are 10% more likely to be highly consistent (with F-score higher than 90%) with eye-examined alignments, and tend to have similar or higher F-scores compared to the protein structure alignments found by TMalign.

Among the 34 pairs of protein structures that have TMalign F-scores equal to zero as shown in Figure 4(b), the L9R3align F-scores reach 36% on average. Specifically, two L9R3align F-scores equal to 100% and 19 L9R3align F-scores are higher than 50%. For the two cases that L9R3align F-scores are equal to 100%, the aligned protein structures contain a high percentage of helices, and TMalign shifts the HOMSTRAD alignment by a few residues, which has also been previously observed [37]. Such shifting is difficult to avoid by evaluating only structure similarities. However, the shifting is avoided by our scoring function, involving both structure and sequence similarities, in this experiment. Therefore, sequence similarity does aid in finding high quality protein structure alignments that are highly consistent with eye-examined alignments, even if structure similarity itself fails to do so.

There is also one pair of protein structures in Figure 4(b) that the L9R3align F-score equals to zero, while the TMalign F-score equals to 74%. Here, the HOMSTRAD alignment can be represented by protein “AB-” aligning to protein “-CD”, where each character represents a protein fragment and “-” represents a gap region. One possible reason for this is that the weight parameters of our scoring function are not yet optimized to completely break the dependency between the alignment score and the protein size. We have observed that such cases can be eliminated by using different weight parameters, and this problem will be addressed in our future work.

4 Discussion and Conclusion

Therefore, our protein structure alignment method is not only reliable in finding the optimal alignment with the highest alignment score, but is also capable of discovering new alignments missed by current state-of-art alignment search algorithms and scoring functions. Our result verifies our assumption that there exists a near optimal ROTRAN that minimizes the RMSD of two small sets of C_α atoms. Our result also verifies that although structure similarity may be efficient in many cases, sequence similarity helps to find better protein structure alignments that are (possibly significantly) more consistent with eye-examined alignments. This is the result of incorporating both local fragments and remote fragment pairs in the alignment search algorithm, and of incorporating both structure similarity and sequence similarity in the scoring function.

Our protein structure alignment algorithm is still subject to improvement and application. Our scoring function remains capable of modeling more types of protein similarities, such as the (ϕ, ψ) dihedral angle distance and the secondary structure distance. Unknown protein domain length problems when aligning multi-domain proteins should also be addressed in the future as proposed by SPalign [30]. It should be interesting to allow flexible ROTRANs within the same cluster to find flexible structure alignments as seen in FATCAT [38] and to find flexible multi-structure alignments as seen in Matt [39]. Moreover, the alignment quality can be further studied by evaluating CASP protein structure prediction [31], by checking self-consistency [37], and by simulating the SCOP fold detection [26]. All these aid in fully automating protein structure alignment process as good as or even better than human experts in the short future.

Acknowledgments. This work was supported by the Startup Grant at City University of Hong Kong [7002731], the National Basic Research Program of China [2012CB316500], an NSERC Grant [OGP0046506], the Canada Research Chair program, an NSERC Collaborative Grant, OCRiT, the Premier's Discovery Award, the Killam Prize and SHARCNET.

References

1. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The protein data bank. *Nucleic Acids Res.* 28, 235–242 (2000)
2. Akutsu, T., Tashimo, H.: Protein structure comparison using representation by line segment sequences. In: *Pac. Symp. Biocomput.*, pp. 25–40 (1996)
3. Alexandrov, N.N.: SARFing the PDB. *Protein Eng.* 9(9), 727–732 (1996)
4. Caprara, A., Lancia, G.: Structural alignment of large-size proteins via lagrangian relaxation. In: *RECOMB 2002: Proceedings of the Sixth Annual International Conference on Computational Biology*, pp. 100–108. ACM, New York (2002)
5. Comin, M., Guerra, C., Zanotti, G.: Proust: a comparison method of three-dimensional structure of proteins using indexing techniques. *Journal of Computational Biology* 11, 1061–1072 (2004)

6. Gerstein, M., Levitt, M.: Using iterative dynamic programming to obtain accurate pairwise and multiple alignments of protein structures. In: Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, pp. 59–67. AAAI Press (1996)
7. Gibrat, J.F., Madej, T., Bryant, S.H.: Surprising similarities in structure comparison. *Current Opinion in Structural Biology* 6(3), 377–385 (1996)
8. Lancia, G., Carr, R., Walenz, B., Istrail, S.: 101 optimal pdb structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. In: RECOMB 2001: Proceedings of the Fifth Annual International Conference on Computational Biology, pp. 193–202. ACM, New York (2001)
9. Singh, A.P., Brutlag, D.L.: Hierarchical protein structure superposition using both secondary structure and atomic representations. In: Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology, pp. 284–293. AAAI Press (1997)
10. Subbiah, S., Laurents, D.V., Levitt, M.: Structural similarity of DNA-binding domains of bacteriophage repressors and the globin core. *Current Biology* 3(3), 141–148 (1993)
11. Shindyalov, I.N., Bourne, P.E.: Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering* 11(9), 739–747 (1998)
12. Xie, L., Bourne, P.E.: Detecting evolutionary relationships across existing fold space, using sequence order-independent profile–profile alignments. *PNAS* 8(4), 5441–5446 (2008)
13. Zhang, Y., Skolnick, J.: Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic Acids Research* 33(7), 2302–2309 (2005)
14. Pandit, S.B., Skolnick, J.: Fr-tm-align: a new protein structural alignment method based on fragment alignments and the tm-score. *BMC Bioinformatics* 9(1), 531 (2008)
15. Zhang, Y., Skolnick, J.: Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics* 57(4), 702–710 (2004)
16. Zemla, A., Venclovas, Č., Moulton, J., Fidelis, K.: Processing and analysis of casp3 protein structure predictions. *Proteins: Structure, Function, and Bioinformatics* 37(S3), 22–29 (1999)
17. Levitt, M., Gerstein, M.: A unified statistical framework for sequence comparison and structure comparison. *Proceedings of the National Academy of sciences* 95(11), 5913–5920 (1998)
18. Pirovano, W., Feenstra, K.A., Heringa, J.: The meaning of alignment: lessons from structural diversity. *BMC Bioinformatics* 9(1), 556 (2008)
19. Daniels, N.M., Nadimpalli, S., Cowen, L.J., et al.: Formatt: Correcting protein multiple structural alignments by incorporating sequence alignment. *BMC Bioinformatics* 13(1), 1–8 (2012)
20. Wang, S., Ma, J., Peng, J., Xu, J.: Protein structure alignment beyond spatial proximity. *Scientific Reports* 3 (2013)
21. Mizuguchi, K., Deane, C.M., Blundell, T.L., Overington, J.P.: Homstrad: a database of protein structure alignments for homologous families. *Protein Science* 7(11), 2469–2471 (1998)
22. Zhang, Y., Skolnick, J.: Spicker: A clustering approach to identify near-native protein folds. *Journal of Computational Chemistry* 25(6), 865–871 (2004)

23. Balcan, M.F., Blum, A., Gupta, A.: Approximate clustering without the approximation. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, pp. 1068–1077 (2009)
24. Moakher, M.: Means and averaging in the group of rotations. *SIAM Journal on Matrix Analysis and Applications* 24(1), 1–16 (2002)
25. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48(3), 443–453 (1970)
26. Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C.: Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology* 247(4), 536–540 (1995)
27. Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences* 89(22), 10915–10919 (1992)
28. Eddy, S.R., et al.: Where did the blosum62 alignment score matrix come from? *Nature Biotechnology* 22(8), 1035–1036 (2004)
29. Xu, J., Zhang, Y.: How significant is a protein structure similarity with tm-score= 0.5? *Bioinformatics* 26(7), 889–895 (2010)
30. Yang, Y., Zhan, J., Zhao, H., Zhou, Y.: A new size-independent score for pairwise protein structure alignment and its application to structure classification and nucleic-acid binding prediction. *Proteins: Structure, Function, and Bioinformatics* 80(8), 2080–2088 (2012)
31. Kinch, L., Yong Shi, S., Cong, Q., Cheng, H., Liao, Y., Grishin, N.V.: Casp9 assessment of free modeling target predictions. *Proteins: Structure, Function, and Bioinformatics* 79(S10), 59–73 (2011)
32. Do, C.B., Mahabhashyam, M.S., Brudno, M., Batzoglou, S.: Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome Research* 15(2), 330–340 (2005)
33. Shi, J., Blundell, T.L., Mizuguchi, K.: Fugue: sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology* 310(1), 243–257 (2001)
34. Konagurthu, A.S., Whisstock, J.C., Stuckey, P.J., Lesk, A.M.: Mustang: a multiple structural alignment algorithm. *Proteins: Structure, Function, and Bioinformatics* 64(3), 559–574 (2006)
35. Rohl, C.A., Strauss, C.E., Misura, K., Baker, D.: Protein structure prediction using rosetta. *Methods in Enzymology* 383, 66–93 (2004)
36. Maadooliat, M., Gao, X., Huang, J.Z.: Assessing protein conformational sampling methods based on bivariate lag-distributions of backbone angles. *Brief. Bioinform.* (2012)
37. Sadowski, M., Taylor, W.: Evolutionary inaccuracy of pairwise structural alignments. *Bioinformatics* 28(9), 1209–1215 (2012)
38. Ye, Y., Godzik, A.: Flexible structure alignment by chaining aligned fragment pairs allowing twists. *Bioinformatics* 19(suppl. 2), ii246–ii255 (2003)
39. Menke, M., Berger, B., Cowen, L.: Matt: local flexibility aids protein multiple structure alignment. *PLoS Computational Biology* 4(1), e10 (2008)

A Minimum-Labeling Approach for Reconstructing Protein Networks across Multiple Conditions

Arnon Mazza¹, Irit Gat-Viks², Hesso Farhan³, and Roded Sharan¹

¹ Blavatnik School of Computer Science,
Tel Aviv University, Tel Aviv 69978, Israel
`roded@post.tau.ac.il`

² Dept. of Cell Research and Immunology,
Tel Aviv University, Tel Aviv 69978, Israel

³ Biotechnology Institute Thurgau, University of Konstanz,
Unterseestrasse 47, 8280 Kreuzlingen, Switzerland

Abstract. The sheer amounts of biological data that are generated in recent years have driven the development of network analysis tools to facilitate the interpretation and representation of these data. A fundamental challenge in this domain is the reconstruction of a protein-protein subnetwork that underlies a process of interest from a genome-wide screen of associated genes. Despite intense work in this area, current algorithmic approaches are largely limited to analyzing a single screen and are, thus, unable to account for information on condition-specific genes, or reveal the dynamics (over time or condition) of the process in question. Here we propose a novel formulation for network reconstruction from multiple-condition data and devise an efficient integer program solution for it. We apply our algorithm to analyze the response to influenza infection in humans over time as well as to analyze a pair of ER export related screens in humans. By comparing to an extant, single-condition tool we demonstrate the power of our new approach in integrating data from multiple conditions in a compact and coherent manner, capturing the dynamics of the underlying processes.

1 Introduction

With the increasing availability of high-throughput data, network biology has become the method of choice for filtering, interpreting and representing these data. A fundamental problem in network biology is the reconstruction of a subnetwork that underlies a process of interest by efficiently connecting a set of implicated proteins (derived by some genome-wide screen) in a network of physical interactions. In recent years, several algorithms have been suggested for different variants of this problem, including the Steiner tree based methods of [1,2], the flow based approach of [3] and the anchored reconstruction method of [4].

Despite the plethora of network reconstruction methods, these have been so far largely limited to explaining a single experiment or condition. In practice, the

network dynamically changes over time or conditions, calling for reconstructions that can integrate such data to a coherent picture of the activity dynamics of the underlying pathways.

Here we tackle this multiple-condition scenario, where the reconstructed subnetwork should explain in a coherent manner multiple experiments driven by the same set of proteins (referred to here as *anchor* proteins) while producing different subsets of affected proteins, or *terminals*. As in the single-condition case, a parsimonious assumption implies that the reconstructed subnetwork should be of minimum size. In addition, we require that its pathways, leading from the anchor to each of the terminals, are as homogeneous as possible in terms of the conditions, or *labels* they span. We formulate the resulting minimum labeling problem, show that it is NP complete and characterize its solutions. We then offer an equivalent formulation that allows us to design a polynomial integer linear programming (ILP) formulation for its solution. We implement the ILP algorithm, *MKL*, and apply it to two data sets in humans concerning the response to influenza infection and ER export regulation. We show that the MKL networks are significantly enriched with respect to the related biological processes and allow obtaining of novel insights on the modeled processes. We further compare MKL with an extant method, ANAT [4], demonstrating the power of our algorithm in integrating data from multiple conditions in a compact and informative manner. For lack of space, some algorithmic details are omitted or deferred to an Appendix.

2 Preliminaries

Let $G = (V, E)$ be a directed graph, representing a protein-protein interaction (PPI) network, with vertex set V and edge set E . For a node $v \in V$, denote by $In(v)$ ($Out(v)$) the set of incoming (outgoing) edges of v , respectively. Let $L = \{1, \dots, k\}$ be a set of labels, representing $k \geq 1$ conditions. Let $f : E \rightarrow 2^{(L)}$ be a labeling function that assigns each edge of E a possibly empty subset of labels. For $1 \leq i \leq k$, we define $E_i(f) := \{e \in E : i \in f(e)\}$ to be the set of edges with label i . We further denote $f_{in}(v) = \bigcup_{e \in In(v)} f(e)$ and $f_{out}(v) = \bigcup_{e \in Out(v)} f(e)$.

We say that a labeling function f is *valid* if for every terminal t and each condition i in which it is affected, there is a path from the anchor to the terminal whose edges are assigned with the label i . Formally, we require a path from a to t that is restricted to $E_i(f)$. We evaluate the *cost* of the labeling according to the number of labels $L(f)$ used and the number of edges $N(f)$ that are assigned with at least one label. Formally, $L(f) = \sum_{e \in E} |f(e)|$ and $N(f) = |\{e \in E : f(e) \neq \emptyset\}|$. The cost is then defined as $\alpha \cdot L(f) + (1 - \alpha) \cdot N(f)$, where $0 \leq \alpha \leq 1$ balances the two terms.

We study the following **minimum k -labeling (MKL)** problem on G : The input is an anchor node $a \in V$ and $k \geq 1$ sets of terminals T_1, \dots, T_k in $V \setminus \{a\}$ that implicitly assign to each terminal the subset of conditions, or labels in which it is affected. The objective is to find a valid labeling of the edges of G of minimum cost.

Clearly, any valid labeling induces a subnetwork that can model each of the experiments: this subnetwork is comprised of those edges that are assigned a non-empty subset of labels. We note that for $k = 1$ we have $L(f) = N(f)$, thus in this case the MKL problem is equivalent to the minimum directed Steiner tree problem. The parameter α balances between two types of solutions: (1) A subnetwork with minimum number of labels ($\alpha = 1$), which is equivalent to the union of independent Steiner trees of each of the experiments. (2) A subnetwork with minimum number of edges ($\alpha = 0$), which is simply a Steiner tree that spans the union of all sets of terminals. However, general instances of MKL where $\alpha \neq 0, 1$ can be solved neither by combining the independent Steiner trees of each of the experiments nor by constructing a single Steiner tree over all terminals. This is illustrated by the toy examples in Figures S5 and S6. Next, we provide a characterization of solutions to the MKL problem.

Theorem 1. *Given a solution labeling f to an MKL instance, let G_i denote the subgraph of G that is induced by the edges in $E_i(f)$. Then G_i is a directed tree rooted at a .*

Proof. By definition, there is a directed path in G_i from a to each of the terminals in T_i . Clearly, any edge directed into a can be removed without affecting the constraints of a valid solution. Thus, it suffices to show that the underlying undirected graph of G_i contains no cycles. By minimality of the solution, every vertex in G_i is reachable from a or else it can be removed along with its edges. Suppose to the contrary that v_1, \dots, v_n is a cycle in the underlying graph. Since a cannot be on this cycle and by the above observation, each of the cycle's vertices is reachable from a . W.l.o.g., let v_1 be the farthest from a in G_i among all cycle vertices. Then one can obtain a smaller solution by removing one of the edges (v_1, v_2) , (v_n, v_1) (depending on their orientations), a contradiction.

As noted earlier, when $k = 1$ the MKL problem is equivalent to the minimum directed Steiner tree problem, which is known to be NP-complete [5]. A simple reduction from this case yields the following result:

Theorem 2. *The MKL problem is NP-complete for every $k \geq 1$.*

3 The MKL Algorithm

As the MKL problem is NP-complete, we aim to design an integer linear program (ILP) for it, which will allow us to solve it to optimality or near-optimality for moderately-sized instances. In order to design an efficient ILP, we first provide an alternative formulation of the MKL problem, expressed in terms of units of flow per label pushed from the anchor toward the terminals. To this end, we extend the labeling to assign multi-sets rather than sets. We denote a multi-set by a pair $M = \langle S, \mu \rangle$, where S is a set and $\mu: S \rightarrow \mathbb{Z}^+$. We say that $x \in M$ if $x \in S$. We let $|M|$ denote the cardinality of the underlying set S .

The union \uplus of two multi-sets $\langle S_1, \mu_1 \rangle, \langle S_2, \mu_2 \rangle$ is defined as the pair $\langle S, \mu \rangle$, where $S = S_1 \cup S_2$; for every $x \in S_1 \cap S_2$, $\mu(x) = \mu_1(x) + \mu_2(x)$; for $x \in S_1 \setminus S_2$, $\mu(x) = \mu_1(x)$; and for $x \in S_2 \setminus S_1$, $\mu(x) = \mu_2(x)$. We extend the definitions of $f_{in}(v)$ and $f_{out}(v)$ to multi-sets using this union operator. Finally, for a vertex $v \neq a$ we let $L(v) = \{i \in L : v \in T_i\}$; note that for non-terminal nodes $L(v) = \emptyset$.

The alternative objective formulation is as follows: Find a multi-set label assignment g that satisfies the following constraints:

- (i) $g_{out}(a) = \langle L, \mu \rangle$, where $\mu(i) = |T_i|$ for every $i \in L$.

(The total amount of flow that goes out from the anchor per label equals the number of terminals that belong to the corresponding experiment).

- (ii) For every $v \neq a$, $g_{in}(v) = g_{out}(v) \uplus L(v)$.

(For each label i , the incoming flow of a node v equals its outgoing flow, incremented by 1 if v is a terminal expressed in experiment i).

- (iii) Denote $L(g) = \sum_{e \in E} |g(e)|$, $N(g) = |\{e \in E : g(e) \neq \emptyset\}|$, and let $0 \leq \alpha \leq 1$. Then $\alpha \cdot L(g) + (1 - \alpha) \cdot N(g)$ is minimal.

We claim that the two formulations are equivalent. Given a multi-set labeling g , it is easy to transform it into a labeling f by taking at each edge the underlying set of labels. One can show that the labeling f is valid, i.e. for each i there are paths in $E_i(f)$ that connect a to each of the terminals in T_i . For the other direction, given a labeling f we can transform it into a multi-set labeling g by defining the multiplicity of a label i at the edge $(u, v) \in E_i(f)$ as the number of terminals from T_i in the subtree of G_i that is rooted at v . It is easy to see that all constraints are satisfied by this transformation.

The above problem formulation can be made stricter by requiring that the set of incoming labels to a terminal is exactly the set of labels associated with the terminal. That is, for every terminal t and $i \in L \setminus L(t)$, we require that $i \notin g_{in}(t)$. Our ILP formulation includes these requirements to reflect the experimental observations, but in practice the strict and non-strict versions produce very similar results.

3.1 An ILP Formulation

In order to formulate the problem as an integer program, we define three sets of variables: (i) binary variables of the form y_e^i , indicating for every $e \in E$ and $i \in L$ whether the edge e is tagged with label i ; (ii) integer variables of the form x_e^i , indicating for every $e \in E$ and $i \in L$ the multiplicity of label i (in the range of 0 to $|T_i|$); and (iii) binary variables of the form z_e , indicating for every $e \in E$ whether the edge e participates in the subnetwork (carrying any label). For a vertex $v \in V$, let b_v^i be a binary indicator of whether $i \in L(v)$ or not. Let α be some fixed value in the range $[0, 1]$. The formulation is as follows (omitting the constraints on variable ranges):

$$\begin{aligned}
& \min \alpha \cdot \sum_{e \in E, i \in L} y_e^i + (1 - \alpha) \cdot \sum_{e \in E} z_e \\
& \text{s.t.}: \\
& \quad y_e^i \leq x_e^i \leq |T_i| \cdot y_e^i \quad \forall e \in E, i \in L \\
& \quad y_e^i \leq z_e \quad \forall e \in E, i \in L \\
& \quad \sum_{e \in \text{Out}(a)} x_e^i = |T_i| \quad \forall i \in L \\
& \quad \sum_{e \in \text{In}(v)} x_e^i = \sum_{e \in \text{Out}(v)} x_e^i + b_v^i \quad \forall v \in V \setminus \{a\}, i \in L \\
& \quad \sum_{e \in \text{In}(t)} y_e^i = 0 \quad \forall t \in T, i \notin L(t)
\end{aligned}$$

3.2 Implementation Details and Performance Evaluation

We used the commercial IBM ILOG CPLEX optimizer to solve the above ILP. Since solving an ILP is time consuming, we devised a heuristic method for filtering the input network, aiming to capture those edges that the MKL optimal solution is more likely to use. Specifically, we focused on (directed) edges that lie on a near shortest path (up to one edge longer than a shortest path) between the anchor and any of the terminals. Further, we accepted approximate solutions which enabled our experiments to end within at most two hours. Restricted by this time frame, we attained solutions deviating by at most 5% and 7% from the optimal value for the influenza dataset and the ER export dataset (see Experimental Results Section), respectively.

We tested the robustness of MKL to different choices of α on the two datasets we analyzed, observing that the number of edges and labels varied by at most 8% and 4%, respectively, over a wide range of values (0.25 – 0.75). Thus, we chose $\alpha = 0.5$ for our analyses in the sequel.

We evaluated a solution subnetwork using both network-based and biological measures. The network-based measures included the cost and a *homogeneity* score. We defined the homogeneity of a node v as the frequency of the most frequent subset of labels among the $t(v)$ terminals under v , divided by $t(v)$; the homogeneity score of the subnetwork was then defined as the average over all nodes that span at least two terminals. To quantify the biological significance of the reconstructed subnetworks, we measured the functional enrichment of their internal nodes (non-input nodes) with respect to validation sets that pertain to the process in question. In addition, we provide expert analysis of the subnetworks.

We compared the performance of our method to that of the state-of-the-art ANAT reconstruction tool [4], which was shown to outperform many existing tools in anchored reconstruction scenarios. For each data set, we applied ANAT (with its default parameters, and without the heuristic filtering) to each condition separately, then unified the results to get an integrated subnetwork. We labeled the solution straightforwardly: an edge e was labeled i if e participated in the subnetwork that was constructed for condition i .

4 Experimental Results

We tested the performance of our algorithm on two human data sets that concern the cellular response to PR8 influenza virus and ER export regulation. The two data sets were analyzed in the context of a human PPI network reported in [4] which contains 44,738 (bidirectional) interactions over 10,169 proteins. We compared our results to those of a previous tool, ANAT [4], applying it independently to each of the terminal sets and taking the union of the subnetworks as the result. We describe these applications below.

4.1 Response to Influenza Infection

We used data on the response to viral infection by the H1N1 influenza strain A/PR/8/34 ('PR8') in primary human bronchial epithelial cells [6]. The data set contains a collection of 135 virus-human PPIs and gene expression profiles, measured at different time points along the course of the infection. We focused on four time points (the "conditions") $t = 2, 4, 6, 8$ (i.e. $k = 4$ labels), in each time point selecting those genes that were differentially expressed above a cutoff of 0.67 [6]. We did not include time points earlier than $t = 2$ or later than $t = 8$, as the former had no or very few differentially expressed genes, while the latter induced an order of magnitude larger gene sets that are presumably associated with secondary responses.

We augmented the human network by the influenza-host PPIs and an auxiliary anchor node (named 'virus') which we connected to the 10 viral proteins. After the filtering, the network contained 1,598 proteins and 8,708 interactions. The four terminal sets contained 8,19,19 and 49 proteins, respectively, with 77 total in their union, out of which 57 were reachable from the anchor. The resulting MKL subnetwork, which is shown in Figure 1, contains 127 edges over 123 nodes (117 human, 5 viral and the anchor node) with 60 internal (non-input) nodes. This is in contrast to the much larger ANAT solution on this data set, containing 173 nodes and 106 internal ones. The subnetworks are quite different in terms of node composition, having 31 internal intersecting nodes. A summary of our network-based measures for the two subnetworks can be found in Figure 2.

Next, we scored the enrichment of both subnetworks with viral infection related processes such as: viral reproduction, intracellular receptor mediated signaling pathway and apoptosis. The MKL subnetwork was highly enriched with these processes, outperforming the ANAT subnetwork (Figure 3). In the following we present a detailed analysis of the MKL inferred subnetwork and demonstrate its high predictive power and its ability to characterize viral proteins and host mediators in terms of their temporal effect on their targets. Specifically, we show that this subnetwork suggests that an imbalance in the timing of effect between viral proteins (e.g. M1 and NP) or between host mediators (such as Smad3 and UBC) can reveal their different kinetics of influence on host proteins. This is in large contrast to the results produced by the ANAT tool, which does not provide any timing imbalance among downstream targets of viral proteins or host mediators (data not shown due to space constraints).

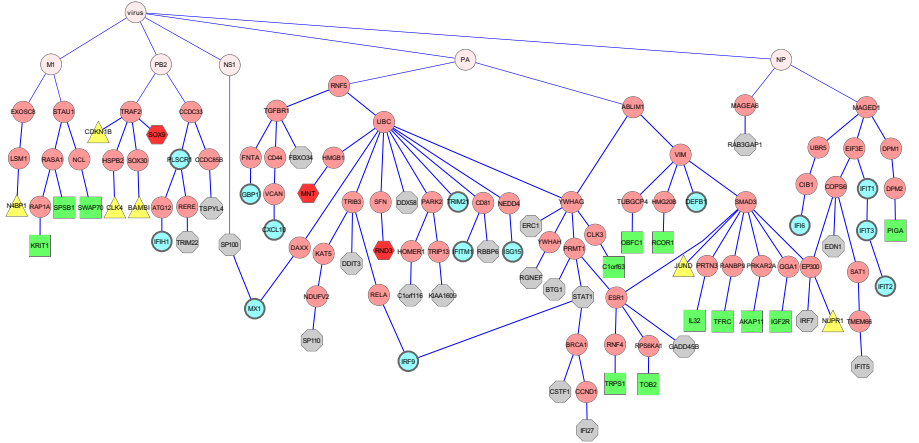


Fig. 1. The MKL subnetwork for the influenza infection data. Terminal nodes are marked by their corresponding time point: $t=2$ - yellow/triangle; $t=4$ - green/square; $t=6$ - red/hexagon; $t=8$ - gray/octagon; more than one time point - cyan oval nodes with thick border. The root is the artificial virus node and the first level is composed solely of viral proteins.

	Measure	MKL	ANAT
Influenza infection	no. of labels	171	277
	no. of edges	127	195
	cost	149	236
	homogeneity score	0.63	0.58
ER export	no. of labels	152	213
	no. of edges	145	203
	cost	148.5	208
	homogeneity score	0.88	0.74

Fig. 2. Summary statistics for the MKL and ANAT subnetworks on the viral infection and ER export data sets

	Biological process	MKL	ANAT
Influenza infection	intracellular receptor mediated signaling pathway (GO:0030522)	6.5E-10	2.1E-04
	apoptosis (GO:0006915)	3.7E-04	1.7E-04
	viral reproduction (GO:0016032)	2.5E-03	>0.05
ER export	vesicle-mediated transport (GO:0016192)	2.7E-10	2.4E-08
	cellular membrane organization (GO:0016044)	5.5E-10	2.7E-09
	intracellular protein transport (GO:0006886)	3.7E-07	1.7E-07

Fig. 3. Comparison of enrichments of the MKL and ANAT solutions with respect to influenza infection and ER export related processes

We first present an example of an inferred pathway, selected to demonstrate our MKL approach. The PA-Rnf5-UBC-DAXX-MX1 and NS1-SP100-MX1 paths are a clear example of a predicted pathway that is well supported by extant experimental findings. It is consistent with the known role of both DAXX and SP100 as major components of the PML bodies which control together the localization of MX1 in distinct nuclear components [7]. Further, DAXX is known to be regulated *in vivo* by ubiquitination through UBC and Rnf5 [8], supporting our placement of DAXX downstream to UBC.

The MKL network shows that the targets of some human proteins have a common temporal behavior, whereas others have different downstream temporal responses. This is consistent with the fact that PPIs naturally represent different mechanisms that might differ in their kinetics. For example, the targets of Traf2 are all early responding genes whereas the targets of Ccdc33 have longer temporal responses. The early effect of Traf2 is consistent with the findings that Traf2 is a signaling transduction kinase protein with fast kinetics. A similar characterization can be applied to other signal transduction proteins such as Smad3. Conversely, the Ccdc33 protein regulates its targets in late time points (6-8 hours) by an unknown mechanism. The results here suggest that this mechanism is orders of magnitude slower than phosphorylation. Similarly, the control of Rnf5 and UBC is expected to show fast kinetics through ubiquitination. In contrast, we find that all the Rnf5/UBC 19 targets are controlled in late time points (6-8 hours), suggesting a novel temporal (late) control on the activity of Rnf5-specific UBC-based ubiquitination during the course of influenza infection.

4.2 Regulation of Endoplasmic Reticulum (ER) Export

The journey of secretory proteins, which make up roughly 30% of the human proteome starts by exit from the ER. Export from the ER is executed by so called COPII vesicles that bud from ER exit sites (ERES). A protein that is of central importance for ERES biogenesis and maintenance is Sec16A, a large (~250 kDa) protein that localizes to ERES and interacts with COPII components [9]. We have recently performed a siRNA screen to test for kinases and phosphatases that regulate the functional organization of the early secretory pathway [10]. Among the hits identified were 64 kinases/phosphatases that when depleted result in a reduction in the number of ERES. Thus, these are 64 different potential regulators of ER export. More recently, a full genome screen tested for genes that regulate the arrival of a reporter protein from the ER to the cell surface [11]. There, the depletion of 45 proteins was shown to affect ERES. However, whether the defect in arrival of the reporter to the cell surface was due to an effect on ER export or due to alterations in other organelles along the secretory route (e.g., Golgi apparatus) remains to be determined.

We applied MKL to these two screens, serving as two “conditions” highlighting different repertoires of ER export signaling-regulatory pathways. As the two screens do not intersect (most likely due to differences in read-outs), there were 109 terminals overall, 85 of them reachable in our human PPI network.

Due to its central importance for ER export and ERES formation, we chose Sec16A as the anchor for this application. After filtering, the network contained 1,907 nodes and 11,329 edges. The resulting MKL subnetwork, containing 145 nodes and 59 internal ones, is depicted in Figure 4. In comparison, the ANAT solution contains 190 nodes and 104 internal ones (with 35 internal nodes common to the two solutions). As evident from Figure 2, the MKL solution has substantially lower cost and is more homogeneous.

We assessed the functional enrichment of the MKL subnetwork with biological processes that are of relevance to ER export such as cellular membrane organization, intracellular protein transport and vesicle-mediated transport. All three categories were highly enriched and the p -values attained compare favorably to those computed for the ANAT solution (Figure 3).

Interestingly, 4 proteins of the MKL solution are related to autophagy (two of them internal nodes, $p = 0.02$). Autophagy is an endomembrane-based cellular process that is responsible for capturing and degradation of surplus organelles and proteins. Links between ER export and autophagy have been proposed [12] but there is very limited mechanistic insight into this link. The vesicle-mediated transport process includes the STX17, SNAP29 and ULK1 proteins. The latter is a kinase that initiates the biogenesis of autophagosomes [13]. STX17 and SNAP29 were recently proposed to be involved in autophagy by promoting the formation of ER-mitochondria contact sites and the fusion of autophagosomes with lysosomes [14,15]. As the MKL network was generated with terminals and an anchor that regulate ER export, we propose that this approach could be used to identify the molecular link between secretion and autophagy in the future.

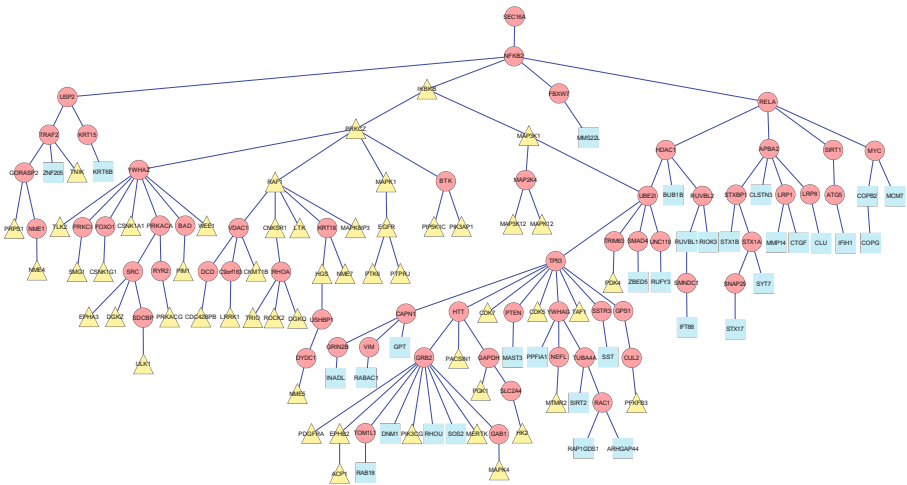


Fig. 4. The MKL subnetwork for the ER export data. Terminal nodes are colored/shaped according to the screen they were discovered in: [10] - yellow/triangle, and [11] - cyan/square.

5 Conclusions

The protein-protein interaction network represents a combination of diverse regulation and interaction mechanisms operating in different conditions and time scales. Integrating such data in a coherent manner to describe a process of interest is a fundamental challenge, which we aim to tackle in this work via a novel ILP-based minimum labeling algorithm. We apply our algorithm to two human data sets and show that it attains compact solutions that capture the dynamics of the data and align well with current knowledge. We expect this type of analysis to gain further momentum as composite data sets spanning multiple conditions and time points continue to accumulate.

Acknowledgments. AM was supported in part by a fellowship from the Edmond J. Safra Center for Bioinformatics at Tel Aviv University. RS was supported by a research grant from the Israel Science Foundation (grant no. 241/11).

References

1. Beisser, D., Klau, G., Dandekar, T., Mueller, T., Dittrich, M.: BioNet an R-package for the functional analysis of biological networks. *Bioinformatics* 26, 1129–1130 (2010)
2. Huang, S., Fraenkel, E.: Integrating proteomic, transcriptional, and interactome data reveals hidden components of signaling and regulatory networks. *Sci. Signal.* 2(81), ra40 (2009)
3. Lotem, E., Riva, L., Su, L., Gitler, A., Cashikar, A., King, O., Auluck, P., Geddie, M., Valastyan, J., Karger, D., Lindquist, S., Fraenkel, E.: Bridging high-throughput genetic and transcriptional data reveals cellular responses to alpha-synuclein toxicity. *Nature Genetics* 41, 316–323 (2009)
4. Yosef, N., Zalckvar, E., Rubinstein, A., Homilius, M., Atias, N., Vardi, L., Berman, I., Zur, H., Kimchi, A., Ruppin, E., Sharan, R.: ANAT: A tool for constructing and analyzing functional protein networks. *Sci. Signal.* 4 (2011)
5. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co. (1979)
6. Shapira, S., Gat-Viks, I., Shum, B., Dricot, A., Degrace, M., Liguio, W., Gupta, P., Hao, T., Silver, S., Root, D., Hill, D., Regev, A., Hacohen, N.: A physical and regulatory map of host-influenza interactions reveals pathways in H1N1 infection. *Cell* 139(7), 1255–1267 (2009)
7. Engelhardt, O., Sirma, H., Pandolfi, P., Haller, O.: Mx1 GTPase accumulates in distinct nuclear domains and inhibits influenza A virus in cells that lack promyelocytic leukaemia protein nuclear bodies. *J. Gen. Virol.* 85(8), 2315–2326 (2004)
8. Wagner, S., Beli, P., Weinert, B., Nielsen, M., Cox, J., Mann, M., Choudhary, C.: A proteome-wide, quantitative survey of in vivo ubiquitylation sites reveals widespread regulatory roles. *Mol. Cell. Proteomics* 10(10) (2011)
9. Watson, P., Townley, A., Koka, P., Palmer, K., Stephens, D.: Sec16 defines endoplasmic reticulum exit sites and is required for secretory cargo export in mammalian cells. *Traffic* 7(12), 1678–1687 (2006)

10. Farhan, H., Wendeler, M., Mitrovic, S., Fava, E., Silberberg, Y., Sharan, R., Zerial, M., Hauri, H.: MAPK signaling to the early secretory pathway revealed by kinase/phosphatase functional screening. *J. Cell. Biol.* 189, 997–1011 (2010)
11. Simpson, J., Joggerst, B., Laketa, V., Verissimo, F., Cetin, C., Erfle, H., Bexiga, M., Singan, V., Hériché, J., Neumann, B., Mateos, A., Blake, J., Bechtel, S., Benes, V., Wiemann, S., Ellenberg, J., Pepperkok, R.: Genome-wide RNAi screening identifies human proteins with a regulatory function in the early secretory pathway. *Nat. Cell Biol.* 14(7), 764–774 (2012)
12. Ishihara, N., Hamasaki, M., Yokota, S., Suzuki, K., Kamada, Y., Kihara, A., Yoshimori, T., Noda, T., Ohsumi, Y.: Autophagosome requires specific early Sec proteins for its formation and NSF/SNARE for vacuolar fusion. *Mol. Biol. Cell.* 12(11), 3690–3702 (2001)
13. Mizushima, N.: The role of the Atg1/ULK1 complex in autophagy regulation. *Curr. Opin. Cell Biol.* 22(2), 132–139 (2010)
14. Hamasaki, M., Furuta, N., Matsuda, A., Nezu, A., Yamamoto, A., Fujita, N., Oomori, H., Noda, T., Haraguchi, T., Hiraoka, Y., Amano, A., Yoshimori, T.: Autophagosomes form at ER-mitochondria contact sites. *Nature* 495(7441), 389–393 (2013)
15. Itakura, E., Mizushima, N.: Syntaxin 17: The autophagosomal SNARE. *Autophagy* 9(6) (2013)

A Supplementary Figures

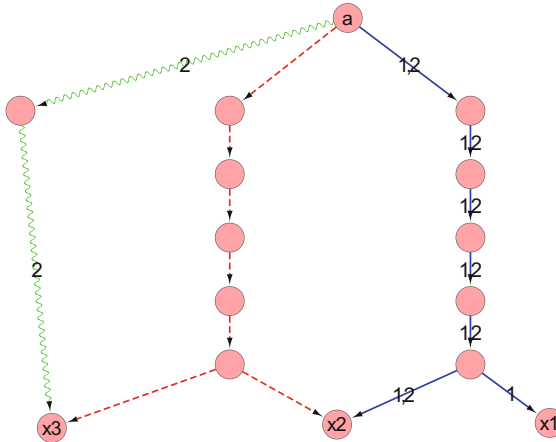


Fig. S5. The optimal MKL solution for $\alpha = 0.5$ is neither the union of label-specific Steiner trees nor a subgraph of it. In this instance $k = 2$, $T_1 = \{x_1, x_2\}$ and $T_2 = \{x_2, x_3\}$. The optimal Steiner trees for T_1 and T_2 are composed of the blue (solid) and red (dashed) edges, resp. The best MKL solution that uses only edges of the union can be achieved by pushing label 1 over the blue edges and 2 over the red edges, resulting in 14 labels and 14 edges. In contrast, the optimal solution, whose labels appear on top of the figure, contains the blue and green (waved) edges, spanning 15 labels and 9 edges.

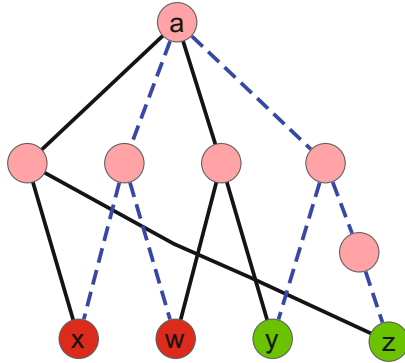


Fig. S6. The optimal MKL solution for $\alpha = 0.6$ is not a minimum Steiner tree over all terminals. In this instance $k = 2$, $T_1 = \{x, w\}$ and $T_2 = \{y, z\}$. The black (solid) edges form a Steiner tree with 6 edges and 8 labels, whereas the blue (dashed) edges constitute an MKL solution with 7 edges and 7 labels.

Faster Mass Decomposition

Kai Dührkop, Marcus Ludwig, Marvin Meusel, and Sebastian Böcker

Chair for Bioinformatics, Friedrich Schiller University, Jena, Germany
sebastian.boecker@uni-jena.de

Abstract. Metabolomics complements investigation of the genome, transcriptome, and proteome of an organism. Today, the vast majority of metabolites remain unknown, in particular for non-model organisms. Mass spectrometry is one of the predominant techniques for analyzing small molecules such as metabolites. A fundamental step for identifying a small molecule is to determine its molecular formula.

Here, we present and evaluate three algorithm engineering techniques that speed up the molecular formula determination. For that, we modify an existing algorithm for decomposing the monoisotopic mass of a molecule. These techniques lead to a four-fold reduction of running times, and reduce memory consumption by up to 94%. In comparison to the classical search tree algorithm, our algorithm reaches a 1000-fold speedup.

1 Introduction

Metabolomics complements investigation of the genome, transcriptome, and proteome of an organism [14]. Today, the vast majority of metabolites remain unknown, and this is particularly the case for non-model organisms and secondary metabolites: for many organisms, there is a striking discrepancy between the number of identified metabolites and the prediction of secondary metabolite-related biosynthetic pathways through recent genome-sequencing results, see for example [8]. The structural diversity of metabolites is extraordinarily large, and much larger than for biopolymers such as proteins. In almost all cases, we cannot deduce the structure of metabolites from genome sequences, as it is done with proteins. Mass spectrometry (MS) is one of the two predominant experimental analysis techniques for detecting and identifying metabolites and other small molecules, the other being nuclear magnetic resonance (NMR). The most important advantage of MS over NMR is that it is orders of magnitude more sensitive, making it the method of choice for medium- to high-throughput screening applications [14]. Newly identified metabolites often serve as leads in drug design [15], in particular for novel antibiotics [7].

In a mass spectrometry experiment, we measure the mass-to-charge ratios (m/z) of a peak, corresponding to an ion of the intact molecule or its fragments. Here, we omit the analysis of the charge state and assume that the mass m of the ion is known: In most cases, small molecules receive only a single charge, and other cases can be either detected by analyzing the isotope pattern on the metabolite, or simply by iterating over the few possible charge states.

One of the most basic — but nevertheless highly important — steps when analyzing a molecule, is to determine its molecular formula. We note in passing that mass spectrometry does not record the mass of the uncharged molecule but rather the mass of the corresponding ion; for the sake of clarity, we ignore this difference in the following. Common approaches first compute all candidate molecules with mass sufficiently close to a peak mass in the measured spectrum, using an alphabet of potential elements. In a second step, additional information is used to score the different candidate molecules, for example using isotope pattern or fragmentation pattern information. The identified molecular formulas may then serve as a basis for subsequent identification steps. The problem of decomposing peak masses lies at the core of practically every approach for the interpretation of small molecule MS data that does not directly depend on a spectra library: See for example [3, 6, 13, 18, 19, 21, 23, 25].

First approaches for decomposing masses date back to at least the 1970’s [9, 22], where the naïve search tree algorithm described below is mentioned for the first time. Running times of this algorithm are often prohibitive, particularly for large alphabets of elements. Fürst *et al.* [10] proposed a faster decomposition algorithm which, unfortunately, is limited to the four elements CHNO. For integer-valued masses, the problem is closely related to unbounded integer knapsacks [17]. Here, an algorithm that works for arbitrary alphabets of elements is “folklore” in computer science, and can solve the problem in pseudo-polynomial running time [17]. Böcker and Lipták [4, 5] presented an algorithm that requires only little memory and is swift in practice. Decomposing real-valued masses using the integer-mass approaches was introduced in [3]. See also the review [24].

In this paper, we present three algorithm engineering techniques to speed up the decomposition of peak masses in practice. First, we replace the recursive decomposition algorithm by an iterative version that mimics the recursive enumeration, but is faster in practice. Second, we show how to minimize rounding error accumulation when transforming real-valued masses to their integer-valued counterparts. Finally, we modify the algorithm from [5] to decompose intervals instead of single masses, based on ideas from [1]. Together, these improvements result in 4.2-fold decreased running times, compared to the previously fastest approach [3]. We evaluate this on four experimental datasets.

2 Preliminaries

In the following, let $a'_1, \dots, a'_k \in \mathbb{R}_{>0}$ denote the masses of our alphabet Σ , see Table 1 for masses of elements.¹ We usually assume that these masses are ordered and, in particular, that a'_1 is minimum. We want to decompose the mass of a peak in a measured spectrum, but we have to take into account measurement inaccuracies. To this end, we assume that we are given an interval $[l', u'] \subseteq \mathbb{R}$, and want to find all *decompositions* $c = (c_1, \dots, c_k) \in \mathbb{N}^k$ such that $\sum_{j=1}^k c_j a'_j \in [l', u']$. Analogously, we can decompose integer masses over an alphabet of integer

¹ For readability, we will denote the real-valued masses by a'_j, l', u' and the integer-valued masses by a_j, l, u .

Table 1. Elements considered in this paper. For each element we report the *monoisotopic mass*, that is, the mass of the naturally occurring isotope with smallest nucleon number (NN). Masses taken from [2].

element	symbol	NN	mass (Da)	element	symbol	NN	mass (Da)
hydrogen	H	1	1.007825	sulfur	S	32	31.972071
carbon	C	12	12.000000	chlorine	Cl	35	34.968853
nitrogen	N	14	14.003074	bromine	Br	79	78.918337
oxygen	O	16	15.994915	iodine	I	127	126.904473
phosphor	P	31	30.973762				

masses a_1, \dots, a_k . Again, we assume that masses are ordered and that a_1 is minimum. We want to find all *decompositions* $c = (c_1, \dots, c_k) \in \mathbb{N}^k$ such that $\sum_{j=1}^k c_j a_j \in \{l, \dots, u\}$ where l, u are integer.

Böcker *et al.* [3] describe how to transform an instance of the real-valued mass decomposition problem into an integer-valued instance, see there for details. We briefly recapitulate the method: For a given blowup factor $b \in \mathbb{R}$ we transform real-valued masses a'_1, \dots, a'_k into integer masses $a_j := \lfloor ba'_j \rfloor$. (Different from [3] we will round down here, as this presentation appears to be somewhat easier to follow.) We want to find all real-valued decompositions in the interval $[l', u'] \subseteq \mathbb{R}$. Regarding the upper bound we have $\sum_j c_j a_j \leq b \sum_j c_j a'_j \leq bu'$ and, as the left side is integer, $\sum_j c_j a_j \leq \lfloor bu' \rfloor$. For the lower bound, we have to take into account rounding error accumulation: We define relative rounding errors

$$\Delta_j = \Delta_j(b) := \frac{ba'_j - \lfloor ba'_j \rfloor}{a'_j} = b - \frac{\lfloor ba'_j \rfloor}{a'_j} \quad \text{for } j = 1, \dots, k, \quad (1)$$

and note that $0 \leq \Delta_j < \frac{1}{a'_j}$. Let $\Delta = \Delta(b) := \max_j \{\Delta_j\}$. Then, $\sum_j a'_j c_j \geq l'$ implies $\sum_j a_j c_j \geq bl' - \Delta l'$, see [3] for details. To this end, we can decompose integer masses in the interval $l := \lceil bl' - \Delta l' \rceil$ to $u := \lfloor bu' \rfloor$. Doing so, we guarantee that no real-valued decomposition will be missed. The list of integer decompositions will contain false positive decompositions, but these can be easily filtered out by checking $\sum_j c_j a'_j \in [l', u']$ for each integer decomposition.

Mass accuracy of an MS instrument depends linearly on the mass that we measure, and is usually given in “parts per million” (ppm). Formally, for a given mass $m \in \mathbb{R}$ and some $\epsilon > 0$, we want to find all masses in the interval $[l', u']$ with $l' := (1 - \epsilon)m$ and $u' := (1 + \epsilon)m$. To this end, the width $u' - l' = 2\epsilon m$ of the interval that we want to decompose, is *linear* in the mass m .

For integer masses, the number of decompositions $\gamma(m)$ of some mass m asymptotically equals $\gamma(m) \sim \frac{1}{a_1 \dots a_k} m^{k-1}$ [26]. This leads to a similar estimate for real-valued masses [3]. In general, this asymptotic estimate is accurate only for very large masses; for molecular formulas, it is a relatively good estimate even for small masses [3].

```

1: procedure FINDALLRECURSIVE(integer  $i \leq k$ , mass  $m$ , decomposition  $c$ )
2:   if  $i = 0$  then
3:     Output  $c$  and return
4:   end if
5:   if  $\text{decomposable}(i - 1, m) = 1$  then
6:     FINDALLRECURSIVE( $i - 1, m, c$ )
7:   end if
8:   if  $m \geq a_i$  and  $\text{decomposable}(i, m - a_i) = 1$  then
9:     FINDALLRECURSIVE( $i, m - a_i, c + e_i$ )
10:  end if
11: end procedure

```

Fig. 1. Recursive algorithm for enumerating all decompositions of a given mass m . To decompose mass M , this algorithm is initially called as $\text{FINDALLRECURSIVE}(k, M, 0)$. Vector e_i denotes the i^{th} unit vector.

3 Algorithms for Decomposing Masses

The conceptually simplest algorithm for decomposing masses is a search tree that recursively builds up the decompositions (molecular formulas), taking into account the mass accuracy. The algorithm is very similar to FINDALLRECURSIVE in Fig. 1, we leave out the straightforward details. This algorithm has been suggested several times in the literature [1, 9, 22]. The major disadvantage of this algorithm is that its running time is not output-sensitive: For a constant alphabet of size k , the algorithm requires $\Theta(m^{k-1})$ time, even if there is not a single decomposition.

To decompose an integer over an alphabet $\Sigma = \{a_1, \dots, a_k\}$ of integer masses, we can use algorithm FINDALLRECURSIVE in Fig. 1. This algorithm requires an oracle such that $\text{decomposable}(i, m) = 1$ if and only if m is decomposable over the sub-alphabet $\{a_1, \dots, a_i\}$. We can build this oracle using a dynamic programming table $D[i, m] = \text{decomposable}(i, m)$: We initialize $D[0, 0] = 1$, $D[0, m] = 0$ for $m \geq 1$, and use the recurrence $D[i, m] = \max\{D[i - 1, m], D[i, m - a_i]\}$ for $m \geq a_i$ and $D[i, m] = D[i - 1, m]$ otherwise. This approach requires $O(kM)$ memory to store D and $O(kM)$ time to compute it, where M is the largest mass that we want to decompose. The algorithm has polynomial time and space with regards to the mass m we want to decompose.² Time for computing each decomposition is $O(km/a_1)$. The decomposition algorithm has polynomial delay and, hence, is output-dependent.

A more memory-efficient way to build the required oracle, is to use the extended residue table (ERT) from [4]: For an integer m , let $m \bmod a_1$ denote the *residue* of m modulo a_1 , where $m \bmod a_1 \in \{0, \dots, a_1 - 1\}$. We define the

² Precisely speaking, running time is pseudo-polynomial in the input m , as polynomial running time would require polynomial dependency on $\log m$. Since the decision version of the problem (“is there a decomposition of mass m ?”) is weakly NP-hard [16], there is little hope for an algorithm with running time polynomial in $\log m$. We will ignore this detail in the following.

extended residue table $N[0 \dots k, 0 \dots a_1 - 1]$ by

$$N[i, r] = \min\{m : r = m \bmod a_1, \text{ and } m \text{ is decomposable over } \{a_1, \dots, a_i\}\}$$

where we define $N[i, r] = +\infty$ if no such number exists, that is, if the minimum is taken over the empty set. Now, we can define the oracle by

$$\text{decomposable}(i, m) := \begin{cases} 1 & \text{if } m \geq N[i, m \bmod a_1], \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Storing the ERT requires $O(ka_1)$ space, and the table can be computed in $O(ka_1)$ time. The time for computing each decomposition using algorithm FINDALLRECURSIVE is again $O(km/a_1)$ but can be reduced to $O(ka_1)$ [4, 5]. The conceptual advantage of this approach is that we do not have to decide upon some “largest mass” during preprocessing, and that both time and space do no longer depend on the mass m that we want to decompose.

4 Iterative Version of the Decomposition Algorithm

The naïve search tree algorithm for decomposing masses can easily be made iterative using k nested FOR-loops. Replacing algorithm FINDALLRECURSIVE by an iterative version is slightly more complicated, as we have to avoid “empty branches” of the search tree, where no decomposition can be found. We can use an auxiliary Boolean vector $d[1 \dots k]$ that stores which of the two alternative recursive calls from FINDALLRECURSIVE has been executed last. We present a more involved version of the iterative algorithm in Fig. 2. We avoid the auxiliary vector by deciding on the alternative calls directly from the decomposition c .

The algorithm is independent of the actual implementation of the oracle $\text{decomposable}(i, m)$. Asymptotically, worst-case running time is identical to that of the recursive version; in practice, the iterative version is nevertheless considerably faster than its recursive counterpart, as we avoid the stack handling.

5 Selecting Optimal Blowup Factors

Transforming the real-valued decomposition problem into its integer-valued counterpart requires that we choose some blowup factor $b \in \mathbb{R}$. Due to the rounding error correction, we have to decompose roughly $\Delta(b)u$ “auxiliary” integers in addition to the $b(u - l + 1)$ “regular” integers, where $(u - l + 1) \in \Theta(u)$. It is reasonable to ask for a blowup factor such that the ratio of additional integers $\frac{\Delta(b)u}{bu} = \frac{\Delta(b)}{b}$ is minimum. For “sufficiently small” $b > 0$ we have $\Delta(b) = b$ and, hence, $\frac{\Delta(b)}{b} = 1$.

Since $\Delta(b) < \max_j \{\frac{1}{a_j}\}$ is bounded, we can make $\frac{\Delta(b)}{b}$ arbitrarily small by choosing an arbitrarily large blowup factor b . But this is not realistic in applications, as memory requirements increase linearly with b . To this end, we suppose


```

1: procedure FINDALLITERATIVE(mass  $m$ )
2:   decomposition  $c = (c_1, \dots, c_k) \leftarrow 0$ 
3:   integer  $i \leftarrow k$  ▷ constant alphabet size  $k$ 
4:   while  $i \leq k$  do
5:     if  $\text{decomposable}(i, m) = 0$  then ▷ is this decomposable at all?
6:       while  $i \leq k$  and  $\text{decomposable}(i, m) = 0$  do ▷ no, end "recursion"
7:          $m \leftarrow m + c_i a_i$ 
8:          $c_i \leftarrow 0$ 
9:          $i \leftarrow i + 1$ 
10:      end while ▷ now,  $\text{decomposable}(i, m) = 1$  holds
11:      if  $i \leq k$  then
12:         $m \leftarrow m - a_i$ 
13:         $c_i \leftarrow c_i + 1$ 
14:      end if
15:      else ▷ yes, decomposable
16:        while  $i > 1$  and  $\text{decomposable}(i - 1, m) = 1$  do
17:           $i \leftarrow i - 1$ 
18:        end while ▷ now,  $\text{decomposable}(i, m) = 1$ 
19:        if  $i = 1$  then ▷ output decomposition
20:           $c_1 \leftarrow \lfloor m/a_1 \rfloor$  ▷ (*)
21:          Output  $c = (c_1, \dots, c_k)$ 
22:           $i \leftarrow 2$  ▷ correct  $i$ 
23:        end if
24:        if  $i \leq k$  then ▷ move to next element
25:           $m \leftarrow m - a_i$ 
26:           $c_i \leftarrow c_i + 1$ 
27:        end if
28:      end if
29:    end while
30: end procedure

```

Fig. 2. Iterative algorithm for enumerating all decompositions of a given mass m

that memory considerations imply an upper bound of $B \in \mathbb{R}$. We want to find $b \in (0, B)$ such that $\frac{\Delta(b)}{b}$ is minimized. We can explicitly find an optimal b as follows: First, we consider the functions

$$\Delta_j : \mathbb{R} \rightarrow \mathbb{R} \quad \text{with} \quad \Delta_j(b) := b - \frac{1}{a'_j} \lfloor b a'_j \rfloor,$$

for all $j = 1, \dots, k$. Each Δ_j is a piecewise linear function with discontinuities $\frac{1}{a'_j}, \frac{2}{a'_j}, \dots, \frac{\lfloor a'_j B \rfloor}{a'_j}$. In every interval, this function has slope 1. Next, we set $\varphi_1 \equiv \Delta_1$ and for $j \geq 2$, we define φ_j as the maximum of φ_{j-1} and Δ_j . Then, φ_j is a piecewise linear function with $O((a'_1 + \dots + a'_j)B)$ discontinuities. Finally, $\Delta \equiv \varphi_k$ is a piecewise linear function with $O((a'_1 + \dots + a'_k)B)$ discontinuities. We sweep over the discontinuities from left to right, and for each discontinuity b , we calculate all $\Delta_j(b)$ and $\Delta(b)$. This can be easily achieved in time $O(k(a'_1 + \dots + a'_k)B) = O(k^2 a'_k B)$, where a'_k is the largest mass in the alphabet. For every

piecewise linear part $I \subseteq \mathbb{R}$ of Δ the minimum of $\frac{\Delta(b)}{b}$ must be located at one of the terminal points, so it suffices to test the $O(ka'_k B)$ discontinuities to find the minimum of $\frac{\Delta(b)}{b}$.

Table 2. Locally optimal blowup factors in the range $b \in [1000, 100000]$, for alphabet of elements CHNOPS (left) and CHNOPSClBrI (right). For two consecutive entries b', b'' from the table, b' is locally optimal in $(0, b'')$, so $\frac{\Delta(b')}{b'} \leq \frac{\Delta(b'')}{b''}$ for all $b \in (0, b'')$. Values b rounded up, other values rounded down. *Entries $\Delta(b)$ that are smaller than both the previous and the following entry.

blowup b	$\Delta(b)$	$\Delta(b)/b$	blowup b	$\Delta(b)$	$\Delta(b)/b$
1127.1810743	0.014407	$1.278199 \cdot 10^{-5}$	1127.1810743	0.014407	$1.278199 \cdot 10^{-5}$
1128.1808548	0.014188	$1.257608 \cdot 10^{-5}$	1128.1808548	0.014188	$1.257608 \cdot 10^{-5}$
1181.7527469	0.012223	$1.034371 \cdot 10^{-5}$	1182.7510330	0.012772*	$1.079876 \cdot 10^{-5}$
1182.7510330	0.010729*	$9.071515 \cdot 10^{-6}$	1680.8473159	0.013982	$8.318731 \cdot 10^{-6}$
1680.8473159	0.013982	$8.318731 \cdot 10^{-6}$	1681.8470521	0.013718	$8.156909 \cdot 10^{-6}$
1681.8470521	0.013718	$8.156909 \cdot 10^{-6}$	2064.8444567	0.012121	$5.870279 \cdot 10^{-6}$
1896.1666667	0.013377	$7.054863 \cdot 10^{-6}$	2309.9247647	0.011976	$5.184759 \cdot 10^{-6}$
1897.1666667	0.012530	$6.604628 \cdot 10^{-6}$	2310.9237056	0.010026*	$4.338805 \cdot 10^{-6}$
2064.8444567	0.012121	$5.870279 \cdot 10^{-6}$	3268.4252033	0.013661	$4.179833 \cdot 10^{-6}$
2309.9247647	0.008097	$3.505730 \cdot 10^{-6}$	3269.4249838	0.012594*	$3.852308 \cdot 10^{-6}$
2310.9237056	0.007038*	$3.045939 \cdot 10^{-6}$	3897.5019225	0.013159	$3.376302 \cdot 10^{-6}$
2939.0036991	0.007079	$2.408730 \cdot 10^{-6}$	3898.5011421	0.012060	$3.093702 \cdot 10^{-6}$
5248.9269781	0.010311	$1.964477 \cdot 10^{-6}$	4206.0872326	0.011084	$2.635363 \cdot 10^{-6}$
5334.2592503	0.009867	$1.849794 \cdot 10^{-6}$	4207.0871650	0.010920*	$2.595644 \cdot 10^{-6}$
5335.2582387	0.008238	$1.544184 \cdot 10^{-6}$	5248.9282869	0.011620	$2.213814 \cdot 10^{-6}$
5963.3376861	0.008003*	$1.342117 \cdot 10^{-6}$	5802.5956469	0.012587	$2.169277 \cdot 10^{-6}$
8519.3436784	0.010925	$1.282415 \cdot 10^{-6}$	5963.3376861	0.008789*	$1.473957 \cdot 10^{-6}$
9072.0116320	0.011631	$1.282179 \cdot 10^{-6}$	7146.0837847	0.010040	$1.405093 \cdot 10^{-6}$
9456.0064464	0.011287	$1.193667 \cdot 10^{-6}$	9701.0919315	0.009977	$1.028442 \cdot 10^{-6}$
9457.0057796	0.010840	$1.146246 \cdot 10^{-6}$	10415.5007612	0.007678*	$7.371910 \cdot 10^{-7}$
9701.0919315	0.009977	$1.028442 \cdot 10^{-6}$	15664.4258530	0.009633	$6.149738 \cdot 10^{-7}$
10415.5000000	0.006917*	$6.641097 \cdot 10^{-7}$	16378.8350902	0.008707	$5.316052 \cdot 10^{-7}$
12725.4231558	0.007214	$5.669173 \cdot 10^{-7}$	16379.8339400	0.006683*	$4.080188 \cdot 10^{-7}$
12726.4199232	0.004531*	$3.560891 \cdot 10^{-7}$	26710.0000000	0.010596	$3.967193 \cdot 10^{-7}$
18689.7544746	0.004911	$2.627644 \cdot 10^{-7}$	26794.3334813	0.010397	$3.880516 \cdot 10^{-7}$
26080.9191881	0.005617	$2.153888 \cdot 10^{-7}$	26795.3333334	0.009376	$3.499167 \cdot 10^{-7}$
29105.2521655	0.005901	$2.027566 \cdot 10^{-7}$	28390.8415041	0.008170	$2.877948 \cdot 10^{-7}$
32044.2526951	0.005370*	$1.676031 \cdot 10^{-7}$	29105.2521655	0.007871*	$2.704370 \cdot 10^{-7}$
42459.7510861	0.006746	$1.588932 \cdot 10^{-7}$	34355.1749622	0.008295	$2.414630 \cdot 10^{-7}$
42460.7500000	0.006678	$1.572787 \cdot 10^{-7}$	38807.3390692	0.008710	$2.244429 \cdot 10^{-7}$
44770.6696249	0.002958*	$6.607444 \cdot 10^{-8}$	44769.6758508	0.009184	$2.051405 \cdot 10^{-7}$
96687.4182692	0.005238	$5.417847 \cdot 10^{-8}$	44770.6721964	0.005529*	$1.235112 \cdot 10^{-7}$
			63460.4230255	0.006358	$1.002015 \cdot 10^{-7}$
			90170.4178028	0.008375	$9.288531 \cdot 10^{-8}$
			96687.4182692	0.006542	$6.767102 \cdot 10^{-8}$

We compute optimal blowup factors for the default alphabet CHNOPS, and for the extended alphabet CHNOPSClBrI suggested in [25]. Since we can find arbitrarily small blowup factors by increasing b , any blowup factor $b' \in \mathbb{R}$ can only be *locally optimal*: that is, for an upper bound $b'' \in \mathbb{R}$ and all $b \in (0, b'')$ we then have $\frac{\Delta(b')}{b'} \leq \frac{\Delta(b)}{b}$. See Table 2 for all locally optimal blowup factors in the range $b \in [1000, 100000]$. We do not report blowup factors below 1000 as, for the mass accuracies considered here, such blowup factors result in a dramatic increase of false positive decompositions and, hence, are not useful in practice.

6 Range Decompositions

Agarwal *et al.* [1] suggested to decompose a range of masses $m, \dots, m + \mu - 1$ for integers m, μ , instead of decomposing each mass individually. In theory, this does not noticeably improve running times: Using the approaches described above, we can iterate over all masses $m' = m, \dots, m + \mu - 1$. Let $\gamma(m, m + \mu)$ denote the number of decompositions in this range, then this results in a total running time of $O(\gamma(m, m + \mu)ka_1 + \mu)$ for the approach of [4, 5]. Clearly, the additive $O(\mu)$ term can be ignored in practice.

But from an algorithm engineering perspective, decomposing a range instead of an integer may result in considerable time savings: For the algorithm `FINDALLRECURSIVE`, this can significantly reduce the number of recursive function calls. To this end, given a range $m, \dots, m + \mu - 1$ and an alphabet $\Sigma = \{a_1, \dots, a_k\}$ we assume an oracle with $\text{decomposable}(i, m) = 1$ if and only if there is at least one $m' \in \{m, \dots, m + \mu - 1\}$ that is decomposable over $\{a_1, \dots, a_i\}$. Solely for the sake of clarity, we will assume μ to be fixed, although it obviously depends on the mass that we want to decompose. With this new oracle, we can reuse the algorithms from Fig. 1 and 2 without further changes.

In the following, let $\text{decomposable}_0(i, m)$ denote the original oracle for a single mass m . Then, a straightforward oracle for the range decomposition is

$$\text{decomposable}(i, m) = \max_{m' \in \{m \dots m + \mu - 1\}} \text{decomposable}_0(i, m').$$

But this requires μ calls of the $\text{decomposable}_0(i, m)$ oracle and results in a *multiplicative* factor of $O(\mu)$ in the running time. Agarwal *et al.* [1] suggested modifying the integer knapsack recurrence mentioned above, to capture the mass range: To this end, we initialize $D_\mu[0, m] = 1$ for $m = 0, \dots, \mu - 1$ and $D_\mu[0, m] = 0$ for $m \geq \mu$. We use the same recurrence as above, namely $D_\mu[i, m] = \max\{D_\mu[i - 1, m], D_\mu[i, m - a_i]\}$ for $m \geq a_i$ and $D_\mu[i, m] = D_\mu[i - 1, m]$ otherwise. Unfortunately, for each μ that we want to consider for decomposing, this requires preprocessing and storing a dynamic programming table. Again, this is not desirable in application, as the mass error of the measurement increases with mass. So, we have to compute and store a table for every $\mu = 1, \dots, \mu_{\max}$.

But with a small trick, we can reduce the multiplicative factor for space from $O(\mu_{\max})$ to $O(\log \mu_{\max})$: Let decomposable_l be an oracle with $\text{decomposable}(i, m) = 1$ if and only if there is at least one $m' \in \{m, \dots, m + 2^l - 1\}$ that is decomposable over $\{a_1, \dots, a_i\}$. Then, we can “recover” the oracle decomposable for the range $\{m, \dots, m + \mu - 1\}$ as

$$\text{decomposable}(i, m) = \max\{\text{decomposable}_l(i, m), \text{decomposable}_l(i, m + \mu - 2^l)\}$$

where $l := \lfloor \log_2 \mu \rfloor$.

We will now show how to use the extended residue table from [4] for range decompositions: We define a family of extended residue tables N_l for $l = 0, \dots, \lfloor \log_2 \mu \rfloor$, where $N_l[i, r]$ is the minimum of all m with $r = m \bmod a_1$, such that some $m' \in \{m, \dots, m + 2^l - 1\}$ is decomposable over $\{a_1, \dots, a_i\}$.

Again, $N_l[i, r] = +\infty$ if no such number exists. Now, we can re-use the oracle from (2): We have $\text{decomposable}_l(i, m) = 1$ if and only if $m \geq N_l[i, m]$. Storing all extended residue tables requires $O(ka_1 \log \mu_{\max})$ space, and the tables can be computed in $O(ka_1 \log \mu_{\max})$ time using the following simple recurrence: We initialize $N_0[i, r] = N[i, r]$ and use

$$N_{l+1}[i, r] = \min\{N_l[i, r], N_l[i, (r + 2^l) \bmod a_1]\} \quad (3)$$

for $l = 0, \dots, \mu_{\max} - 1$, $i = 0, \dots, k$, and $r = 0, \dots, a_1 - 1$. Here, $N[i, r]$ refers to the extended residue table for the single integer decomposition problem.

The iterative algorithm `FINDALLITERATIVE` does not consider the degenerate case where the width of the interval we want to decompose, is large compared to the masses of the alphabet. In particular, for $u - l \geq a_1$ every decomposition with mass at most u can be “completed” using element a_1 to find a decomposition with mass in $\{l, \dots, u\}$. For this case, we have to adapt the algorithm by replacing the line marked (*) in Fig. 2 by a loop over appropriate numbers of elements a_1 . But for this degenerate case, we find a decomposition for every leaf of the naïve search tree algorithm; so, the iterative version of this algorithm outperforms all other, more involved algorithms.

7 Results

We implement the algorithms mentioned above in Java 1.6. `SEARCHTREE` denotes the naïve search tree algorithm. We distinguish two algorithms based on the recursive decomposition (Fig. 1), namely `RECURSIVE+KNAPSACK` using the knapsack DP, and `RECURSIVE+ERT` using the extended residue table. We also implement two versions of the iterative decomposition (Fig. 2), namely `ITERATIVE+ERT` and `ITERATIVE+RANGE` which uses range decompositions from Sec. 6. For brevity, we exclude other combinations, as these will in all likelihood not result in better running times.

We evaluate the algorithms on four datasets: The *Orbitrap* dataset [20] contains 97 compounds measured on a Thermo Scientific Orbitrap XL instrument. The *MassBank* dataset [12] consists of 370 compounds measured on a Waters Q-ToF Premier spectrometer. The *Eawag* dataset [25] contains 60 compounds measured on a LTQ Orbitrap XL Thermo Scientific and is also accessible from MassBank. The *Hill* dataset [11] consists of 102 compounds with 502 spectra measured on a Waters Micromass QTOF II instrument. We omit experimental details. All of these datasets are used in computations that require the decomposition of peak masses. See Table 3 for details.

Table 3. Statistics of the datasets used in our evaluation

	Orbitrap	MassBank	Eawag	Hill
peaks	5 393	2 455	10 017	12 054
maximum mass	1153	821	444	610
median mass	205	211	149	186

We discard peaks with mass below 100 Da because for such masses, the problem becomes easy regardless of the used algorithm. We report running times for decomposing a single peak mass as well as all peaks in a dataset. For algorithms working on integer masses, we generate integer-valued instances as described in Sec. 2. We use a mass accuracy of 20 ppm, so $\epsilon = 0.00002$. For RECURSIVE+KNAPSACK, RECURSIVE+ERT, and ITERATIVE+ERT, we decompose intervals by decomposing all integer values separately.

Running time measurements are done on a Intel Xeon E5645 with 48 GB RAM. For each algorithm we repeat computations five times and report minimum running times. For the complete datasets, total running times can be found in Table 4 and Figure 3 for alphabets CHNOPSClBrI and CHNOPS. We find that the fastest ERT-based algorithm ITERATIVE+RANGE was 56-fold faster than the SEARCHTREE algorithm for alphabet CHNOPS; this increases to 150-fold speedup for alphabet CHNOPSClBrI.

Replacing the knapsack DP by an ERT table results in a 2.3-fold speedup. In addition, memory requirements decrease considerably: For example, to decompose the maximal mass of 1153.395 with a blowup of 100 000 and the extended

Table 4. We report running times of the algorithms for decomposing all peaks in a dataset, for alphabets CHNOPSClBrI (top) and CHNOPS (bottom). Running times are reported in seconds except for SEARCHTREE and alphabet CHNOPSClBrI, where running times are reported in minutes. For all measurements, we use mass accuracy 20 ppm.

algorithm	blowup	Orbitrap	MassBank	Eawag	Hill
SEARCHTREE	—	166.0 min	37.3 min	2.3 min	23.8 min
RECURSIVE+KNAPSACK	100000	729.70	86.48	8.54	34.12
RECURSIVE+ERT		308.22	39.67	2.10	19.85
	100000	216.21	30.83	2.10	14.64
	44770.6721964	129.97	21.25	1.62	9.75
ITERATIVE+ERT	5963.3376861	102.97	17.92	0.94	7.32
	1182.7510330	122.91	22.40	0.95	7.66
	1000	2289.36	420.58	15.81	190.32
ITERATIVE+RANGE	5963.3376861	66.88	13.98	0.88	6.13
algorithm	blowup	Orbitrap	MassBank	Eawag	Hill
SEARCHTREE	—	221.97	106.16	28.65	131.07
RECURSIVE+KNAPSACK	100000	54.47	19.98	5.62	13.21
RECURSIVE+ERT		18.03	5.77	1.17	4.77
	100000	12.99	4.63	1.15	4.31
	44770.6721964	10.15	3.63	0.75	3.14
ITERATIVE+ERT	5963.3376861	5.86	2.27	0.39	1.93
	1182.7510330	7.39	2.76	0.40	2.17
	1000	121.83	46.60	5.89	40.71
ITERATIVE+RANGE	5963.3376861	4.71	1.89	0.36	1.77

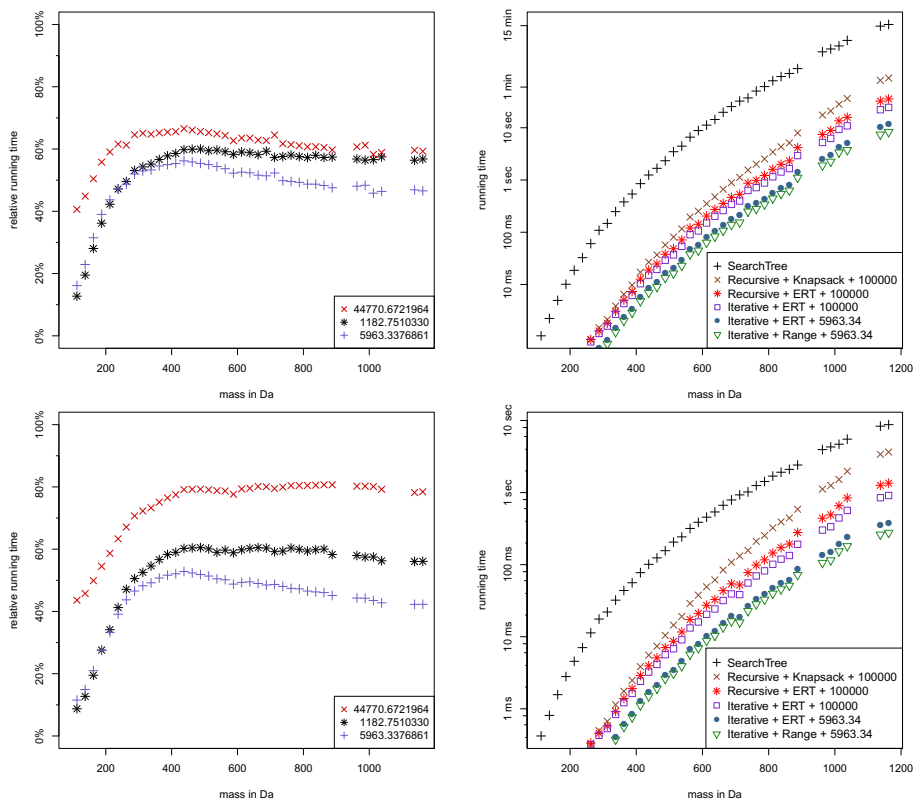


Fig. 3. Running times on the Orbitrap dataset using alphabet CHNOPSCiBrI (top) or CHNOPS (bottom), and 20 ppm mass accuracy. Average running times are reported for bins of width 25 Da. Left: Relative running times of ITERATIVE+ERT for different blowup factors, normalized to blowup factor $b = 100000$ as 100%. Right: Running times for different algorithms. Note the logarithmic y-axis.

alphabet, the integer knapsack DP table requires 124 megabyte, whereas the ERT requires only 3.5 megabyte.

Replacing the recursive search by its iterative counterpart has only limited impact: We find that the iterative algorithm ITERATIVE+ERT is merely 1.4-fold faster than its recursive counterpart, RECURSIVE+ERT.

We observe that the blowup factor has a major impact on running times. Choosing locally optimal blowup factors does in fact significantly reduce running times: For example, blowup factor $b = 1182.7510330$ results in 19-fold faster running times than $b = 1000$. We test all locally optimal blowup factors from Table 2. We observe best running time for blowup $b = 5963.3376861$, whereas larger and smaller blowup factors result in increased running times. We refrain from reporting all running times, see Fig. 3 and Table 4 for some examples. The best blowup factor $b = 5963.3376861$ results in a two-fold speedup when

compared to the “default” blowup factor $b = 100000$ from [3]. As a pleasant side effect, this decreases the memory requirements of the algorithm by 94 %.

The range decomposition improves running times by about 1.5-fold for the best blowup factor. A stronger improvement is achieved when more integers are to be decomposed using, say, a larger blowup factor. Using blowup factor $b = 5963.3376861$, memory increases to 2.1 MB for storing ten ERT tables.

Finally, we repeat our experiments for an improved mass accuracy 1 ppm. For all algorithms except SEARCHTREE this results in roughly a 6- to 11-fold decrease of running time, whereas SEARCHTREE running times does not change. For this mass accuracy and alphabet CHNOPSCIBrI, the best algorithm ITERATIVE+RANGE is 1000-fold faster than the naïve SEARCHTREE algorithm.

8 Conclusion

We suggest three techniques to improve the running time for decomposing real masses. We measure the improvements on four different datasets. All techniques together result in a 4-fold improvement in running time, compared to the RECURSIVE+ERT algorithm from [4, 5]. We note in passing that the implementation of the RECURSIVE+ERT algorithm used in this evaluation, was two-fold faster than the one provided as part of SIRIUS [3]. The competitive edge of the new method is even larger for “hard” problem instances, e.g. high masses, large mass deviations, and bigger alphabets. Compared to the naïve search tree algorithm, we reach improvements between 56-fold and 1000-fold, reducing the total running times from hours to minutes or even seconds.

Regarding the degenerate case $u - l \geq a_1$ mentioned in Sec. 6, we argue that this case is of no interest, from either the practical or the theoretical side: Modern MS instruments easily reach mass accuracies of 10 ppm and below, whereas metabolite and even peptide masses rarely exceed 5000 Da. Even a peptide mass of 5000 Da can be measured with an accuracy of at least 0.05 Da, well below the mass of a single ^1H atom. From the theoretical side, we would have to deal with a humongous number of decompositions, rendering time to compute the decompositions irrelevant in comparison to subsequent analysis steps.

Acknowledgments. We thank Tim White for proofreading earlier versions of this work.

References

1. Agarwal, D., Cazals, F., Malod-Dognin, N.: Stoichiometry determination for mass-spectrometry data: the interval cases. In: Research Report 8101, Inria, Research Centre Sophia Antipolis – Méditerranée (October 2012)
2. Audi, G., Wapstra, A., Thibault, C.: The AME2003 atomic mass evaluation (ii): Tables, graphs, and references. Nucl. Phys. A 729, 129–336 (2003)
3. Böcker, S., Letzel, M., Lipták, Z., Pervukhin, A.: SIRIUS: Decomposing isotope patterns for metabolite identification. Bioinformatics 25(2), 218–224 (2009)

4. Böcker, S., Lipták, Z.: Efficient mass decomposition. In: Proc. of ACM Symposium on Applied Computing (ACM SAC 2005), pp. 151–157. ACM press, New York (2005)
5. Böcker, S., Lipták, Z.: A fast and simple algorithm for the Money Changing Problem. *Algorithmica* 48(4), 413–432 (2007)
6. Böcker, S., Rasche, F.: Towards de novo identification of metabolites by analyzing tandem mass spectra. *Bioinformatics* 24, I49–I55 (2008); Proc. of European Conference on Computational Biology (ECCB 2008)
7. Cooper, M.A., Shlaes, D.: Fix the antibiotics pipeline. *Nature* 472(7341), 32 (2011)
8. Cortina, N.S., Krug, D., Plaza, A., Revermann, O., Müller, R.: Myxoprincomide: a natural product from *Myxococcus xanthus* discovered by comprehensive analysis of the secondary metabolome. *Angew. Chem. Int. Ed. Engl.* 51(3), 811–816 (2012)
9. Dromey, R.G., Foyster, G.T.: Calculation of elemental compositions from high resolution mass spectral data. *Anal. Chem.* 52(3), 394–398 (1980)
10. Fürst, A., Clerc, J.-T., Pretsch, E.: A computer program for the computation of the molecular formula. *Chemom. Intell. Lab. Syst.* 5, 329–334 (1989)
11. Hill, D.W., Kertesz, T.M., Fontaine, D., Friedman, R., Grant, D.F.: Mass spectral metabolomics beyond elemental formula: Chemical database querying by matching experimental with computational fragmentation spectra. *Anal. Chem.* 80(14), 5574–5582 (2008)
12. Horai, H., Arita, M., Kanaya, S., Nihei, Y., Ikeda, T., Suwa, K., Ojima, Y., Tanaka, K., Tanaka, S., Aoshima, K., Oda, Y., Kakazu, Y., Kusano, M., Tohge, T., Matsuda, F., Sawada, Y., Hirai, M.Y., Nakanishi, H., Ikeda, K., Akimoto, N., Maoka, T., Takahashi, H., Ara, T., Sakurai, N., Suzuki, H., Shibata, D., Neumann, S., Iida, T., Tanaka, K., Funatsu, K., Matsuura, F., Soga, T., Taguchi, R., Saito, K., Nishioka, T.: MassBank: A public repository for sharing mass spectral data for life sciences. *J. Mass Spectrom.* 45(7), 703–714 (2010)
13. Jarussophon, S., Acoca, S., Gao, J.-M., Deprez, C., Kiyota, T., Draghici, C., Purisima, E., Konishi, Y.: Automated molecular formula determination by tandem mass spectrometry (MS/MS). *Analyst.* 134(4), 690–700 (2009)
14. Last, R.L., Jones, A.D., Shachar-Hill, Y.: Towards the plant metabolome and beyond. *Nat. Rev. Mol. Cell Biol.* 8, 167–174 (2007)
15. Li, J.W.-H., Vederas, J.C.: Drug discovery and natural products: End of an era or an endless frontier? *Science* 325(5937), 161–165 (2009)
16. Lueker, G.S.: Two NP-complete problems in nonnegative integer programming. Technical Report TR-178, Department of Electrical Engineering, Princeton University (March 1975)
17. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Chichester (1990)
18. Meringer, M., Reinker, S., Zhang, J., Muller, A.: MS/MS data improves automated determination of molecular formulas by mass spectrometry. *MATCH-Commun. Math. Co.* 65, 259–290 (2011)
19. Pluskal, T., Uehara, T., Yanagida, M.: Highly accurate chemical formula prediction tool utilizing high-resolution mass spectra, MS/MS fragmentation, heuristic rules, and isotope pattern matching. *Anal. Chem.* 84(10), 4396–4403 (2012)
20. Rasche, F., Scheubert, K., Hufsky, F., Zichner, T., Kai, M., Svatoš, A., Böcker, S.: Identifying the unknowns by aligning fragmentation trees. *Anal. Chem.* 84(7), 3417–3426 (2012)
21. Rasche, F., Svatoš, A., Maddula, R.K., Böttcher, C., Böcker, S.: Computing fragmentation trees from tandem mass spectrometry data. *Anal. Chem.* 83(4), 1243–1251 (2011)

22. Robertson, A.L., Hamming, M.C.: MASSFORM: a computer program for the assignment of elemental compositions to high resolution mass spectral data. *Biomed. Mass Spectrom.* 4(4), 203–208 (1977)
23. Rojas-Chertó, M., Kasper, P.T., Willighagen, E.L., Vreeken, R.J., Hankemeier, T., Reijmers, T.H.: Elemental composition determination based on MS^n . *Bioinformatics* 27, 2376–2383 (2011)
24. Scheubert, K., Hufsky, F., Böcker, S.: Computational mass spectrometry for small molecules. *J. Cheminform.* 5, 12 (2013)
25. Stravs, M.A., Schymanski, E.L., Singer, H.P., Hollender, J.: Automatic recalibration and processing of tandem mass spectra using formula annotation. *J. Mass Spectrom.* 48(1), 89–99 (2013)
26. Wilf, H.: *generatingfunctionology*, 2nd edn. Academic Press (1994), Freely available from <http://www.math.upenn.edu/~wilf/DownldGF.html>

On NP-Hardness of the Paired de Bruijn Sound Cycle Problem

Evgeny Kapun and Fedor Tsarev

St. Petersburg National Research University of Information
Technologies, Mechanics and Optics
Genome Assembly Algorithms Laboratory
197101, Kronverksky pr., 49, St. Petersburg, Russia
tsarev@rain.ifmo.ru
<http://genome.ifmo.ru/>

Abstract. The paired de Bruijn graph is an extension of de Bruijn graph incorporating mate pair information for genome assembly proposed by Mevdedev et al. However, unlike in an ordinary de Bruijn graph, not every path or cycle in a paired de Bruijn graph will spell a string, because there is an additional soundness constraint on the path. In this paper we show that the problem of checking if there is a sound cycle in a paired de Bruijn graph is NP-hard in general case. We also explore some of its special cases, as well as a modified version where the cycle must also pass through every edge.

Keywords: paired de Bruijn graph, genome assembly, complexity, NP-hard.

1 Introduction

Current genome sequencing technologies rely on the shotgun method — the genome is split into several small fragments which are read directly. Some of the technologies generate single reads, while others generate mate-pair reads — genome fragments are read from both sides. The problem of reconstructing the initial genome from these small fragments (reads) is known as the genome assembly problem. It is one of the fundamental problems of bioinformatics. Several models for genome assembly were studied by researchers.

One of the models for the single reads case is based on the maximum parsimony principle — the original genome should be the shortest string containing all reads as substrings. This leads to the Shortest Common Superstring (SCS) problem which is NP-hard [1]. In the de Bruijn graph model proposed in [8] each read is represented by a walk in the graph. Any walk containing all the reads as subwalks represents a valid assembly. Consequently, the genome assembly problem is formulated as finding the shortest superwalk. This problem, known as Shortest De Bruijn Superwalk problem (SDBS), was shown to be NP-hard [6].

In [5] an algorithm for reads' copy counts estimation based on maximum likelihood principle was proposed. A similar algorithm can be applied to find

multiplicities of the de Bruijn graph edges, so, the De Bruijn Superwalk with Multiplicities problem (DBSM) can be formulated. This problem have been proven to be NP-hard as well [2].

Paired-end reads case is much less studied. To the best of our knowledge the only model which deals with paired-end reads is the paired de Bruijn graph proposed in [7]. However, not every path or cycle in a paired de Bruijn graph corresponds to a correct genome assembly, because there is an additional soundness constraint on the walk. Computational complexity for the problem of finding a sound cycle in the paired de Bruijn graph remained unknown [9]. In this paper we show that this problem is NP-hard.

2 Definitions

A *de Bruijn graph* of order k over an alphabet Σ is a directed graph in which every vertex has an associated label (a string over Σ) of length k and every edge has an associated label of length $k+1$. All labels within a graph must be distinct. If an edge (u, v) has an associated label l , then the label associated with u must be a prefix of l and the label associated with v must be a suffix of l .

Every path in a de Bruijn graph spells a string. A string spelled by a path $v_1, e_1, v_2, \dots, e_{n-1}, v_n$ of length n is a unique string s of length $n+k-1$ such that the label associated with v_i occurs in s at position i for all $1 \leq i \leq n$, and the label associated with e_i occurs in s at position i for all $1 \leq i \leq n-1$. Every cycle of length n in a de Bruijn graph spells a cyclic string of length n having the same properties.

In a *paired de Bruijn graph* each vertex and each edge has an associated *bilabel* instead of a label. A bilabel is an ordered pair of strings of the same length (equal to the order of the graph), denoted as (a, b) . We say that (a_1, b_1) is a prefix of (a_2, b_2) iff a_1 is a prefix of a_2 and b_1 is a prefix of b_2 . Suffix is defined analogously. As in ordinary de Bruijn graphs, all bilabels must be distinct, however, individual labels of which bilabels consist may coincide.

Similarly to the ordinary de Bruijn graph, every path in a paired de Bruijn graph spells a pair of strings, and every cycle spells a pair of cyclic strings. We say that a pair of strings $(s_1 s_2 \dots s_n, t_1 t_2 \dots t_n)$ of length n *matches with shift d* iff $s_{i+d} = t_i$ for all $1 \leq i \leq n-d$. Analogously, a pair of cyclic strings $(s_1 s_2 \dots s_n, t_1 t_2 \dots t_n)$ matches with shift d iff $s_{i+d} = t_i$ for all $1 \leq i \leq n-d$ and $s_i = t_{i+n-d}$ for all $1 \leq i \leq d$.

We say that a path in a paired de Bruijn graph is *sound with respect to shift d* , or just *sound*, iff the pair of strings it spells matches with shift d . We say that a cycle in a paired de Bruijn graph is sound iff the pair of cyclic strings matches with shift d .

We say that a path or a cycle is *covering* if it includes all the edges in a graph. We say that a set of paths or cycles covers the graph iff every edge of the graph belongs to at least one path or cycle in the set.

A *promise problem* is a kind of decision problem where only inputs from some set of valid inputs are considered. Specifically, a promise problem is defined by

a pair of disjoint sets (S_+, S_-) . A solution to the problem is a program which outputs “yes” when run on inputs in S_+ and outputs “no” when run on inputs in S_- . However, when run on inputs outside of $S_+ \cup S_-$, its behavior may be arbitrary: it may return any result, exceed its allowed time and memory bounds, or even hang.

Note that a promise problem (S_+, S_-) is at most as hard as (S'_+, S'_-) if $S_+ \subseteq S'_+$ and $S_- \subseteq S'_-$, because the solution for the latter problem would solve the former problem as well. Particularly, (S'_+, S'_-) is NP-hard if (S_+, S_-) is NP-hard. Also, an ordinary decision problem defined by set S is the same as the promise problem $(S, \mathbb{C}S)$ (here, \mathbb{C} means set complement).

In the following problems, it would be assumed that the input consists of Σ , an alphabet, G , a paired de Bruijn graph of order k over Σ , as well as 1^d , that is unary coding of d .

3 Trivial Cases

If $|\Sigma| = 1$, a paired de Bruijn graph can have at most one vertex and at most one edge, and every cycle is sound. If $k = 0$, a paired de Bruijn graph can have at most one vertex and at most $|\Sigma|^2$ edges, and the problem is a bit harder. However, it can be solved in polynomial time in the following way: construct a directed graph with one vertex for each element of Σ and edge (u, v) iff there is an edge labeled with (u, v) in the original graph (this new graph may contain loops). Now, there is a sound cycle in the original graph iff there is a cycle in the new graph, and there is a covering sound cycle in the original graph iff there is a set of at most d cycles covering the new graph. Both properties can be easily checked in polynomial time.

4 A Case with Fixed k

Theorem 1. *For any fixed $k \geq 1$, the promise problem (S_+, S_-) , where S_+ is the set of paired de Bruijn graphs which have a covering sound cycle and S_- is the set of paired de Bruijn graphs which do not have a sound cycle, is NP-hard.*

Proof. The proof of this theorem consists of two parts. Firstly, NP-hardness of a specific graph theory problem is proven by reduction from Hamiltonian Cycle problem. Then, the intermediate problem is reduced to the problem formulated in the theorem.

Lemma 1. *The promise problem (S_+, S_-) , where S_+ is the set of undirected graphs with a hamiltonian cycle and S_- is the set of undirected graphs without hamiltonian paths, is NP-hard.*

Proof. First note that the problem is well-defined, because every graph with a hamiltonian cycle has a hamiltonian path. We will start with an instance G of Hamiltonian Cycle problem, which is NP-hard [3]. Without loss of generality, let us assume that G has at least three vertices.

Now build a new graph G' in the following way: firstly, pick a vertex in G and duplicate it together with all the edges incident to it. Let the copies of the vertex be a_1 and b_1 . Now let us duplicate the whole graph, let the first copy be G_1 and the second copy be G_2 , and let the copies of a_1 and b_1 be a_3 and b_3 . Add two new vertices a_2 and b_2 and four new edges $\{a_1, a_2\}$, $\{a_2, a_3\}$, $\{b_1, b_2\}$, and $\{b_2, b_3\}$ (see Figure 1).

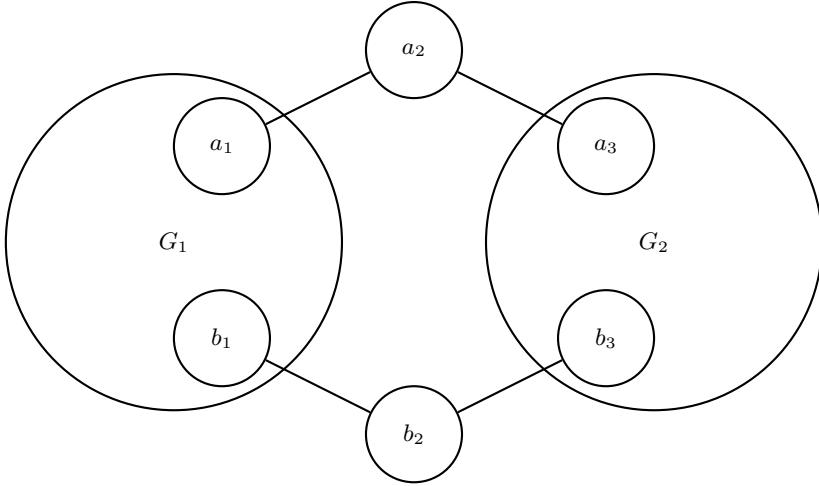


Fig. 1. Graph G'

The following two theorems show that the transformation described above maps all positive instances of hamiltonian cycle problem to S_+ and all negative instances of hamiltonian cycle problem to S_- .

Theorem 2. *If a graph G has a hamiltonian cycle, then the graph G' produced as described above has a hamiltonian cycle.*

Proof. After the vertex in G is duplicated, the cycle in G maps to a hamiltonian path in G_1 from a_1 to b_1 . Analogously, G_2 has a hamiltonian path from a_3 to b_3 . So, the cycle in G' is constructed as follows: start at a_1 , traverse the path in G_1 to b_1 , go to b_2 , then to b_3 , then traverse the path in G_2 to a_3 , then go to a_2 , and return to a_1 .

Theorem 3. *If a graph G does not have hamiltonian cycles, then the graph G' does not have hamiltonian paths.*

Proof. Suppose that, on the contrary, G' contains a hamiltonian path. First consider the case when one end of the path is in G_1 and the other end is in G_2 . Then, the path either traverses edges $\{a_1, a_2\}$ and $\{a_2, a_3\}$ but not $\{b_1, b_2\}$ and $\{b_2, b_3\}$, or $\{b_1, b_2\}$ and $\{b_2, b_3\}$ but not $\{a_1, a_2\}$ and $\{a_2, a_3\}$. In both cases,

either a_2 or b_2 is not visited, so the path is not hamiltonian. So, ends of the path are either both outside G_1 , or both outside G_2 . Let us assume they are outside G_1 , the other case is proved analogously. Besides a_1 and b_1 , G_1 contains at least one internal vertex because of the assumption that G has at least three vertices. To reach that vertex, the path must enter G_1 through a_1 and leave through b_1 (or the opposite, which doesn't matter). Because there are no other ways to enter G_1 , the path enters G_1 only once and traverses all vertices of G_1 . So, the fragment of the path within G_1 , when mapped back to G , becomes a hamiltonian cycle. So, G has a hamiltonian cycle, a contradiction.

Lemma 2. *If a graph has a hamiltonian cycle, then:*

- For each vertex v in the graph, there is a hamiltonian path having v as one of its endpoints.
- For each edge $\{u, v\}$ in the graph, there is a hamiltonian path passing through $\{u, v\}$.
- For each edge $\{u, v\}$ and vertex $w \neq u, v$, there is a hamiltonian path passing through $\{u, v\}$ such that v resides between u and w on the path.

Proof. Let n be the number of vertices in the graph, and let v_1, v_2, \dots, v_n be the vertices numbered in the order of the cycle. Let $u = v_i$ and $v = v_j$, $i < j$ (otherwise, vertices can be renumbered in the reverse order), and let $w = v_k$. Then, the first point of the theorem is obvious, the path for the second point is $v_{j+1}, v_{j+2}, \dots, v_n, v_1, v_2, \dots, v_i, v_j, v_{j-1}, \dots, v_{i+1}$, and the path for the third point is the same if $i < k < j$ and $v_{j-1}, v_{j-2}, \dots, v_i, v_j, v_{j+1}, \dots, v_n, v_1, v_2, \dots, v_{i-1}$ otherwise.

Now return to Theorem 1. First consider the case $k = 1$. Begin with an instance G of the problem from Lemma 1. Let G have n vertices v_1, v_2, \dots, v_n . Without loss of generality, let us assume that $n \geq 3$. Set $d = n + 1$. Now, we are going to construct a paired de Bruijn graph $G' = (V, A)$. It would have block structure: there will be $2n + 2$ blocks $V_1, V_2, \dots, V_{2n+2}$ and $2n + 2$ separator vertices $s_1, s_2, \dots, s_{2n+2}$, so $V = V_1 \cup V_2 \cup \dots \cup V_{2n+2} \cup \{s_1, s_2, \dots, s_{2n+2}\}$ (see Figure 2). This graph will contain edges of three kinds:

- Within a block.
- From s_i to an element of V_i .
- From an element of V_i to s_{i+1} , or from an element of V_{2n+2} to s_1 .

The alphabet would be analogously divided into $2n + 2$ blocks $C_1, C_2, \dots, C_{2n+2}$ and $2n + 2$ separator characters $t_1, t_2, \dots, t_{2n+2}$, so $\Sigma = C_1 \cup C_2 \cup \dots \cup C_{2n+2} \cup \{t_1, t_2, \dots, t_{2n+2}\}$. For each vertex $v \in V_i$, the first component of the associated bilabel will be in C_i , and the second component will be in C_{i+1} (or C_1 if $i = 2n + 2$). Each s_i would be associated with a bilabel (t_i, t_{i+1}) , s_{2n+2} will be associated with a bilabel (t_{2n+2}, t_1) .

The blocks will be formed as follows: the blocks V_1 and V_{2n+2} would be copies of G , while blocks V_2 through V_{2n+1} would each contain two copies of G , except for one vertex of which only one copy would be present. The vertices from the

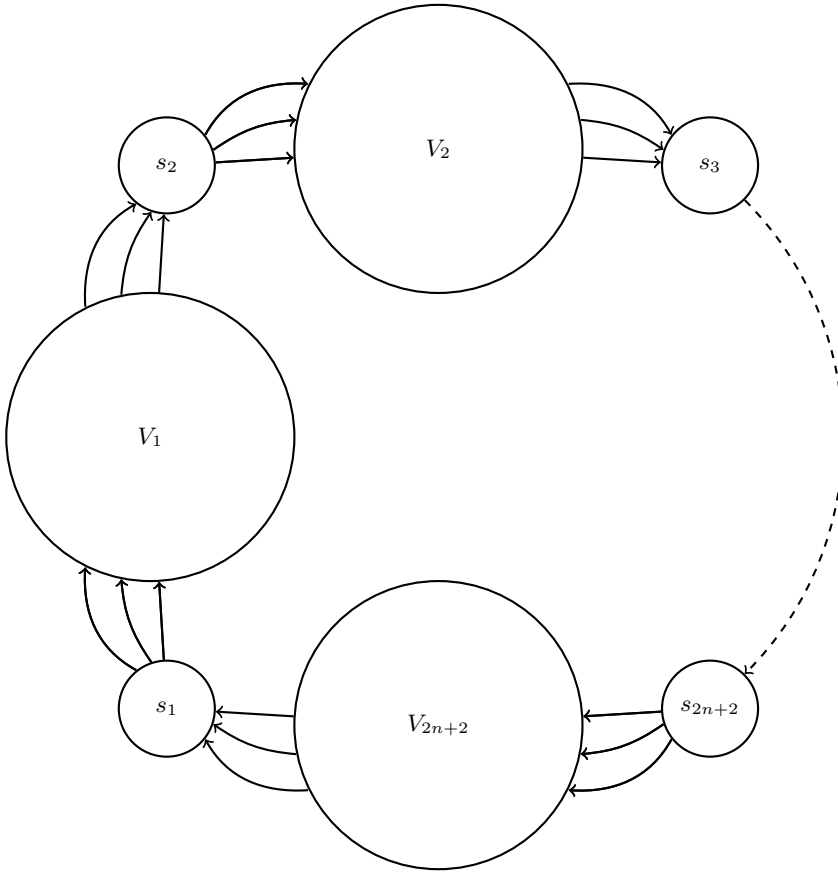


Fig. 2. Structure of the paired de Bruijn graph

first copy would be called $v_{i,j}$, the vertices from the second copy would be called $v''_{i,j}$, and the only copy of $v_{[i/2]}$ in block i would be called $v'_{i,[i/2]}$. The edges would be added such that every path through such block would pass through this vertex.

By assigning a dedicated subset of the alphabet to each block, we prevent vertices from different blocks from being assigned the same bilabel. In fact, it can be seen from the following definition that each vertex is assigned a distinct bilabel, so the assignment is valid. We also note that, with the exceptions of the bilabels containing u , the second index of a character (j in $c_{i,j}$) is the same in both components of a bilabel. This means that in a sound path the sequence of second indices must repeat with a period of d .

The precise definition is as follows:

- For $i = 1, 2n + 2$ block $V_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,n}\}$.
- For $i = 2 \dots 2n + 1$ block $V_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,[i/2]-1}, v_{i,[i/2]+1}, v_{i,[i/2]+2}, \dots, v_{i,n}, v'_{i,[i/2]}, v''_{i,1}, v''_{i,2}, v''_{i,[i/2]-1}, v''_{i,[i/2]+1}, v''_{i,[i/2]+2}, \dots, v''_{i,n}\}$.

- Alphabet block $C_1 = \{u\}$.
- For $i = 1 \dots n + 1$ alphabet block $C_{2i} = \{c_{2i,1}, c_{2i,2}, \dots, c_{2i,n}\}$.
- For $i = 1 \dots n$ alphabet block $C_{2i+1} = \{c_{2i+1,1}, c_{2i+1,2}, \dots, c_{2i+1,i-1}, c_{2i+1,i+1}, c_{2i+1,i+2}, \dots, c_{2i+1,n}, c'_{2i+1,i}, c''_{2i+1,1}, c''_{2i+1,2}, \dots, c''_{2i+1,i-1}, c''_{2i+1,i+1}, c''_{2i+1,i+2}, \dots, c''_{2i+1,n}\}$.
- For $i = 1 \dots n$ the bilabel associated with $v_{1,i}$ is $(u, c_{2,i})$.
- For $i = 1 \dots n$ the bilabel associated with $v_{2n+2,i}$ is $(c_{2n+2,i}, u)$.
- For $i = 2 \dots 2n + 1, j = 1 \dots n, j \neq \lfloor i/2 \rfloor$ the bilabel associated with $v_{i,j}$ is $(c_{i,j}, c_{i+1,j})$.
- For $i = 1 \dots n$ the bilabel associated with $v'_{2i,i}$ is $(c_{2i,i}, c'_{2i+1,i})$.
- For $i = 1 \dots n$ the bilabel associated with $v'_{2i+1,i}$ is $(c'_{2i+1,i}, c_{2i+2,i})$.
- For $i = 1 \dots n, j = 1 \dots n, j \neq i$ the bilabel associated with $v''_{2i,j}$ is $(c_{2i,j}, c''_{2i+1,j})$.
- For $i = 1 \dots n, j = 1 \dots n, j \neq i$ the bilabel associated with $v''_{2i+1,j}$ is $(c''_{2i+1,j}, c_{2i+2,j})$.

The edges are added:

- For $i = 1 \dots n$ the edges $(s_1, v_{1,i}), (v_{1,i}, s_2), (s_{2n+2}, v_{2n+2,i})$, and $(v_{2n+2,i}, s_1)$.
- For $i = 2 \dots 2n + 1, j = 1 \dots n, j \neq \lfloor i/2 \rfloor$ the edges $(s_i, v_{i,j})$ and $(v'_{i,j}, s_{i+1})$.
- For $i = 2 \dots 2n + 1$ the edges $(s_i, v'_{i, \lfloor i/2 \rfloor})$ and $(v'_{i, \lfloor i/2 \rfloor}, s_{i+1})$.

Also, for each edge $\{v_i, v_j\}$ in G ($1 \leq i \leq n, 1 \leq j \leq n, i \neq j$) the following edges are added:

- The edges $(v_{1,i}, v_{1,j}), (v_{2n+2,i}, v_{2n+2,j}), (v_{2j,i}, v'_{2j,j}), (v_{2j+1,i}, v'_{2j+1,j}), (v'_{2i,i}, v'_{2i,j})$, and $(v'_{2i+1,i}, v''_{2i+1,j})$.
- For $r = 2 \dots 2n + 1, i \neq \lfloor r/2 \rfloor, j \neq \lfloor r/2 \rfloor$ the edges $(v_{r,i}, v_{r,j})$ and $(v''_{r,i}, v''_{r,j})$.

Note that, as edges of G are undirected, each edge should be processed twice, once as $\{v_i, v_j\}$ and once as $\{v_j, v_i\}$. The bilabels associated with the edges can be unambiguously determined from the bilabels associated with their ends.

The size of G' and the parameter d is polynomial in terms of n by construction. The following two theorems show that the transformation described above maps all positive instances of the problem formulated in Lemma 1 to paired de Bruijn graphs with covering sound cycles and all negative instances of the problem formulated in Lemma 1 to paired de Bruijn graphs without sound cycles.

Theorem 4. *If a graph G has a hamiltonian cycle, then the paired de Bruijn graph G' produced as described above has a covering sound cycle.*

Proof. Remember that $d = n + 1$. Construct the cycle as follows: first, select a hamiltonian path in G , let it be $v_{p_1}, v_{p_2}, \dots, v_{p_n}$. Start at s_1 , then go to $v_{1,p_1}, v_{1,p_2}, \dots, v_{1,p_n}$. Then, for each i from 2 to $2n + 1$, visit s_i , then $v_{i,p_1}, v_{i,p_2}, \dots, v_{i,p_{r_{\lfloor i/2 \rfloor}-1}}$, where $r_{\lfloor i/2 \rfloor}$ is such that $p_{r_{\lfloor i/2 \rfloor}} = \lfloor i/2 \rfloor$, then $v'_{i,p_{r_{\lfloor i/2 \rfloor}}} = v'_{i, \lfloor i/2 \rfloor}$, then $v''_{i,p_{r_{\lfloor i/2 \rfloor}+1}}, v''_{i,p_{r_{\lfloor i/2 \rfloor}+2}}, \dots, v''_{i,p_n}$. After that, visit $s_{2n+2}, v_{2n+2,p_1}, v_{2n+2,p_2}, \dots, v_{2n+2,p_n}$, and finally return to s_1 .

This cycle visits each block, and the sequence of second indices within each block is the same (it is p_1, p_2, \dots, p_n), therefore, from the construction, the cycle is sound. However, it is not necessarily covering. To make a covering cycle, first use the procedure described above to construct one cycle per every property from Lemma 2, namely:

- For every vertex v_i , use the path having v_i as an endpoint to construct cycles passing through $(s_j, v_{j,i})$ ($1 \leq j \leq 2n + 2, i \neq \lfloor j/2 \rfloor$), $(s_{2i}, v'_{2i,i})$, $(s_{2i+1}, v'_{2i+1,i})$, $(v_{1,i}, s_2)$, $(v_{2n+2,i}, s_1)$, $(v'_{2i,i}, s_{2i+1})$, $(v'_{2i+1,i}, s_{2i+2})$, and $(v''_{j,i}, s_{j+1})$ ($2 \leq j \leq 2n + 1, i \neq \lfloor j/2 \rfloor$).
- For every edge $\{v_i, v_j\}$, use the path passing through $\{v_i, v_j\}$ to construct cycles passing through $(v_{r,i}, v_{r,j})$ ($r = 1, 2n + 2$), $(v_{2j,i}, v'_{2j,j})$, $(v_{2j+1,i}, v'_{2j+1,j})$, $(v'_{2i,i}, v''_{2i,j})$, and $(v'_{2i+1,i}, v''_{2i+1,j})$.
- For every edge $\{v_i, v_j\}$ and vertex v_k ($k \neq i, j$), use the path passing through $\{v_i, v_j\}$, such that v_j resides between v_i and v_k on the path, to construct cycles passing through $(v_{2k,i}, v_{2k,j})$, $(v_{2k+1,i}, v_{2k+1,j})$, $(v''_{2k,j}, v''_{2k,i})$, and $(v''_{2k+1,j}, v''_{2k+1,i})$.

Together, these cycles should cover all the edges of G' . To make a single covering cycle, cut all these cycles at s_1 and join them together. The resulting cycle is sound because the second component of every bilabel from V_{2n+2} is u , and the first component of every bilabel from V_1 is also u , so they always match.

Theorem 5. *If a graph G doesn't have hamiltonian paths, then the paired de Bruijn graph G' produced as described above doesn't have sound cycles.*

Proof. Within each block, the set of characters used for the first component of bilabels and the set of characters used for the second component of the bilabel do not intersect. Therefore, every contiguous segment of a sound cycle within a single block must have length at most d . Because the blocks are connected in a circle (see Figure 2), and the cycle cannot be contained within a single block, it must pass around the circle at least once. Therefore, it must pass through s_1 . Exactly d vertices later, it must pass through s_2 , as it is the only vertex with a matching bilabel. The $d - 1 = n$ vertices between s_1 and s_2 must be spent within V_1 , as the only other way to get to s_2 is to pass around the whole circle at least once, and the circle is longer than d , so this is impossible. Then it must pass through $s_3, V_3, s_4, V_4, \dots, s_{2n+2}, V_{2n+2}$, then return to s_1 .

Let us call a segment between successive visits to s_1 a pass. Within a pass, each block is visited exactly once, and a path within each block has length n . Moreover, every pair of consecutive blocks, except (V_{2n+2}, V_1) , has their vertices labeled such that the sequences of second indices within each block must be the same. However, for each i , such that $1 \leq i \leq n$, the structure of blocks V_{2i} and V_{2i+1} requires the sequence of second indices to include i , as it is impossible to pass through these blocks otherwise. Therefore, the sequence must include every value from 1 to n , so it is a permutation. Since every edge in G' within a block corresponds to an edge in G , the permutation defines a hamiltonian path in G , a contradiction.

The case $k > 1$ is handled as follows: first, produce a graph G over an alphabet Σ for the case $k = 1$. Then, construct a new alphabet Σ' as being equal to $\Sigma \cup \{f\}$, where f is a new character. After that, construct a new graph G' from G by replacing each vertex labeled (a, b) with k' vertices labeled $(f^{k'-1}a, f^{k'-1}b), (f^{k'-2}af, f^{k'-2}bf), \dots, (af^{k'-1}, bf^{k'-1})$ and $k' - 1$ edges labeled $(f^{k'-1}af, f^{k'-1}bf), (f^{k'-2}aff, f^{k'-2}bff), \dots, (faf^{k'-1}, fbf^{k'-1})$, and replacing each edge labeled with (ab, cd) with an edge labeled $(af^{k'-1}b, cf^{k'-1}d)$. Finally, set d' equal $k'd$. Now, every sound cycle in can be unambiguously mapped from G to G' and vice versa. Therefore, the new solution is equivalent to the old one.

These immediately follow from Theorem 1:

Corollary 1. *The problem of checking whether a paired de Bruijn graph contains a sound cycle is NP-hard, both in general case and for any fixed $k \geq 1$.*

Corollary 2. *The problem of checking whether a paired de Bruijn graph contains a covering sound cycle is NP-hard, both in general case and for any fixed $k \geq 1$.*

5 A Case with Fixed $|\Sigma|$

Theorem 6. *For any fixed $|\Sigma| \geq 2$, the promise problem (S_+, S_-) , where S_+ is the set of paired de Bruijn graphs which have a covering sound cycle and S_- is the set of paired de Bruijn graphs which don't have a sound cycle, is NP-hard.*

Proof. This is proven by reduction from the same problem with fixed $k = 1$. Let the instance with $k = 1$ be G , and let its alphabet be Σ . We are going to build an instance G' of the same problem with alphabet $\Sigma' = \{0, 1\}$. Set $l = \lceil \log_2 |\Sigma| \rceil$. Now, every character from Σ can be unambiguously encoded with l binary digits. Take that encoding, and replace each digit 0 with the sequence 01, and each digit 1 with the sequence 10. The resulting encoding of length $2l$ has the following properties: it does not contain repetitions of three or more of the same digit as a substring, and it does not begin or end with a repeated digit. Set $k' = 4l + 5$. Let $\text{enc}(c)$ denote the $2l$ -character encoding of c described above. Then, for each vertex in G labeled (a, b) , add a vertex labeled $(\text{enc}(a)01110\text{enc}(a), \text{enc}(b)01110\text{enc}(b))$. Here, the sequence 111 unambiguously determines the center of the encoding of a character. Each edge from G is translated to $4l + 9$ new vertices and $4l + 10$ new edges: if the original edge has the bilabel (ab, cd) , the bilabels of the new vertices and edges will spell $(\text{enc}(a)01110\text{enc}(a)10001\text{enc}(b)01110\text{enc}(b), \text{enc}(c)01110\text{enc}(c)10001\text{enc}(d)01110\text{enc}(d))$. Each bilabel would include at least one of the marker sequences 000 and 111 and at least one complete encoding of a character, so there will be no undesired overlaps. It can be shown that each sound cycle from G can be mapped to G' and vice versa, so they are equivalent for the purposes of the problem.

These immediately follow from Theorem 6:

Corollary 3. *The problem of checking whether a paired de Bruijn graph contains a sound cycle is NP-hard for any fixed $|\Sigma| \geq 2$.*

Corollary 4. *The problem of checking whether a paired de Bruijn graph contains a covering sound cycle is NP-hard for any fixed $|\Sigma| \geq 2$.*

6 A Case with Both k and $|\Sigma|$ Fixed

If both k and $|\Sigma|$ are fixed, the number of possible paired de Bruijn graphs is limited: there are at most $|\Sigma|^{2k}$ different vertex bilabels, and at most $|\Sigma|^{2k+2}$ different edge bilabels, and each bilabel is used by at most one vertex or edge, so the total number of different paired de Bruijn graphs is limited by a number which only depends on k and $|\Sigma|$. Let us denote this number by N .

There are at most N different problem instances for each instance length: otherwise, there would be two different instances having the same graph and the same length, but such instances can only differ in d , which is represented in unary coding, so any instances which only differ in d must have different length. Therefore, the number of instances is polynomial in instance length, so the language defined by the problem is *sparse*. Unless $P=NP$, a sparse language is never NP-hard [4]. Therefore, the problem of checking whether a paired de Bruijn graph has a sound cycle cannot be NP-hard if both k and $|\Sigma|$ are fixed.

7 Conclusion

We have proved that the Paired de Bruijn Sound Cycle problem is NP-hard in general case. Results of this work combined with previous works on genome assembly complexity show that all known models for genome assembly both from single and mate-pair reads are NP-hard.

However, the problem considered in this paper has a special case with both k and $|\Sigma|$ fixed which is not NP-hard unless $P=NP$. A reasonable direction of future research is to determine if this case is solvable in polynomial time.

References

1. Galant, J., Maier, D., Astorer, J.: On finding minimal length superstrings. *Journal of Computer and System Sciences* 20(1), 50–58 (1980)
2. Kapun, E., Tsarev, F.: De Bruijn superwalk with multiplicities problem is NP-hard. *BMC Bioinformatics* 14(suppl. 5), S7 (2013)
3. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W., Bohlinger, J.D. (eds.) *Complexity of Computer Computation. The IBM Research Symposia Series*, pp. 85–103. Plenum Press (1972)
4. Mahaney, S.R.: Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences* 25(2), 130–143 (1982)

5. Medvedev, P., Brudno, M.: Maximum likelihood genome assembly. *Journal of Computational Biology* 16(8), 1101–1116 (2009)
6. Medvedev, P., Georgiou, K., Myers, G., Brudno, M.: Computability of models for sequence assembly. In: Giancarlo, R., Hannenhalli, S. (eds.) *WABI 2007. LNCS (LNBI)*, vol. 4645, pp. 289–301. Springer, Heidelberg (2007)
7. Medvedev, P., Pham, S., Chaisson, M., Tesler, G., Pevzner, P.: Paired de Bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology* 18(11), 1625–1634 (2011)
8. Pevzner, P.A., Tang, H., Waterman, M.S.: An eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences* 98(17), 9748–9753 (2001)
9. Pham, S.: Mate-pair consistency and generating problems. In: *Talk at RECOMB Satellite Conference on Open Problems in Algorithmic Biology* (2012)

Accurate Decoding of Pooled Sequenced Data Using Compressed Sensing

Denisa Duma¹, Mary Wootters², Anna C. Gilbert²,
Hung Q. Ngo³, Atri Rudra³, Matthew Alpert¹,
Timothy J. Close⁴, Gianfranco Ciardo¹, and Stefano Lonardi¹

¹ Dept. of Computer Science and Eng., University of California, Riverside, CA 92521

² Dept. of Mathematics, The University of Michigan, Ann Arbor, MI 48109

³ Dept. of Computer Science and Eng., Univ. at Buffalo, SUNY, Buffalo, NY 14260

⁴ Dept. of Botany & Plant Sciences, University of California, Riverside, CA 92521

Abstract. In order to overcome the limitations imposed by DNA barcoding when multiplexing a large number of samples in the current generation of high-throughput sequencing instruments, we have recently proposed a new protocol that leverages advances in combinatorial pooling design (group testing) [9]. We have also demonstrated how this new protocol would enable *de novo* selective sequencing and assembly of large, highly-repetitive genomes. Here we address the problem of decoding pooled sequenced data obtained from such a protocol. Our algorithm employs a synergistic combination of ideas from compressed sensing and the decoding of error-correcting codes. Experimental results on synthetic data for the rice genome and real data for the barley genome show that our novel decoding algorithm enables significantly higher quality assemblies than the previous approach.

Keywords: second/next-generation sequencing, pooled sequencing, compressed sensing, error-correcting codes.

1 Introduction

The second generation of DNA sequencing instruments offer unprecedented throughput and extremely low cost per base, but read lengths are much shorter compared to Sanger sequencing. An additional limitation is the small number of distinct samples that these instruments can accommodate (e.g., two sets of eight lanes on the Illumina HiSeq). When the sequencing task involves a large number of individual samples, a common solution is to employ DNA barcoding to “multiplex” samples within a single lane. DNA barcoding, however, does not scale readily to thousands of samples. As the number of samples reaches the hundreds, exhaustive DNA barcoding becomes time consuming, error-prone, and expensive. Additionally, the resulting distribution of reads for each barcoded sample can be severely skewed (see, e.g., [1]).

Combinatorial pooling design or *group testing* allows one to achieve multiplexing without exhaustive barcoding. In group testing, a *design* or *scheme* is

a set of tests (or pools) each of which is a subset of a large collection of items that needs to be tested for the presence of (a few) ‘defective’ items. The result of testing a pool is a boolean value indicating whether the pool contains at least one defective. The goal of group testing is to *decode* the information obtained by testing all the pools in order to determine the precise identity of the defectives, despite the fact that the defectives and non-defectives are mixed together. The challenge is to achieve this goal while, at the same time, minimizing the number of pools needed. Recently, *compressed sensing* (CS) has emerged as a powerful technique for solving the decoding problem when the results of testing the pools are more than boolean outcomes, for instance, real or complex values.

Combinatorial pooling has been used previously in the context of genome analysis (see, e.g., [5–7, 2, 12]), but not for *de novo* genome sequencing. Our proposed pooling method for genome sequencing and assembly was first described in [9] and has generated considerable attention. It was used to produce one of the critical datasets for the first draft sequence of the barley genome [14]. In our sequencing protocol, thousands of BAC clones are pooled according to a combinatorial design so that, at the outset of sequencing, one can ‘decode’ each read to its source BACs. The underlying idea is to encode the identity of a BAC within the pooling pattern rather than by its association with a specific DNA barcode. We should stress that combinatorial pooling is not necessarily an alternative to DNA barcoding, and both methods have advantages and disadvantages. They can be used together to increase the number of samples that can be handled and benefit from the advantages of both.

In this paper we address the problem of decoding pooled sequenced data obtained from a protocol such as the one in [9]. While the main objective is to achieve the highest possible accuracy in assigning a read to the correct BAC, given that one sequencing run can generate hundreds of millions of reads, the decoding procedure has also to be time- and space-efficient. Since in [9] we pooled BAC clones according to the *Shifted Transversal Design* [15] which is a *Reed-Solomon* based pooling design, our proposed decoding approach combines ideas from the fields of compressive sensing and decoding of error-correcting codes. Specifically, given the result of ‘testing’ (in this case, sequencing) pools of genomic BAC clones, we aggregate read frequency information across the pools and cast the problem as a compressed sensing problem where the unknowns are the BAC assignments of the reads. We solve (decode) for the unknown assignments using a *list recovery* strategy as used in the decoding of error-correcting codes. Reed-Solomon codes are known to be good list-recoverable codes which can also tolerate a large fraction of errors. We also show that using readily available information about the reads like overlap and mate pair information can improve the accuracy of the decoding. Experimental results on synthetic reads from the rice genome as well as real sequencing reads from the barley genome show that the decoding accuracy of our new method is almost identical to that of HASHFILTER [9]. However, when the assembly quality of individual BAC clones is the metric of choice, the decoding accuracy of the method proposed here is significantly better than HASHFILTER.

2 Related Work

The resemblance between our work and the closest related research efforts using combinatorial pooling and compressed sensing ideas stops at the pooling of sequencing data. Our application domain, pooling scheme employed and algorithmic approach to decoding, are completely different. To the best of our knowledge, all compressed sensing work in the domain of genomics deals with the problem of *genotyping* large population samples, whereas our work deals with *de novo* genome sequencing. For instance in [5], the authors employ a pooling scheme based on the Chinese Remainder Theorem (CRT) to identify carriers of rare alleles in large cohorts of individuals. The pooling scheme allows the detection of mutants within a pool, and by combining information across pools one is able to determine the identity of carriers. In true group testing style, the unknown carrier identities are encoded by a boolean vector of length equal to the number of individuals, where a value of one indicates a carrier and zero a normal individual. To decode their pooling scheme and find the unknown vector, the authors devise a greedy decoding method called *Minimum Discrepancy Decoder*. In [6], loopy belief propagation decoding is used for the same pooling scheme. A similar application domain is described in [12], where the authors identify carriers of rare SNPs in a group of individuals pooled with a random pooling scheme (Bernoulli matrix) and use the *Gradient Projection for Sparse Reconstruction* (GPSR) algorithm to decode the pooling scheme and recover the unknown carrier identities. The same problem is tackled in [11] with a pooling design inspired from the theory of error correcting codes. However, this design is only able to identify a single rare-allele carrier within a group. In [2], the authors organize domain-specific (linear) constraints into a compressed sensing matrix which they use together with GPSR decoding to determine the frequency of each bacterial species present in a metagenomic mixture.

3 Preliminaries

As mentioned in the introduction, in [9] we pool DNA samples (BAC clones) according to a combinatorial pooling scheme, then sequence the pools using high-throughput sequencing instruments. In this paper we show how to efficiently recover the sequence content of each BAC by combining ideas from the theory of *sparse signal recovery* or *compressed sensing* (CS) as well as from the large body of work developed for the decoding of *error-correcting codes*.

Formally, a combinatorial pooling design (or pooling scheme) can be represented by a binary matrix Φ with m rows (corresponding to pools) and n columns (corresponding to items to be pooled), where entry (i, j) is 1 if item j is present in pool i , 0 otherwise. The matrix Φ is called the *design matrix*, *sensing matrix* or *measurement matrix* by various authors in the literature. In this paper we only use the first two names to designate Φ . An important property of a combinatorial pooling design is its *decodability* d (also called *disjunctness*), which is the maximum number of ‘defectives’ it guarantees to reliably identify. Let w be

a subset of the columns (pooled variables) of the design matrix Φ and $p(w)$ be the set of rows (pools) that contain at least one variable in w : the matrix Φ is said to be d -decodable (d -disjunct) if for any choice of w_1 and w_2 with $|w_1| = 1$, $|w_2| = d$ and $w_1 \not\subseteq w_2$, we have that $p(w_1) \not\subseteq p(w_2)$.

In this paper, we pool BACs using the combinatorial pooling scheme called *Shifted Transversal Design* (STD) [15]. STD is a *layered* design, *i.e.*, the rows of the design matrix are organized into multiple redundant layers such that each pooled variable appears only once in each layer, that is, a *layer* is a partition of the set of variables. STD is defined by parameters (q, L, Γ) where L is the number of layers, q is a prime number equal to the number of pools (rows) in each layer and Γ is the *compression level* of the design. Thus, in order to pool n variables, STD uses a total of $m = q \times L$ pools. The set of L pools defines a unique pooling pattern for each variable which can be used to retrieve its identity. This set of L integers is called the *signature* of the variable. The compression level Γ is defined to be the smallest integer such that $q^{\Gamma+1} \geq n$. STD has the desirable property that any two variables co-occur in at most Γ pools, therefore by choosing a small value for Γ one can make STD pooling extremely robust to errors. The parameter Γ is also related to the decodability of the design through the equation $d = \lfloor (L - 1)/\Gamma \rfloor$. Therefore, Γ can be seen as a trade-off parameter: the larger it is, the more items can be tested (up to $q^{\Gamma+1}$), but fewer defectives can be reliably identified (up to $\lfloor (L - 1)/\Gamma \rfloor$). For more details on the pooling scheme and its properties please refer to [15].

In order to decode measurements obtained through STD (*i.e.*, reconstruct the sequence content of pooled BACs) we borrow ideas from compressed sensing (CS), an area of signal processing that describes conditions and efficient methods for capturing sparse signals from a small number of aggregated measurements [6]. Unlike combinatorial group testing, in compressed sensing measurements can be more general than boolean values, allowing recovery of hidden variables which are real or complex-valued. Specifically, in CS we look for an unknown vector or *signal* $\mathbf{x} = (x_1, x_2, \dots, x_n)$ which is *s-sparse*, *i.e.*, has at most s non-zero entries. We are given a vector $\mathbf{y} = (y_1, y_2, \dots, y_m)$ of measurements ($m \ll n$), which is the product between the (known) design matrix Φ and the unknown vector \mathbf{x} , that is $\mathbf{y} = \Phi\mathbf{x}$. Under certain conditions on Φ , by using the measurements \mathbf{y} , the assumption on the sparsity of \mathbf{x} and information encoded by Φ , it is possible to recover the original sparse vector \mathbf{x} . The latter equation corresponds to the ideal case when the data is noise-free. In practice, if the signal \mathbf{x} is not as sparse as needed and if measurements are corrupted by noise, the equation becomes $\mathbf{y} = \Phi\mathbf{x} + \epsilon$. In CS theory there are two main approaches for solving the latter equation, namely *linear programming* (LP) decoding and *greedy pursuit* decoding. Greedy pursuit algorithms have faster decoding time than LP-based approaches, frequently sub-linear in the length of \mathbf{x} (although for specially designed matrices). Their main disadvantage is that they usually require a slightly larger number of measurements and do not offer the same uniformity and stability guarantees as LP decoding. Greedy pursuits are iterative algorithms which proceed in a series of steps: (1) identify the locations of the

largest coefficients of \mathbf{x} by greedy selection, (2) estimate their values, (3) update \mathbf{y} by subtracting the contribution of estimated values from it, and iterate (1-3) until some convergence criterion is met. Usually $O(s)$ iterations, where s is the sparsity of \mathbf{x} , suffice [17]. Updating \mathbf{y} amounts to solving a least squares problem in each iteration.

The most well known greedy decoding algorithm is *Orthogonal Matching Pursuit* (OMP) [16], which has spawned many variations. In OMP, the greedy rule selects in each iteration the largest coordinate of $\Phi^T \mathbf{y}$, *i.e.*, the column of Φ which is the most correlated with \mathbf{y} . In this paper, we are interested in a variant of OMP called *Simultaneous Orthogonal Matching Pursuit* (S-OMP). S-OMP is different from OMP in that it approximates multiple sparse signals $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_K$ simultaneously by using multiple linear combinations, $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_K$, of the sensing matrix Φ [17]. The unknown signals $\{\mathbf{x}_k\}_{k \in \{1, \dots, K\}}$ as well as measurement vectors $\{\mathbf{y}_k\}_{k \in \{1, \dots, K\}}$ can be represented by matrices $\mathbf{X} \in \mathcal{R}^{n \times K}$ and $\mathbf{Y} \in \mathcal{R}^{m \times K}$. Intuitively, by jointly exploiting information provided by \mathbf{Y} , S-OMP is able to achieve better approximation error especially when the signals to be approximated are corrupted by noise which is not statistically independent [17].

The mapping from the CS setting into our problem follows naturally and we give here a simplified and intuitive version of it. The detailed model will be introduced in the next section. The variables to be pooled are BAC clones. Each column of the design matrix corresponds to a BAC to be pooled and each row corresponds to a pool. For each read r (to be decoded) there is an unknown s -sparse vector \mathbf{x} which represents at most s BACs which could have generated r . The vector of measurements \mathbf{y} (*frequency vector*) of length m gives for each read r , the number of times r appears in each of the m pools. The use of numerical measurements (read counts) rather than boolean values indicating the presence or the absence of r from a pool is in accordance with CS theory and offers additional valuable information for decoding. To carry out the latter, we use a S-OMP style algorithm but replace the greedy selection rule by a *list recovery* criterion. Briefly, we obtain a list of candidate BACs for read r as those columns of Φ whose non-zero coordinates consistently correspond to the heaviest-magnitude measurements in each layer of \mathbf{y} [10]. This allows for a finer-grained usage of the values of \mathbf{y} on a layer-by-layer basis rather than as a whole. Additionally, by requiring that the condition holds for at least l layers with $l \leq L$, one can make the algorithm more robust to the noise in vector \mathbf{y} .

4 Decoding Algorithms

In this section we present our decoding algorithms that assign reads back to the BACs from which they were derived. Recall that we have n BACs pooled into m pools according to STD and each BAC is pooled in exactly L pools. The input data to the decoding algorithm consists of (1) m datasets containing the reads obtained from sequencing the m pools, and (2) the parameters of the pooling design, including the signatures of all n BACs. We will assume that each read

r may originate from up to s BACs with $s \ll n$; ideally, we can make the same assumption for each k -mer (a k -mer is a substring of r of length k) of r , provided that k is ‘large enough’. In practice, this will not be true for all k -mers (e.g., some k -mers are highly repetitive), and we will address this issue later in this document.

We start by preprocessing the reads to correct sequencing errors in order to improve the accuracy of read decoding. For this task, we employ SGA [13], which internally employs a k -mer based error correction strategy. An additional benefit of error correction is that it reduces the total number of distinct k -mers present in the set of reads. After the application of SGA, there still remains a small proportion of erroneous k -mers, which we discard because they will likely introduce noise in the decoding process. An advantage of pooled sequencing is that erroneous k -mers are easy to identify because they appear in fewer than L pools. To be conservative, we only discard k -mers appearing in fewer than γ pools where $\gamma \leq L$ is a user-defined parameter (see Section 5.1 for details on the choice of this parameter). The closer γ is to L the more likely it is that a k -mer that appears in γ pools is correct, but missing from the remaining $L - \gamma$ pools due to sequencing errors. Henceforth, we will call a k -mer *valid* if it appears in a number of pools in the range $[\gamma, sL]$ where s is the sparsity parameter. Any k -mer occurring in more than sL pools is considered highly repetitive, and will likely not be useful in the decoding process. The decoding algorithm we employ can safely ignore these repetitive k -mers.

To carry out the decoding, we first compute the frequencies of all the k -mers in all the m pools. Specifically, we decompose all SGA-corrected reads into k -mers by sliding a window of length k (there are $|r| - k + 1$ such windows for each read r). For each distinct k -mer, we count the number of times it appears in each of the m pools, and store the sequence of the k -mer along with its vector of m counts into a hash table. We refer to the vector of counts of a k -mer as its *frequency vector*.

We are now ready to apply our CS-style decoding algorithm. We are given a large number of reads divided into m sets (pools). For each read r , we want to determine which of the n BACs is the source. Since we decomposed r into its constitutive k -mers, we can represent the pool counts of all its k -mers by a *frequency matrix* \mathbf{Y}_r . Matrix \mathbf{Y}_r is a non-negative integer matrix where the number of columns is equal to the number K_r of k -mers in r , the number of rows is equal to the numbers m of pools, and entry (i, j) reports the number of times the j^{th} k -mer of r appears in pool i . The input to the decoding algorithm for read r is given by (1) the frequency matrix \mathbf{Y}_r , (2) the design matrix $\Phi \in \{0, 1\}^{m \times n}$, and (3) the maximum number s of BACs which could have generated r . To decode r means to find a matrix $\mathbf{X}_r \in \mathbf{Z}^{n \times K_r}$ such that $\mathbf{X}_r = \operatorname{argmin}_{\mathbf{X}} \|\Phi \mathbf{X} - \mathbf{Y}_r\|_2$ with the constrain that \mathbf{X}_r is row-sparse, *i.e.*, it has at most s non-zero rows (one for each source BAC).

Since finding the source BACs for a read is sufficient for our purposes, we can reduce the problem of finding matrix \mathbf{X} to the problem of finding its *row support* $S(\mathbf{X})$, which is the union of the supports of its columns. The support $\operatorname{Supp}(\mathbf{X}_{:,j})$

of a column j of \mathbf{X} is the set of indices i such that $\mathbf{X}_{i,j} \neq 0$. In our case, the non-zero indices represent the set of BACs which generated the read (and by transitivity its constitutive k -mers). Since this set has cardinality at most s , in the ideal case, \mathbf{X} is row-sparse with support size at most s . In practice, the same k -mer can be shared by multiple reads and therefore the number of non-zero indices can differ from s . By taking a conservative approach, we search for a *good* s -sparse approximation of $S(\mathbf{X})$, whose quality we evaluate according to the following definition.

Definition: A non-empty set S is *good* for \mathbf{X} if for any column j of \mathbf{X} , we have $S \subset \text{Supp}(\mathbf{X}_{:,j})$.

Our decoding Algorithm 1 finds S in two steps, namely FILTER and ESTIMATE, which are explained next.

FILTER (Algorithm 2) is a one-iteration S-OMP style algorithm in which multiple candidate BACs are selected (we tried performing multiple iterations without significant improvement in the results). Whereas S-OMP selects one BAC per iteration as the column of Φ most correlated (inner product) with all the columns of \mathbf{Y}_r , our algorithm employs a list recovery criterion to obtain an approximation $\tilde{\mathbf{X}}_r$ of \mathbf{X}_r . Specifically, for each column y of \mathbf{Y}_r and for each layer $l \in [1, \dots, L]$, we select a set S_l of candidate pools for that layer as follows. We choose set S_l by considering the h highest-magnitude coordinates of y in layer l and selecting the corresponding pools. BACs whose signature pools belong to all L sets S_l are kept while the rest of them are removed, *i.e.*, their $\tilde{\mathbf{X}}$ -entries are set to zero. Finally, for the BACs that are not filtered out, the $\tilde{\mathbf{X}}$ -entry estimate follows the *min-count* estimate. The value of h should be chosen to be $\Theta(s)$: $h = 3s$ is sufficient even for noisy data [10].

Next, the ESTIMATE (Algorithm 3) algorithm determines S_r by computing a score for each BAC. Based on the computed scores, we select and return the top s BACs as the final support S_r of \mathbf{X}_r . Read r is then assigned to all the BACs in S_r . The scoring function we employ for each BAC b is the number of k -mers “voting” for b , *i.e.*, having a frequency of at least τ in each pool in the signature of b . The value we used for τ is given in Section 5. If we consider the rows of $\tilde{\mathbf{X}}_r$ as vectors of length K_r , our scoring function is simply the l_0 norm of these vectors, after zeroing out all the entries smaller than τ . We also tried

Algorithm 1. FINDSUPPORT (Φ, \mathbf{Y}_r, h, s)

Input : $\Phi \in \{0, 1\}^{m \times n}$, $\mathbf{Y}_r \in \mathbf{N}^{m \times K_r}$ and sparsity s such that
 $\mathbf{X}_r = \text{argmin}_{\mathbf{X}} \|\Phi \mathbf{X} - \mathbf{Y}_r\|_2$ for a s -row-sparse matrix $\mathbf{X}_r \in \mathbf{N}^{n \times K_r}$;
 $h \leq q$ the number of entries per layer considered by list recovery

Output: A non-empty set S_r with $|S_r| \leq s$ which is *good* for \mathbf{X}_r .

- 1 $\tilde{\mathbf{X}}_r \leftarrow \text{FILTER}(\Phi, \mathbf{Y}_r, h)$
 - 2 $S_r \leftarrow \text{ESTIMATE}(\tilde{\mathbf{X}}_r, s)$
 - 3 **return** S_r
-

Algorithm 2. FILTER(Φ, \mathbf{Y}_r, h)

Input : $\Phi \in \{0, 1\}^{m \times n}$, $\mathbf{Y}_r \in \mathbf{N}^{m \times K_r}$, parameter h
Output: An approximation $\tilde{\mathbf{X}}_r$ for \mathbf{X}_r

- 1 // Recall that Φ has L layers with q pools each
- 2 // For a column y of \mathbf{Y}_r , denote by $y[l]_i$ the i^{th} entry in layer l
- 3 $\tilde{\mathbf{X}}_r \leftarrow 0$
- 4 **for** $k = 1, \dots, K_r$ **do**
- 5 Let $y = \mathbf{Y}_{r, :k}$ be the k^{th} column of \mathbf{Y}_r
- 6 **for** $l = 1, \dots, L$ **do**
- 7 $S_l \leftarrow$ set of h indices $i \in \{1, \dots, q\}$ such that the corresponding counts
 $y[l]_i$ are the h heaviest-magnitude counts in layer l of column y
- 8 **for** $b = 1, \dots, n$ **do**
- 9 layersMatched $\leftarrow 0$
- 10 Let $\phi = \Phi_{:, b}$ be the b^{th} column of Φ
- 11 **for** $l = 1, \dots, L$ **do**
- 12 **if** the unique i such that $\phi[l]_i = 1$ belongs to S_l **then**
- 13 layersMatched \leftarrow layersMatched + 1
- 14 **if** layersMatched = L **then**
- 15 $\tilde{\mathbf{X}}_{b, k} \leftarrow \min_{\phi_p=1} \{y_p\}$

l_1 and l_2 norms without observing significant improvements in the accuracy of read assignments.

Observe that algorithms FINDSUPPORT, FILTER and ESTIMATE process one read at a time. Since there is no dependency between the reads, processing multiple reads in parallel is trivial. However, better total running time, improved decoding accuracy as well as a smaller number of non-decodable reads can be achieved by jointly decoding multiple reads at once. The idea is to use additional sources of information about the reads, namely (1) read overlaps and (2) mate-pair information. For the former, if we can determine clusters of reads that are mutually overlapping, we can then decode all the reads within a cluster as a single unit. Not only this strategy increases the decoding speed, but it also has

Algorithm 3. ESTIMATE($\tilde{\mathbf{X}}_r, s$)

Input : $\tilde{\mathbf{X}}_r$, sparsity parameter s
Output: Support set S_r , with $|S_r| \leq s$

- 1 **for** $b = 1, \dots, n$ **do**
- 2 $\text{score}(b) \leftarrow |\{k : \tilde{\mathbf{X}}_{b, k} \geq \tau\}|$
- 3 $S_r \leftarrow$ set of indices b with the highest s scores
- 4 **return** S_r

the potential to improve the accuracy of read assignments because while some of the reads in the cluster might have sequencing errors, the others might be able to ‘compensate’. Thus, we can have more confidence in the vote of high-quality shared k -mers. There is, however, the possibility that overlaps are misleading. For instance, overlaps between repetitive reads might lead one to assign them to the same cluster while in reality these reads belong to different BACs. To reduce the impact of this issue we allow any read that belongs to multiple clusters to be decoded multiple times and take the intersection of the multiple assignments as the final assignment for the read. If a read does not overlap any other read (which could be explained due to the presence of several sequencing errors) we revert to the single read decoding strategy. In order to build the clusters we compute all pairwise read overlaps using SGA [13], whose parameters are discussed in Section 5.

In order to apply FINDSUPPORT on a cluster c of reads, we need to gather the frequency matrix \mathbf{Y}_c for c . Since the total number of k -mers within a cluster can be quite large as the clusters themselves can be quite large, and each k -mer can be shared by a subset of the reads in the cluster, we build \mathbf{Y}_c on the most frequently shared valid k -mers in the cluster. Our experiments indicate that retaining a number of k -mers equal to the numbers of k -mers used in the decoding of individual reads is sufficient. When reads within a cluster do not share a sufficient number of valid k -mers, we break the cluster into singletons and decode its reads individually. We denote by μ the minimum number of valid k -mers required to attempt decoding of both clusters and individual reads. The choice of this parameter is also discussed in Section 5.

We can also use mate pair information to improve the decoding, if reads are sequenced as paired-ends (PE). The *mate resolution strategy* (MRS) we employ is straightforward. Given a PE read r , (1) if the assignment of one of the mates of r is empty, we assign r to the BACs of the non-empty mate; (2) if both mates of r have BAC assignments and the intersection of these assignments is non-empty, we assign r to the BACs in the intersection; (3) if both mates of r have BAC assignments and their intersection is empty, we discard both mates. In what follows, we will use *RBD* to refer to the read based-decoding and *CBD* to refer to the cluster-based decoding versions of our algorithm. CBD with MRS is summarized in Algorithm 4.

5 Experimental Results

While our algorithms can be used to decode any set of DNA samples pooled according to STD, in this paper, we evaluate their performance on sets of BAC clones selected in such a way that they cover the genome (or a portion thereof) with minimum redundancy. In other words, the BACs we use form a *minimum tiling path* (MTP) of the genome. The construction of a MTP for a given genome requires a physical map, but both are well-known procedures and we will not discuss them here (see, *e.g.*, [4] and references therein). Once the set of MTP BAC clones has been identified, we (1) pool them according to STD, (2) sequence

Algorithm 4. CLUSTERFINDSUPPORT($\Phi, \mathcal{C}, \{\mathbf{Y}_c\}_{c \in \mathcal{C}}, h, s$)

Input : $\Phi \in \{0, 1\}^{m \times n}$, parameter h , sparsity parameter s , set \mathcal{C} of all clusters, frequency matrix \mathbf{Y}_c for each cluster $c \in \mathcal{C}$

Output: A support set S_r with $|S_r| \leq s$ for each read r

```

1 for each cluster  $c \in \mathcal{C}$  do
2    $S_c \leftarrow \text{FINDSUPPORT}(\Phi, \mathbf{Y}_c, h, s)$ 
3   for each read  $r \in c$  do
4     if  $S_r = \emptyset$  then  $S_r \leftarrow S_c$ 
5     else  $S_r \leftarrow S_r \cap S_c$  // Take intersection of all assignments to  $r$ 
6 // MRS
7 for each PE read  $(r_1, r_2)$  do
8   if  $S_{r_1} = \emptyset$  then  $S_{r_1} \leftarrow S_{r_2}$ 
9   if  $S_{r_2} = \emptyset$  then  $S_{r_2} \leftarrow S_{r_1}$ 
10  if  $S_{r_1} \neq \emptyset$  and  $S_{r_2} \neq \emptyset$  then
11     $S_{r_1, r_2} \leftarrow S_{r_1} \cap S_{r_2}$ 
12    if  $S_{r_1, r_2} \neq \emptyset$  then
13       $S_{r_1} \leftarrow S_{r_1, r_2}$ 
14       $S_{r_2} \leftarrow S_{r_1, r_2}$ 

```

the resulting pools, (3) apply our decoding algorithm to assign reads back to their source BACs. Step (3) makes it possible to assemble reads BAC-by-BAC, thus simplifying the genome assembly problem and increasing the accuracy of the resulting BAC assemblies [9].

Recall that CS decoding requires the unknown assignment vector \mathbf{x} to be s -sparse. Since we use MTP BAC clones, if the MTP was truly a set of minimally overlapping clones, setting s equal to 2 would be sufficient; we set it equal to 3 instead to account for imperfections in the construction of the MTP and to obtain additional protection against errors. Figure 1 illustrates the three cases (read belongs to one BAC, two BACs or three BACs) we will be dealing with during decoding, and how it affects our STD parameter choice.

Next, we present experimental evaluations where we pool BAC clones using the following STD parameters. Taking into consideration the need for a 3-decodable pooling design for MTP BACs, we choose parameters $q = 13$, $L = 7$ and $\Gamma = 2$, so that $m = qL = 91$, $n = q^{\Gamma+1} = 2197$ and $d = \lfloor (L - 1)/\Gamma \rfloor = 3$.

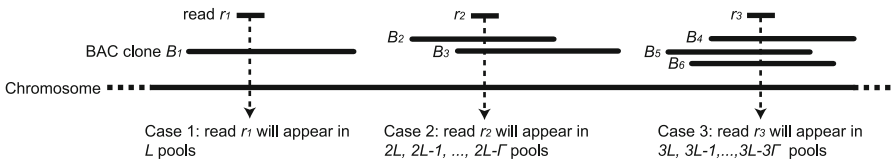


Fig. 1. The three cases we are dealing with during read decoding

In words, we pool 2197 BACs in 91 pools distributed in 7 layers of 13 pools each. Each BAC is pooled in exactly 7 pools and each pool contains $q^\Gamma = 169$ BACs. Recall that we call the set of L pools to which a BAC is assigned the BAC signature. In the case of STD, any two-BAC signatures can share at most $\Gamma = 2$ pools and any three-BAC signatures can share at most $3\Gamma = 6$ pools.

5.1 Simulation Results on the Rice Genome

To simulate our combinatorial pooling protocol and subsequent decoding, we used the genome of rice (*Oryza sativa*) which is about 390 Mb and fully sequenced. We started from an MTP of 3,827 BAC clones selected from a real physical map library for rice of 22,474 clones. The average BAC length in the MTP was ≈ 150 kB. Overall the clones in the MTP spanned 91% of the rice genome. We pooled a subset of 2,197 of these BACs into 91 pools according to the pooling parameters defined above. The resulting pools were ‘sequenced’ *in silico* using SIMSEQ, which is a high-quality short read simulator used to generate the synthetic data for Assemblathon [3]. SIMSEQ uses error profiles derived from real Illumina data to inject “realistic” substitution errors. For each pool, we generated 10^6 PE reads of 100 bases each with an average insert size of 300 bases. A total of 200M usable bases gave an expected $\approx 8\times$ sequencing depth for a BAC in a pool. As each BAC is present in 7 pools, this is an expected $\approx 56\times$ combined coverage before decoding. After decoding however, since a read can be assigned to more than one BAC, the actual average BAC sequencing depth became $91.68\times$ for RBD, $93\times$ for CBD and $97.91\times$ for CBD with MRS.

To simulate our current workflow, we first performed error-correction on the synthetic reads using SGA [13] with k -mer size parameter $k = 26$. Then, the hash table for $k = 26$ was built on the corrected reads, but we only stored k -mers appearing in at least $\gamma = 3$ pools. Due to the error-correction preprocessing step and the fact that we are discarding k -mers with low pool count, the hash table was relatively small (about 30GB).

In order to objectively evaluate and compare the performance of our decoding algorithms, we first had to precisely define the ‘ground truth’ for simulated reads. An easy choice would have been to consider ‘true’ only the single BAC from which each read was generated. However, this notion of ground truth is not satisfactory: for instance, since we can have two or three BACs overlapping each other in the MTP, reads originating from an overlap region are expected to be assigned to all the BACs involved. In order to find all the BACs that contain a read, we mapped all synthetic reads (error-free version) against the BAC primary sequences using BOWTIE [8] in stringent mode (paired-end end-to-end alignment with zero mismatches). The top three paired-end hits returned by BOWTIE constituted the ground truth against which we validated the accuracy of the decoding.

In our experiments we observed that although the majority of the reads are assigned to 1–3 BACs, due to the repetitive nature of the genome, a small fraction ($\approx 1\%$) can be correctly assigned to more than 3 BACs. To account for this, rather than sorting BAC scores and retaining the top 3, we decided to assign

Table 1. Accuracy of the decoding algorithms on synthetic reads for the rice genome (see text for details). All values are an average of 91 pools. Boldface values highlight the best result in each column (excluding perfect decoding).

	<i>Mapped to source BAC</i>	<i>Precision</i>	<i>Recall</i>	<i>F-score</i>	<i>Not decoded</i>
Perfect decoding	100.00%	98.11%	49.62%	65.90%	0.00%
HASHFILTER [9]	99.48%	97.45%	99.28%	98.36%	16.25%
RBD	98.05%	97.81%	97.46%	97.64%	14.58%
CBD	97.23%	97.74%	96.35%	97.04%	12.58%
CBD + MRS	96.60%	97.89%	95.58%	96.72%	7.09%

a read to all BACs whose score was above a certain threshold. We found that retaining all BACs whose score was at least $0.5K_r$ gave the best results. Recall that the score function we are using is the l_0 norm, so we are effectively asking that at least half of the k -mers ‘vote’ for a BAC.

Table 1 summarizes and compares the decoding performance of our algorithms. The first row of the table reports the performance of an ‘ideal’ method that always assigns each read to its original source BAC. The next four rows summarize (1) the performance of HASHFILTER [9] with default parameters; (2) our read-based decoding (RBD); (3) our cluster-based decoding (CBD); (4) our cluster-based decoding with mate resolution strategy (CBD + MRS). For all three versions of the decoding algorithm we used parameters $h = \lfloor q/2 \rfloor = 6$ and $\tau = 1$.

To build clusters, we require a minimum overlap of 75 bases between two reads and a maximum error rate of 0.01 (SGA parameters). The resulting clusters contained on average about 5 reads. Our methods make a decoding decision if a read (or cluster) contains at least $\mu = 15$ valid k -mers. The columns in Table 1 report the percentage of reads assigned to the original source BAC, *precision* (defined as $TP/(TP + FP)$ where TP is the number of true positive BACs across all decoded reads; FP and FN are computed similarly), *recall* (defined as $TP/(TP + FN)$), *F-score* (harmonic mean of precision and recall) and the percentage of reads that were not decoded. Observe that the highest precision is achieved by the cluster-based decoding with MRS, and the highest recall is obtained by HASHFILTER. In general, all methods are comparable from the point of view of decoding precision and recall. In terms of decoding time, once the hash table is built (≈ 10 h on one core), RBD takes on average 14.03s per 1M reads and CBD takes on average 33.46s per 1M clusters. By comparison, HASHFILTER [9] takes about 30s per 1M reads. These measurements were done on 10 cores of an Intel Xeon X5660 2.8 GHz server with 12 cores and 192 GB of RAM.

As a more meaningful measure of decoding performance, we assembled the set of reads assigned by each method to each BAC. We carried out this step using VELVET [18] for each of the 2,197 BACs, using a range of l -mer from 25 to 79 with an increment of 6, and chose the assembly that achieved the highest N50¹.

¹ The N50 is the contig length such that at least half of the total bases of a genome assembly are contained within contigs of this length or longer.

Table 2. Assembly results for rice BACs for different decoding algorithms (see text for details). All values are an average of 2197 BACs. Boldface values highlight the best result in each column (excluding perfect decoding).

	<i>Reads used</i>	<i># of contigs</i>	<i>N50</i>	<i>Sum/size</i>	<i>BAC coverage</i>
Perfect decoding (ideal)	97.1%	4	136,570	107.4	87.1%
HASHFILTER [9]	95.0%	24	52,938	93.8	76.2%
RBD	96.5%	20	46,477	90.0	81.1%
CBD	97.3%	22	53,097	93.8	84.7%
CBD + MRS	97.0%	11	103,049	97.0	82.9%

Table 2 reports the main statistics for the assemblies: percentage of reads used by VELVET in the assembly, number of contigs (at least 200 bases long) of the assembly, value of N50, ratio of the sum of all contigs sizes over BAC length, and the coverage of the BAC primary sequence by the assembly. All reported values are averages over 2,197 BACs. We observe that our decoding algorithms lead to superior assemblies than HASHFILTER’s. In particular, the N50 and the average coverage of the original BACs are both very high, and compare favorably with the statistics for the assembly of perfectly decoded reads.

The discrepancy between similar precision/recall figures but quite different assembly statistics deserves a comment. First, we acknowledge that the way we compute precision and recall by averaging TP , FP and FN across all decoded reads might not be the best way of measuring the accuracy of the decoding. Taking averages might not accurately reflect mis-assignments at the level of individual reads. Second, our decoding algorithms makes a better use of the k -mer frequency information than HASHFILTER, and, at the same time, takes advantage of overlap and mate pair information, which is expected to result in more reads decoded and more accurate assemblies.

5.2 Results on the Barley Genome

We have also collected experimental results on real sequencing data for the genome of barley (*Hordeum vulgare*), which is about 5,300 Mb and at least 95% repetitive. We started from an MTP of about 15,000 BAC clones selected from a subset of nearly 84,000 gene-enriched BACs for barley (see [9] for more details). We divided the set of MTP BACs into seven sets of $n = 2197$ BACs and pooled each set using the STD parameters defined above. In this manuscript, we report on one of these seven sets, called HV3 (the average BAC length in this set is about 116K bases). The 91 pools in HV3 were sequenced on one flow cell of the Illumina HiSeq2000 by multiplexing 13 pools on each lane. After each sample was demultiplexed, we quality-trimmed and cleaned the reads of spurious sequencing adapters and vectors. We ended up with high quality reads of about 87–89 bases on average. The number of reads in a pool ranged from 4.2M to 10M, for a grand total of 826M reads. We error-corrected and overlap-clustered the reads using SGA (same parameters as for rice). The average cluster size was

Table 3. Assembly results for barley BACs for different decoding algorithms. All values are an average of 2197 BACs. Boldface values highlight the best result in each column. Column “% coverage” refers to the coverage of known unigenes by assembled contigs.

	<i>Reads used</i>	<i># contigs</i>	<i>N50</i>	<i>Sum/size</i>	<i># obs unigenes</i>	<i>% coverage</i>
HASHFILTER [9]	83.6%	101	8,190	96.7%	1,433	92.9%
RBD	85.7%	54	14,419	101.0%	1,434	92.4%
CBD	92.9%	54	13,482	94.5%	1,436	92.6%
CBD + MRS	94.3%	50	26,842	126.8%	1,434	92.5%

about 26 reads. Computing pairwise overlaps took an average of 217.60s per 1M reads on 10 cores. The hash table for $k = 26$ (after discarding k -mers appearing in fewer than $\gamma = 3$ pools) used about 26GB of RAM. After decoding the reads to their BAC, we obtained an average sequencing depth for one BAC of $409.2\times$, $382.2\times$ and $412.8\times$ for RBD, CBD and CBD + MRS, respectively. The average running time was 10.25s per 1M reads for RBD and 82.12s per 1M clusters for CBD using 10 cores.

The only objective criterion to assess the decoding performance on barley genome is to assemble the reads BAC-by-BAC and analyze the assembly statistics. We used VELVET with the same l -mer choices as used for rice. Table 3 summarizes the statistics for the highest N50 among those l -mer choices. As before, rows corresponds to the various decoding methods. Columns show (1) percentage of reads used by VELVET in the assembly, (2) number of contigs (at least 200 bases long), (3) value of N50, (4) ratio of the sum of all contigs sizes over estimated BAC length, (5) the number of barley known unigenes observed in the assemblies, and (6) the coverage of observed unigenes. Observe that, out of a total of 1,471 known unigenes expected to be contained in these BACs, a large fraction are reported by all assemblies. However, cluster-based decoding appears to generate significantly longer contigs than the other methods.

6 Conclusions

We have presented a novel modeling and decoding approach for pooled sequenced reads obtained from protocols for *de novo* genome sequencing, like the one proposed in [9]. Our algorithm is based on the theory of compressed sensing and uses ideas from the decoding of error-correcting codes. It also effectively exploits overlap and mate pair information between the sequencing reads. Experimental results on synthetic data from the rice genome as well as real data from the genome of barley show that our method enables significantly higher quality assemblies than the previous approach, without incurring higher decoding times.

Acknowledgments. SL and TJC were supported by NSF [DBI-1062301 and DBI-0321756] and by USDA [2009-65300-05645 and 2006-55606-16722]. MW and ACG were supported by NSF [CCF-1161233]. HQN and AR were supported by NSF [CCF-1161196].

References

1. Alon, S., Vigneault, F., Eminaga, S., et al.: Barcoding bias in high-throughput multiplex sequencing of mirna. *Genome Research* 21(9), 1506–1511 (2011)
2. Amir, A., Zuk, O.: Bacterial community reconstruction using compressed sensing. In: Bafna, V., Sahinalp, S.C. (eds.) RECOMB 2011. LNCS, vol. 6577, pp. 1–15. Springer, Heidelberg (2011)
3. Earl, D., et al.: Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research* 21(12), 2224–2241 (2011)
4. Engler, F.W., Hatfield, J., Nelson, W., Soderlund, C.A.: Locating sequence on FPC maps and selecting a minimal tiling path. *Genome Research* 13(9), 2152–2163 (2003)
5. Erlich, Y., Chang, K., Gordon, A., et al.: DNA sudoku - harnessing high-throughput sequencing for multiplexed specimen analysis. *Genome Research* 19(7), 1243–1253 (2009)
6. Erlich, Y., Gordon, A., Brand, M., et al.: Compressed genotyping. *IEEE Transactions on Information Theory* 56(2), 706–723 (2010)
7. Hajirasouliha, I., Hormozdiari, F., Sahinalp, S.C., Birol, I.: Optimal pooling for genome re-sequencing with ultra-high-throughput short-read technologies. *Bioinformatics* 24(13), i32–i40 (2008)
8. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* 10(3), R25 (2009)
9. Lonardi, S., Duma, D., Alpert, M., et al.: Combinatorial pooling enables selective sequencing of the barley gene space. *PLoS Comput. Biol.* 9(4), e1003010 (2013)
10. Ngo, H.Q., Porat, E., Rudra, A.: Efficiently decodable compressed sensing by list-recoverable codes and recursion. In: STACS, pp. 230–241 (2012)
11. Prabhu, S., Pe’er, I.: Overlapping pools for high-throughput targeted resequencing. *Genome Research* 19(7), 1254–1261 (2009)
12. Shental, N., Amir, A., Zuk, O.: Identification of rare alleles and their carriers using compressed se(que)nsing. *Nucleic Acids Research* 38(19), e179–e179 (2010)
13. Simpson, J.T., Durbin, R.: Efficient de novo assembly of large genomes using compressed data structures. *Genome Research* 22(3), 549–556 (2012)
14. The International Barley Genome Sequencing Consortium. A physical, genetic and functional sequence assembly of the barley genome. *Nature* (advance online publication October 2012) (in press)
15. Thierry-Mieg, N.: A new pooling strategy for high-throughput screening: the shifted transversal design. *BMC Bioinformatics* 7(28) (2006)
16. Tropp, J.A., Gilbert, A.C.: Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inform. Theory* 53, 4655–4666 (2007)
17. Tropp, J.A., Gilbert, A.C., Strauss, M.J.: Algorithms for simultaneous sparse approximation: part i: Greedy pursuit. *Signal Process.* 86(3), 572–588 (2006)
18. Zerbino, D., Birney, E.: Velvet: Algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Research* 8(5), 821–829 (2008)

A Novel Combinatorial Method for Estimating Transcript Expression with RNA-Seq: Bounding the Number of Paths

Alexandru I. Tomescu¹, Anna Kuosmanen¹, Romeo Rizzi², and Veli Mäkinen¹

¹ Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Finland
{tomescu, aekuosma, vmakinen}@cs.helsinki.fi

² Department of Computer Science, University of Verona, Italy
romeo.rizzi@univr.it

Abstract. RNA-Seq technology offers new high-throughput ways for transcript identification and quantification based on short reads, and has recently attracted great interest. The problem is usually modeled by a weighted splicing graph whose nodes stand for exons and whose edges stand for split alignments to the exons. The task consists of finding a number of paths, together with their expression levels, which optimally explain the coverages of the graph under various fitness functions, such as least sum of squares. In (Tomescu *et al.* RECOMB-seq 2013) we showed that under general fitness functions, if we allow a polynomially bounded number of paths in an optimal solution, this problem can be solved in polynomial time by a reduction to a min-cost flow program. In this paper we further refine this problem by asking for a bounded number k of paths that optimally explain the splicing graph. This problem becomes NP-hard in the strong sense, but we give a fast combinatorial algorithm based on dynamic programming for it. In order to obtain a practical tool, we implement three optimizations and heuristics, which achieve better performance on real data, and similar or better performance on simulated data, than state-of-the-art tools Cufflinks, IsoLasso and SLIDE. Our tool, called Traph, is available at <http://www.cs.helsinki.fi/gsa/traph/>

1 Introduction

In this paper we tackle a biological multi-assembly problem [26] motivated by the recent RNA-Seq technology [17,20,19]: reconstruct as accurately as possible the RNA transcripts of a gene, given only a set of short RNA reads sequenced from them. The transcripts are concatenations of exons, the difficulty of the problem arising from the fact that they can have some identical exons.

This problem has attracted great interest from the community, resulting in tools such as Cufflinks [22], IsoInfer/IsoLasso [4,11], SLIDE [12], CLIQ [9], Scripture [5], iReckon [16], TRIP [13], NSMAP [25], Montebello [8], FlipFlop [2]. The methods rely on a graph model, the most common being a *splicing graph* [6]. Its nodes represent contiguous stretches of DNA uninterrupted by

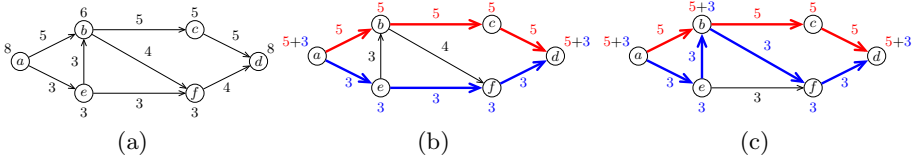


Fig. 1. An example for $k = 2$, and fitness function $f_v(x) = x^2$, $f_{uv}(x) = x^2$, for all nodes v , and edges (u, v) . In Fig. 1(a), a splicing directed acyclic graph; its nodes and edges are labeled with their observed average coverage. In Fig. 1(b), the optimal 2 paths for Problem 2-UTEO, with expression levels 5 and 3; their cost is $1 + 1 = 2$ (from node b , and edge (f, d) , respectively). In the case of Problem 2-UTEC, we have to add $3^2 + 4^2$ to their cost (from uncovered edges (e, b) , (b, f)), which is not optimal. In Fig. 1(c), the optimal 2 paths for Problem 2-UTEC, with expression levels 5 and 3; their cost is $2^2 + (1 + 1 + 3^2) = 15$ (from node b , and edges (b, f) , (f, d) , (e, f) , respectively).

spliced reads (called *pseudo-exons*), while its edges are derived from overlaps, or from spliced read alignments. The splicing graph is directed and acyclic (a DAG), the orientation of the edges being according to the starting position of the pseudo-exons inside the genome. Every node v has an associated observed average coverage, computed as the total number of reads aligned to the pseudo-exon v , divided by the exon length. Similarly, every edge (u, v) has an associated coverage, which is the total number of reads split aligned to the junction between pseudo-exons u and v .

The biological problem translates to covering the graph with intersecting paths, under different cost models, such as least sum of squares (IsoInfer/IsoLasso, SLIDE), least sum of absolute differences (CLIIQ). Many of the above mentioned tools work by exhaustively enumerating all possible (combinations of) paths, with some restrictions, and then estimating their fitness with an Integer Linear Program, Quadratic Program, or a QP + LASSO regression. Cufflinks computes a path cover with a minimum number of paths, and only in a second step estimates their expression levels.

In [21] we introduced a novel very general framework, encompassing many of the previous proposals; according to the survey [1], it can be classified as *de novo* genome-based, since it does not use annotation information. Apparently, parallel to our work, a similar min-cost flow approach, called FlipFlop, was proposed in [2]. Our method assumes that for every node v and edge (u, v) of the splicing graph, we are given fitness functions f_v and f_{uv} which penalize the difference between the observed average coverage and the predicted coverage. The problem was translated as finding (an unlimited number of) paths with associated expression levels such that the sum of all penalties is minimum. For example, if for every node or edge z , $f_z(x) = x^2$, then we have a least sum of squares model as in IsoInfer/IsoLasso and SLIDE, and if $f_z(x) = x$ we have a least sum of absolute difference model as in CLIIQ (see Fig. 1 for an example). Various other fitting functions can be considered, such as $f_z(x) = x/cov(z)$ [7], or $f_v(x) = x^2 * length(v)^2$. Therein, we proposed a min-cost flow method to

solve this problem in polynomial-time, assuming the fitting functions are convex, which was competitive with Cufflinks and IsoLasso. In that approach, the size of the solution is polynomially bounded, but it was left open to find even more parsimonious optimal or good solutions.

We now tackle the problem of optimally covering the splicing graph with a bounded number k of paths (see Fig. 1 for an example). This is relevant in practice since a small fraction of the graph can be erroneous due to various biological events or technical errors, like template switching, self-priming, reading errors, wrong splicing alignment [3,19,14,15].

Problem 1 (k -UTEC). Given a splicing DAG $G = (V, E)$ with positive coverage values $cov(v)$ and $cov(u, v)$, integer $k \geq 1$, and fitting functions $f_v(\cdot)$ and $f_{uv}(\cdot)$, for all $v \in V$ and $(u, v) \in E$, the k -Unannotated Transcript Expression Cover Problem is to find a tuple \mathcal{P} of k paths from the sources of G to the sinks of G , with an estimated expression level $e(P)$ for each path $P \in \mathcal{P}$, which minimize

$$\text{sum_err_c}(G, \mathcal{P}) := \sum_{v \in V} f_v \left(\left| cov(v) - \sum_{P \in \mathcal{P} \text{ s.t. } v \in P} e(P) \right| \right) + \sum_{(u,v) \in E} f_{uv} \left(\left| cov(u, v) - \sum_{P \in \mathcal{P} \text{ s.t. } (u,v) \in P} e(P) \right| \right).$$

We also study the following outlier sensitive variant asking for k paths which best fit to the coverage only of the nodes and edges that they contain.

Problem 2 (k -UTEO). Under the same assumptions as for Problem k -UTEC, the k -Unannotated Transcript Expression Outlier Problem is to find a tuple \mathcal{P} of $k \geq 1$ paths from the sources of G to the sinks of G , with an estimated expression level $e(P)$ for each path $P \in \mathcal{P}$, which minimize

$$\text{sum_err_o}(G, \mathcal{P}) := \sum_{P \in \mathcal{P}, v \in P} f_v \left(\left| cov(v) - \sum_{P \in \mathcal{P} \text{ s.t. } v \in P} e(P) \right| \right) + \sum_{P \in \mathcal{P}, (u,v) \in P} f_{uv} \left(\left| cov(u, v) - \sum_{P \in \mathcal{P} \text{ s.t. } (u,v) \in P} e(P) \right| \right).$$

In Sec. 2.1 we show that both problems are NP-hard in the strong sense.

¹ Nevertheless, in Sec. 2.2 we give dynamic programming algorithms with a time-complexity of $O(|M|^k(n^2 + \Delta^k)n^k)$, where M is the set of all possible expression levels, and Δ is the maximum in-degree of the graph. To obtain a practical implementation of these algorithms, we apply, as explained in Sec. 2.3, the following optimizations and heuristics:

¹ We should note that a preliminary version of this paper was presented as a poster at the RECOMB-seq, April 2013, conference <http://bioinfo.au.tsinghua.edu.cn/recomb2013/upload/programseq.pdf>, and that our NP-hardness proof has already inspired the NP-hardness proof [10] of the isoform reconstruction by maximum likelihood problem, deployed in tools such as iReckon, NSMAP, Montebello.

1. We decompose the problem along cut nodes, i.e., we find a node whose removal disconnects the graph into two components, and recursively solve the problem on the two subgraphs.
2. We employ a genetic algorithm for finding the optimal expression levels: the fitness of a given k -tuple of expression levels is the cost of the optimal paths having these expression levels, obtained by our dynamic programming in time $O((n^2 + \Delta^k)n^k)$; experiments show that the genetic algorithm has very small variability in practice.
3. In order to reduce the exponential dependency on k , we choose a $k' \leq k$ depending on the size of the graph and guaranteeing that the problem is tractable, then compute the optimal k' paths, remove their weight from the graph, and recurse until obtaining k paths in total.

Experimental results, given in Sec. 3, show that our algorithm, together with the above optimizations and heuristics, has better performance on real RNA-Seq data, and similar or better performance on simulated data, than our min-cost flow method, and than state-of-the-art tools Cufflinks, IsoLasso, and SLIDE. In these experiments, we run the program for all values of k up to a bound depending on the size of the input graph, and choose the k such that the paths returned by the program have the minimum value of the objective function. However, the choice of k is highly customizable by the user.

2 Methods

2.1 The NP-Hardness Proof

Theorem 1. *If the cost functions f_v and f_{uv} are such that $f_v(0) = 0$, $f_{uv}(0) = 0$, and $f_v(x) > 0$, $f_{uv}(x) > 0$ for all $x > 0$ and all nodes v and edges (u, v) of the input splicing graph, then Problems k -UTEC and k -UTEO are NP-hard in the strong sense.*

Proof. We follow the proof of [24], underlining the differences in what follows. We reduce from 3-PARTITION. In this problem, we are given a set $A = \{a_1, \dots, a_{3q}\}$ with $3q$ elements, and for all $a \in A$, a positive integer $s(a)$, its size, such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = qB$. We are asked whether there exists a partition of A into q disjoint sets each of size B .

Given an instance (A, s) to 3-PARTITION, we construct (see also Fig. 2(a)) the graph $G_{A,s}$ having:

- $V(G_{A,s}) = \{s, x_1, \dots, x_{3q}, y, z_1, \dots, z_q, t\}$,
- for every $i \in \{1, \dots, 3q\}$, we add arcs (s, x_i) , (x_i, y) to $G_{A,s}$, both with coverage $s(a_i)$, and also set the coverage of x_i to $s(a_i)$,
- for every $i \in \{1, \dots, q\}$, we add arcs (y, z_i) and (z_i, t) to $G_{A,s}$, both with coverage B , and also set the coverage of z_i to B ,
- the coverage of s , y and t is qB .

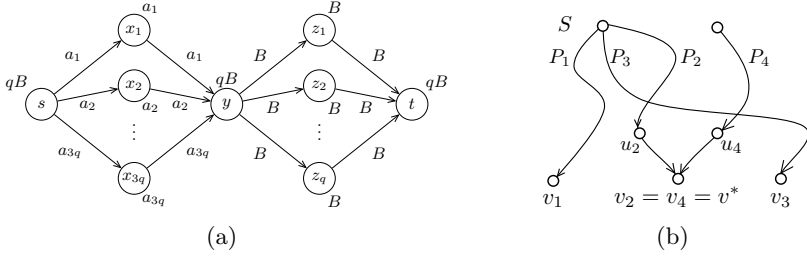


Fig. 2. In Fig. 2(a), a reduction of 3-PARTITION to Problems k -UTEC or k -UTEO. In Fig. 2(b), computing $\text{solution}(v_1, v_2, v_3, v_4)$. We assume that $v_2 = v_4$ is a sink of G_{v_1, \dots, v_k} , and it is chosen as v^* ; we then enumerate through all pairs of vertices from $N^-(v^*) \times N^-(v^*)$; in this case, we find (u_2, u_4) and we extend the optimal paths ending in (v_1, u_2, v_3, u_4) with the edges (u_2, v^*) and (u_4, v^*)

We prove that there exists a partition of A into q sets of size B if and only if Problem k -UTEC admits on $G_{A,s}$ a solution with cost 0 made up of at most $3q$ paths, and analogously for Problem k -UTEO.

For the forward implication, let A_1, \dots, A_q be a partition of A into q sets of size B . To obtain a solution to Problem k -UTEC with cost 0, for every $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}\}$ we add to the solution the three paths (s, x_{i_1}, y, z_i, t) , (s, x_{i_2}, y, z_i, t) , (s, x_{i_3}, y, z_i, t) . These three paths completely cover the edges $(s, x_{i_1}), (x_{i_1}, y)$, $(s, x_{i_2}), (x_{i_2}, y)$, and $(s, x_{i_3}), (x_{i_3}, y)$, respectively, and they are the only paths to do so, since A_1, \dots, A_q is a partition of A . This results in a zero cost to be added to the objective function. Moreover, since $s(a_{i_1}) + s(a_{i_2}) + s(a_{i_3}) = B$, then these three paths together completely cover the edges (y, z_i) and (z_i, t) . This also implies a zero cost to be added to the objective function.

For the backward implication, observe that a solution to Problem k -UTEC with cost 0 and at most $3q$ paths must have exactly $3q$ paths. To see why this is the case, observe that the sum of the expression levels of the paths is exactly qB , as they pass through node y , and this is a zero-cost solution. Moreover, the sum of the coverages of vertices x_1, \dots, x_{3q} is qB , by construction. The fact that this is a zero-cost solution thus implies that each of the $3q$ vertices x_1, \dots, x_{3q} must be covered by at least one of the $3q$ paths. Therefore, each x_i is covered by exactly one path.

For every $i \in \{1, \dots, q\}$, let Q_i denote the set of paths in this optimal solution covering node z_i . As this is a zero-cost solution, the sum of their expression levels is B , and their expression levels belong to A . Since $B/4 < s(a) < B/2$, for all $a \in A$, then each Q_i contains exactly three paths. This entails that for any $1 \leq i < j \leq q$, $Q_i \cap Q_j = \emptyset$. Therefore, by associating with each $i \in \{1, \dots, q\}$ the subsets of A that correspond to the first arc of the three paths of Q_i we obtain a partition of A into q sets of size B .

The proof of [24, Proposition 2] can be followed identically from this point onwards to show that this is a pseudo-polynomial reduction. The proof for Problem k -UTEO is entirely similar. \square

2.2 The Dynamic Programming Algorithms

Onwards, we propose dynamic programming algorithms for Problems k -UTEO and k -UTEC. Since the algorithm for Problem k -UTEO is simpler than for Problem k -UTEC, we present the former here, and defer the latter to the full version of this paper.

We will assume that the possible expression levels of the paths in an optimal solution belong to a finite set M . Our strategy is to find the optimal k -tuple of paths having a fixed k -tuple of expression levels. The solution is then obtained by enumerating all k -tuples of expression levels from M^k , and taking the k -tuple of paths having the smallest value of the objective function. Despite the dependency on the expression levels' values, having the two steps separated means that we can employ, for a practical implementation, any local search heuristic for finding the optimal expression levels. This search will be guided by the cost of the objective function returned by the dynamic programming; the search can be done at any chosen granularity, eventually including a priori information about the true expression levels. We employ a genetic algorithm which behaves well in practice (see Sec. 2.3).

Let us assume from now on one such choice (e_1, \dots, e_k) of expression levels fixed. The main difficulty behind the algorithm is that the paths can share vertices. Accordingly, we have to process all k -tuples of vertices of V ; for every $(v_1, \dots, v_k) \in V^k$, we define

$$\text{solution}(v_1, \dots, v_k) := \min_{\substack{\text{paths } P_1, \dots, P_k \text{ in } G, \\ \text{each } P_i \text{ is from a source to } v_i}} \text{sum_err_o}(G, P_1, \dots, P_k).$$

Since the input directed graph is acyclic, we let \prec be a topological order on V , and define the partial order \prec^k on V^k as follows:

$$(v'_1, \dots, v'_k) \prec^k (v_1, \dots, v_k) \quad \text{iff} \quad \exists i \in \{1, \dots, k\} \text{ such that } v'_i \prec v_i.$$

Then, the computation of `solution` is done by dynamic programming, by enumerating the tuples $(v_1, \dots, v_k) \in V^k$ in the order \prec^k , and computing `solution` (v_1, \dots, v_k) from the previous values, according to \prec^k , as indicated in Algorithm 1, where $N^-(v)$ denotes the in-neighborhood of a node v , and G_{v_1, \dots, v_k} denotes the subgraph of G induced by the vertices from which there is a directed path to one of v_1, \dots, v_k (see Fig. 2(b) for a sketch).

Theorem 2. *If the cost functions f_v and f_{uv} are such that $f_v(x) \geq 0$, $f_{uv}(x) \geq 0$ for all $x \geq 0$ and all nodes v and edges (u, v) of the input splicing graph, then Problems k -UTEO and k -UTEC can be solved in time $O(|M|^k(n^2 + \Delta^k)n^k)$, where $n := |V(G)|$, we assume that M is the set of possible expression levels, and the maximum in-degree of G is Δ .*

Proof. We give the proof only for Problem k -UTEO. The algorithm and the proof for Problem k -UTEC are analogous, but more involved; they will be presented in the full version of this paper.

Algorithm 1. Computing $\text{solution}(v_1, \dots, v_k)$ for a fixed tuple (e_1, \dots, e_k) of expression levels, for Problem k -UTEO.

```

/* initialization for all possible source tuples */
foreach  $(s_1, \dots, s_k) \in S^k$  do
  solution $(s_1, \dots, s_k) \leftarrow \sum_{\substack{i \in \{1, \dots, k\} \text{ such that} \\ \forall i' < i, s_i \neq s_{i'}}} f_{s_i} \left( \left| \text{cov}(s_i) - \sum_{\substack{j \in \{i, \dots, k\} \\ \text{s.t. } s_j = s_i}} e_j \right| \right);$ 
solution $(v_1, \dots, v_k)$ 
  min  $\leftarrow \infty$ ;
  let  $v^*$  be a sink of  $G_{v_1, \dots, v_k}$ , which is not the source of  $G$ ;
  let  $i_1, \dots, i_\ell$  be all the positions in the tuple  $(v_1, \dots, v_k)$  where  $v^*$  appears;
  /* we enumerate through all  $\ell$ -tuples of in-neighbors of  $v^*$  */
  foreach  $(u_{i_1}, \dots, u_{i_\ell}) \in N^-(v^*)^\ell$  do
    /* we get the optimal cost for such a tuple */
    err  $\leftarrow \text{solution}(v_1, \dots, v_{i_1-1}, u_{i_1}, v_{i_1+1}, v_{i_\ell-1}, u_{i_\ell}, v_{i_\ell+1}, \dots, v_k)$ ;
    /* we sum up the cost of covering  $v^*$  with the  $\ell$  paths extended from
        $u_{i_1}, \dots, u_{i_\ell}$  having expression levels  $e_{i_1}, \dots, e_{i_\ell}$  */
    err  $\leftarrow \text{err} + f_{v^*} \left( \left| c(v^*) - \sum_{j=1}^{\ell} e_{i_j} \right| \right)$ ;
    /* we sum up the cost of covering the edges  $(u_{i_1}, v^*), \dots, (u_{i_\ell}, v^*)$  with the  $\ell$ 
       paths extended from  $u_{i_1}, \dots, u_{i_\ell}$  having expression levels  $e_{i_1}, \dots, e_{i_\ell}$ ,
       respectively */
    err  $\leftarrow \text{err} + \sum_{\substack{j \in \{1, \dots, \ell\} \text{ such that} \\ \forall j' < j, u_{i_j} \neq u_{i_{j'}}}} f_{u_{i_j} v^*} \left( \left| c(u_{i_j}, v^*) - \sum_{\substack{t \in \{j, \dots, \ell\} \\ \text{s.t. } u_{i_t} = u_{i_j}}} e_{i_t} \right| \right)$ ;
    if err < min then min  $\leftarrow$  err;
  return min.

```

Let (P_1, \dots, P_k) be a tuple of k optimal paths starting in a source and ending in v_1, \dots, v_k , i.e.,

$$\text{sum_err_o}(G, P_1, \dots, P_k) = \min_{\substack{\text{paths } Q_1, \dots, Q_k \text{ in } G, \\ \text{each } Q_i \text{ is from a source to } v_i}} \text{sum_err_o}(G, Q_1, \dots, Q_k). \quad (1)$$

Let v^* be a sink of G_{v_1, \dots, v_k} which is not a source of G (if none such node exists, then all v_1, \dots, v_k , are sources and the value of $\text{solution}(v_1, \dots, v_k)$ has already been set). Assume also that i_1, \dots, i_ℓ , $\ell \geq 1$, are the positions in the tuple (v_1, \dots, v_k) where v^* appears (see Fig. 2(b) for a sketch).

Let $u_{i_1}, \dots, u_{i_\ell}$ be the predecessors of v^* on $P_{i_1}, \dots, P_{i_\ell}$, respectively. For every $j \in \{1, \dots, \ell\}$, denote by $P_{i_j}^*$ the path P_{i_j} from which we remove its last node, v^* . To simplify notation, denote by $(P_1, \dots, P^*, \dots, P_k)$ the tuple (P_1, \dots, P_k) in which, for every $j \in \{1, \dots, \ell\}$, we replace P_{i_j} by $P_{i_j}^*$. Similarly, we denote by $(v_1, \dots, u, \dots, v_k)$ the tuple (v_1, \dots, v_k) in which, for every $j \in \{1, \dots, \ell\}$, we replace v_{i_j} by u_{i_j} .

From the fact that v^* is a sink of G_{v_1, \dots, v_k} , neither the node v^* , nor the edges in $new_edges := \{(u_{i_1}, v^*), \dots, (u_{i_\ell}, v^*)\}$, belong to any path in G ending in $\{v_1, \dots, v_k\} \setminus \{v^*\}$. Therefore, the following relation holds:

$$\begin{aligned} \text{sum_err_o}(G, P_1, \dots, P_k) &= \text{sum_err_o}(G, P_1, \dots, P^*, \dots, P_k) + \\ &+ f_{v^*} \left(\left| c(v^*) - \sum_{j=1}^{\ell} e_{i_j} \right| \right) + \sum_{\substack{j \in \{1, \dots, \ell\} \text{ such that} \\ \forall j' < j, u_{i_{j'}} \neq u_{i_j}}} f_{u_{i_j} v^*} \left(\left| c(u_{i_j}, v^*) - \sum_{\substack{t \in \{j, \dots, \ell\} \\ \text{s.t. } u_{i_t} = u_{i_j}}} e_{i_t} \right| \right). \end{aligned} \quad (2)$$

Let (P'_1, \dots, P'_k) be any tuple of k paths from a source to $(v_1, \dots, u, \dots, v_k)$ such that $\text{sum_err_o}(G, (P'_1, \dots, P'_k)) = \text{solution}(v_1, \dots, u, \dots, v_k)$. From the fact that also the paths $P_1, \dots, P^*, \dots, P_k$ end in $v_1, \dots, u, \dots, v_k$, respectively, and the optimality of (P'_1, \dots, P'_k) , we have

$$\text{sum_err_o}(G, P_1, \dots, P^*, \dots, P_k) \geq \text{sum_err_o}(G, P'_1, \dots, P'_k). \quad (3)$$

From (3) and (2) we get

$$\begin{aligned} \text{sum_err_o}(G, P_1, \dots, P_k) &\geq \text{sum_err_o}(G, P'_1, \dots, P'_k) + \\ &+ f_{v^*} \left(\left| c(v^*) - \sum_{j=1}^{\ell} e_{i_j} \right| \right) + \sum_{\substack{j \in \{1, \dots, \ell\} \text{ such that} \\ \forall j' < j, u_{i_{j'}} \neq u_{i_j}}} f_{u_{i_j} v^*} \left(\left| c(u_{i_j}, v^*) - \sum_{\substack{t \in \{j, \dots, \ell\} \\ \text{s.t. } u_{i_t} = u_{i_j}}} e_{i_t} \right| \right). \end{aligned} \quad (4)$$

Let us denote by $(P'_1, \dots, P' \cup \{v\}, \dots, P'_k)$ the tuple (P'_1, \dots, P'_k) in which, for every $j \in \{1, \dots, \ell\}$, we add node v^* at the end of path P'_{i_j} . Resorting again to the fact that the node v^* and the edges in new_edges do not belong to any path in G ending in $\{v_1, \dots, v_k\} \setminus \{v^*\}$, we get that the right-hand side of the inequality (4) is equal to $\text{sum_err_o}(G, P'_1, \dots, P' \cup \{v^*\}, \dots, P'_k)$, thus:

$$\text{sum_err_o}(G, P_1, \dots, P_k) \geq \text{sum_err_o}(G, P'_1, \dots, P' \cup \{v^*\}, \dots, P'_k). \quad (5)$$

To conclude, if we enumerate through all $(v'_{i_1}, \dots, v'_{i_\ell}) \in N^-(v^*)^\ell$, we will find the nodes $u_{i_1}, \dots, u_{i_\ell}$ preceding v^* on the optimal paths $P_{i_1}, \dots, P_{i_\ell}$, respectively. If we extend each optimal path ending in $v_1, \dots, u, \dots, v_k$ by the node v^* , we obtain paths $P'_1, \dots, P' \cup \{v^*\}, \dots, P'_k$ whose cost is optimal, by (1) and (5).

Once table solution is computed, the solution for Problem k -UTEQ is $\min_{(t_1, \dots, t_k) \in T^k} \text{solution}(t_1, \dots, t_k)$, where T is the set of sinks of G . Finally, note that if we store in $\text{solution}(v_1, \dots, v_k)$ also the predecessors u_1, \dots, u_k of v_1, \dots, v_k on the optimal paths from the sources, we can then trace back the k optimal paths from sources to sinks.

The time complexity bound follows from the fact that there are n^k tuples $(v_1, \dots, v_k) \in V^k$, computing the sinks of G_{v_1, \dots, v_k} takes time linear in the number of edges of G_{v_1, \dots, v_k} , which are $O(n^2)$, and there are at most Δ^k candidate predecessors $u_{i_1}, \dots, u_{i_\ell}$ of v^* on $P_{i_1}, \dots, P_{i_\ell}$, respectively, in the neighborhood of v^* . \square

2.3 Optimizations and Heuristics for a Practical Implementation

We implemented the two algorithms in our tool Traph [21], which can be seen as an alternative version of our min-cost flow method.

In order to speed-up the dynamic programming, we look for nodes whose removal disconnects the input graph G . If v is such a cut node, then we consider the subgraphs G_1 , induced by v and the nodes from which there is a directed path to v , and G_2 , induced by v and the nodes to which there is a directed path starting from v . It is clear that v is the only sink in G_1 and the only source in G_2 . We can then recursively solve Problem k -UTEC/ k -UTEO for G_1 , and use the optimal solution for v in G_1 as initialization to the dynamic programming table of G_2 , and solve the problem on G_2 .

Moreover, in order to avoid enumerating through all tuples of possible expression levels, at present we employ a genetic algorithm: for each individual (e_1, \dots, e_k) in the current population, its fitting function $\phi(e_1, \dots, e_k)$ is the cost of the optimal paths having the expression levels (e_1, \dots, e_k) , computed with our dynamic programming algorithms.

As by its nature genetic algorithm results vary, Traph iterates a hundred times each run, and the algorithm is rerun five times with different initial values. The solution with the smallest value of the objective function is then chosen. We tested the variability of the results by running Traph a hundred times on a sample containing alignments from one gene. All hundred runs resulted to the same transcripts, with the standard deviation of the path weight being less than 0.001% of the mean path weight for every transcript.

In order to reduce the exponential dependency on k , we choose a number $k' \leq k$ which depends on n and Δ such that $O((n^2 + \Delta^k)n^k)$ is small for practical purposes. We then solve the problem by looking for the best k' paths with their optimal expression levels, remove their weights from the graph, and then recurse until obtaining k paths as requested.

We use a least sum of squares model, i.e., the fitness function that we use is $f_z(x) = x^2$, for all nodes and edges z . In the present work, we also tried the fitting function $f_v(x) = x^2 * length(v)^2$ (which tries to explain the total coverage of the exons, not only their average coverage), and in [21] we tried the fitting function $f_v(x) = x/cov(v)$, but they did not give better results.

3 Experimental Results

We compared Traph cover (Problem UTEC), and Traph outlier (Problem UTEO) with Cufflinks [22], IsoLasso [11], SLIDE [9], and to our min-cost flow method [21]. We tried to compare also against Scripture, iReckon, CLIQ, but we ran into compatibility issues in installing them, or we could not get reliable results. Even though Traph is not yet employing paired-end read information, the experiments (both simulated and real) were conducted with paired-end reads, and Cufflinks, IsoLasso and SLIDE had access to the paired-end information. As Traph is a *de novo* genome-based tool, we ran the other tools without annotation. Our

experiment setup and validation criteria are the same as in [21].² We run the algorithms for all values of k up to a bound depending on the size of the input graph, and choose the minimum k giving the minimum of the objective function. However, the choice of k can be easily customized by the user.

Matching criteria. In order to match the predicted transcripts with the true transcripts, we take into account sequences but also expression levels. For each gene, we construct a bipartite graph with the true transcripts $\mathcal{T} = (T_1, T_2, \dots)$ as nodes in one set of the bipartition, and the predicted transcripts $\mathcal{P} = (P_1, P_2, \dots)$ as nodes in the other set of the bipartition. Empty sequences with 0 expression level were added so that both sets of the bipartition had an equal number of nodes. The cost of an edge between a true transcript T_i with expression level $e(T_i)$ and a predicted transcript P_j with expression level $e(P_j)$ is defined as a combined measure between: (i) the edit distance between T_i and P_j , divided by $\max(|T_i|, |P_j|)$, which we call *sequence dissimilarity*; and (ii) the ratio $|e(P_j) - e(T_i)|/e(T_i)$, which we call *relative expression level difference* (see [21] for further details). The minimum weight perfect matching was then computed; this gives a one-to-one mapping between true and predicted transcripts. Each matched node pair with relative expression level difference and sequence dissimilarity under given thresholds define a true positive event (TP). The other kind of nodes define false positive (FP) and false negative (FN) events, depending on side of the bipartite graph they reside. The prediction efficiency is based on precision = $\text{TP}/(\text{TP}+\text{FP})$, recall = $\text{TP}/(\text{TP}+\text{FN})$ and F-measure = $2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall})$.

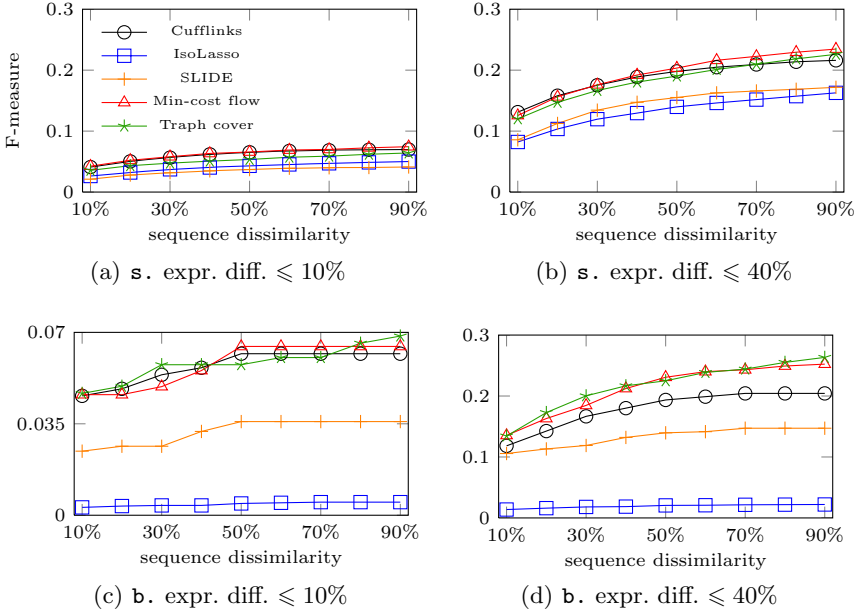
3.1 Simulated Human Data

For creating the simulated data we used the annotated genes in human chromosome 2 as reported by Ensembl database. Excluding the genes that had no transcripts as long or longer than the fragment size, we were left with 1,462 genes. We simulated reads with the RNASeqReadSimulator³ by first choosing an expression level for each transcript at random from lognormal distribution with mean -4 and variance 1, and then creating paired-end reads with fragment length mean 300 and standard deviation 20, with the starting positions of the fragments being chosen uniformly inside the transcripts. As argued in the case of IsoLasso [11], various error models can be incorporated in these steps, but we chose to compare the performance of the methods in neutral conditions.

We devised two experiment setups. In the first one, which we call **singles**, 300,000 paired-end reads were generated independently from the transcripts of

² In [21] we use *bitscore* instead of sequence dissimilarity, which is based on normalized compression distance and is better grounded as a measure. However, it needs full alignment to be output. Here we approximate this measure with sequence dissimilarity, which is fast to compute using Myers's bitparallel algorithm [18], and this enables much larger data sets to be evaluated within the time constraints of article submission.

³ <http://www.cs.ucr.edu/~liw/rnaseqreadsimulator.html>



Tool	Total predicted	Shared with annotation at sequence dissimilarity under				
		10%	20%	30%	40%	50%
Cufflinks	1916	648	955	1171	1307	1413
IsoLasso	1468	589	782	923	1022	1100
SLIDE	2229	635	983	1242	1391	1474
Min-cost flow	2148	722	1000	1228	1341	1456
Traph cover	2109	788	1063	1283	1407	1501

(e) The total number of transcripts reported by the tools on real data

Fig. 3. Performance of the tools on simulated and real data. Plots 3(a) and 3(b) depict results in the **singles** scenario, and plots 3(c) and 3(d) depict results in the **batch** scenario; the legend for all plots is as in Fig. 3(a). Real data results are in Fig. 3(e).

each of the genes, with the already assigned expression levels. They were independently given to TopHat [23] for alignment, and these independent alignment results were fed to each tool. In the second, more realistic experiment, which we call **batch**, we randomly chose 100 of the genes, chose expression levels for them with the same distribution as before and simulated $100 * 300,000$ reads as above. All these reads were fed to TopHat for alignment, and these combined alignment results were fed to the tools. The fragment length mean and standard deviation were passed to the tools, except for Cufflinks in **batch**, when it was able to infer them automatically; as our simulated data did not contain any single exon genes, SLIDE was unable to infer the fragment distribution, and it was given the fragment length mean and the standard deviation.

Cufflinks, IsoLasso and SLIDE were ran with the default parameters, because the parameters they offer relate to RNA-seq lab protocol, which was not simulated; we could not see changes to other parameters which could be relevant to the prediction. SLIDE’s results are highly dependent on the lambda values, and as such it encourages the user to manually adjust the lambda values if the result set seems to either be missing isoforms or contain too many short isoforms, but for the sake of having automated tests we used the lambda values SLIDE estimated from the data as is. We use FPKM values as expression levels. Full simulated experiment input data is available on the webpage of Traph.

Fig. 3 shows selected validation results. For each experiment, we choose two thresholds for the relative expression level differences, namely 10% and 40%. Overall, Traph cover outperformed Traph outlier; at the moment we consider Traph cover as the default implementation, and plan to apply Traph outlier to other multi-assembly problems. In the `singles` scenario, Cufflinks, our min-cost flow method and Traph cover have very similar F-measure and out-perform IsoLasso and SLIDE. In the `batch` scenario, we obtain the same situation when the relative expression level difference is allowed to be at most 10%, but the min-cost flow method and Traph cover out-perform the other tools when the threshold for relative expression level difference is 40%, Traph cover giving slightly better results. Note that in the `batch` scenario the tools predicted transcripts which fall outside the annotated gene areas, which we accounted as FP events in the plots. For the 100 genes, Cufflinks predicted 512 transcripts inside gene areas, and 215 outside gene areas, IsoLasso had 384 predictions inside and a surprising 7,422 outside; SLIDE had 725 inside and 94 outside; Traph cover had 458 inside and 98 outside; the min-cost flow method had 413 inside and 74 outside.

Running times. On the batch dataset of reads from 100 genes, Cufflinks ran in 421 min, IsoLasso in 38 min, SLIDE in 257 min. Our script for creating the splicing graphs is written in Python and took 180 min; the min-cost flow method ran on these splicing graphs in 117 min, and Traph cover in 538 min.

3.2 Real Human Data

We used the same real dataset from the IsoLasso paper, Caltech RNA-Seq track from the ENCODE project [GenBank:SRR065504], consisting of 75bp paired-end reads. Out of these reads, we picked the 2,406,339 which mapped to human chromosome 2. We selected the 735 genes where all tools made some prediction; these genes have 6,325 annotated transcripts.

The transcripts predicted by each tool are matched with the annotated transcripts, employing the same minimum weight perfect matching method introduced before, but without taking into account expression levels. A true positive is a match selected by the perfect matching with varying sequence dissimilarity threshold. We present these results in Table 3(e), where we note that Traph cover reports the most transcripts which match the annotation at all thresholds of sequence dissimilarity.

4 Conclusion

In this paper we tackled two multi-assembly problems arising from transcript identification and quantification with RNA-Seq, which ask for the k paths which best explain, under given fitting functions, the coverages of a splicing graph. In our experiments we worked with least sum of squares as fitting function, but our method supports very general fitting functions. We expect that these two models and algorithms to be applicable to other multi-assembly problems, such as in metagenomics or in viral quasi-species assembly.

The two problems considered, Problem k -UTEO and k -UTEC, are shown to be NP-hard in the strong sense, proof which already inspired a similar NP-hardness proof [10] of another problem pertaining to multi-assembly. If some of the input parameters are bounded (k , the maximum in-degree of the graph, the set of possible expression levels), then the problems can be solved in polynomial-time using dynamic programming. Nonetheless, in order to obtain a practical implementation, we considered three optimizations and heuristics which work well in practice, and in a feasible amount of time: on real data we report more annotated transcripts than Cufflinks, IsoLasso, SLIDE and our previous min-cost flow method, while on simulated data we obtain similar or better performance.

Acknowledgements. We wish to thank Antti Honkela for many insightful discussions on transcript prediction. We also thank Travis Gagie for discussions on the NP-hardness proof. This work was partially supported by Academy of Finland under grant 250345 (CoECGR).

References

1. Alamancos, G.P., Agirre, E., Eyras, E.: Methods to study splicing from high-throughput RNA Sequencing data. CoRR abs/1304.5952 (2013)
2. Bernard, E., et al.: Efficient RNA Isoform Identification and Quantification from RNA-Seq Data with Network Flows. SU2C-AACR-DT0409; SES-0835531; CCF-0939370
3. Brett, D., et al.: Alternative splicing and genome complexity. *Nature Genetics* 30(1), 29–30 (2001)
4. Feng, J., Li, W., Jiang, T.: Inference of isoforms from short sequence reads. In: Berger, B. (ed.) RECOMB 2010. LNCS, vol. 6044, pp. 138–157. Springer, Heidelberg (2010)
5. Guttman, M., et al.: Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. *Nat. Biotechnol.* 28(5), 503–510 (2010)
6. Heber, S., et al.: Splicing graphs and EST assembly problem. *Bioinformatics* 18(suppl. 1), S181–S188 (2002)
7. Heijden, V.D., et al.: Estimating the size of a criminal population from police records using the truncated poisson regression model. *Statistica Neerlandica* 57(3), 289–304 (2003)
8. Hiller, D., et al.: Simultaneous Isoform Discovery and Quantification from RNA-Seq., pp. 1–19 (2012)

9. Li, J.J., et al.: Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proc. of the National Academy of Sciences* 108(50), 19867–19872 (2011)
10. Li, T., Jiang, R., Zhang, X.: Isoform reconstruction using short RNA-Seq reads by maximum likelihood is NP-hard. *CoRR abs/1305.0916* (2013)
11. Li, W., et al.: IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly. *J. Comput. Biol.* 18(11), 1693–1707 (2011)
12. Lin, Y.-Y., Dao, P., Hach, F., Bakhshi, M., Mo, F., Lapuk, A., Collins, C., Sahinalp, S.C.: CLIIQ: Accurate Comparative Detection and Quantification of Expressed Isoforms in a Population. In: Raphael, B., Tang, J. (eds.) *WABI 2012*. LNCS, vol. 7534, pp. 178–189. Springer, Heidelberg (2012)
13. Mangul, S., et al.: An integer programming approach to novel transcript reconstruction from paired-end RNA-Seq reads. In: Ranka, S., et al. (eds.) *BCB*, pp. 369–376. ACM (2012)
14. Maniatis, T., Tasic, B.: Alternative pre-mRNA splicing and proteome expansion in metazoans. *Nature* 418(6894), 236–243 (2002)
15. McIntyre, L., et al.: RNA-seq: technical variability and sampling. *BMC Genomics* 12(1), 293 (2011)
16. Mezlini, A.M., et al.: iReckon: Simultaneous isoform discovery and abundance estimation from RNA-seq data. *Genome Research* 23(3), 519–529 (2012)
17. Mortazavi, A., et al.: Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nature Methods* 5, 621–628 (2008)
18. Myers, G.: A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM* 46(3), 395–415 (1999)
19. Ozsolak, F., Milos, P.M.: RNA sequencing: advances, challenges and opportunities. *Nature Reviews. Genetics* 12(2), 87–98 (2011)
20. Pepke, S., Wold, B., Mortazavi, A.: Computation for ChIP-seq and RNA-seq studies. *Nature Methods* 6(11), s22–s32 (2009)
21. Tomescu, A.I., Kuosmanen, A., Rizzi, R., Mäkinen, V.: A Novel Min-Cost Flow Method for Estimating Transcript Expression with RNA-Seq. *BMC Bioinformatics* 14(suppl. 5), S15 (2013), Presented at RECOMB-Seq, Beijing, China (2013)
22. Trapnell, C., et al.: Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology* 28, 511–515 (2010)
23. Trapnell, C., Pachter, L., Salzberg, S.L.: TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics* 25(9), 1105–1111 (2009)
24. Vatinlen, B., et al.: Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research* 185(3), 1390–1401 (2008)
25. Xia, Z., et al.: NSMAP: A method for spliced isoforms identification and quantification from RNA-Seq. *BMC Bioinformatics* 12(1), 162 (2011)
26. Xing, Y., et al.: The multiassembly problem: reconstructing multiple transcript isoforms from EST fragment mixtures. *Genome Res.* 14(3), 426–441 (2004)

A Polynomial Delay Algorithm for the Enumeration of Bubbles with Length Constraints in Directed Graphs and Its Application to the Detection of Alternative Splicing in RNA-seq Data

Gustavo Sacomoto^{1,2}, Vincent Lacroix^{1,2}, and Marie-France Sagot^{1,2}

¹ INRIA Rhône-Alpes, 38330 Montbonnot Saint-Martin, France

² Université de Lyon, 69000 Lyon; Université Lyon 1; CNRS, UMR5558, Laboratoire de Biométrie et Biologie Évolutive, 69622 Villeurbanne, France

{gustavo.sacomoto, Marie-France.Sagot}@inria.fr,
vincent.lacroix@univ-lyon1.fr

Abstract. We present a new algorithm for enumerating bubbles with length constraints in directed graphs. This problem arises in transcriptomics, where the question is to identify all alternative splicing events present in a sample of mRNAs sequenced by RNA-seq. This is the first polynomial-delay algorithm for this problem and we show that in practice, it is faster than previous approaches. This enables us to deal with larger instances and therefore to discover novel alternative splicing events, especially long ones, that were previously overseen using existing methods.

1 Introduction

Transcriptomes of model or non model species can now be studied by sequencing, through the use of RNA-seq, a protocol which enables us to obtain, from a sample of RNA transcripts, a (large) collection of (short) sequencing reads, using Next Generation Sequencing (NGS) technologies [15,10]. Nowadays, a typical experiment produces 100M reads of 100nt each. However, the original RNA molecules are longer (typically 500-3000nt) and the general computational problem in the area is then to be able to assemble the reads in order to reconstruct the original set of transcripts. This problem is not trivial for mainly two reasons. First, genomes contain repeats that may be longer than the read length. Hence, a read does not necessarily enable to identify unambiguously the locus from which the transcript was produced. Second, each genomic locus may generate several types of transcripts, either because of genomic variants (i.e. there may exist several alleles for a locus) or because of transcriptomic variants (i.e. alternative splicing or alternative transcription start/end may generate several transcripts from a single locus that differ by the inclusion or exclusion of subsequences). Hence, if a read matches a subsequence shared by several alternative transcripts, it is a priori not possible to decide which of these transcripts generated the read.

General purpose transcriptome assemblers [7,12,14] aim at the general goal of identifying all alternative transcripts, but because of the extensive use of heuristics, they usually fail to identify infrequent transcripts, tend to report several fragments for each gene, or fuse genes that share repeats. Local transcriptome assemblers [13], on the other hand, aim at a simpler goal, as they do not reconstruct full length transcripts. Instead, they focus on reporting all variable regions (polymorphisms): whether genomic (SNPs, indels) or transcriptomic (alternative splicing events). They are much less affected by the issue of repeats, since they focus only on the variable regions. They can afford to be exact and therefore are able to have access to infrequent transcripts. The fundamental idea is that each polymorphism corresponds to a recognizable pattern, called a bubble in the de Bruijn graph built from the RNA-seq reads. In practice, only bubbles with specific length constraints are of interest. However, even with this restriction, the number of such bubbles can be exponential in the size of the graph. Therefore, as with other enumeration problems, the best possible algorithm is one spending time polynomial in the input size between the output of two bubbles, i.e. a polynomial delay algorithm.

In this paper, we introduce the first polynomial delay algorithm to enumerate all bubbles with length constraints in a weighted directed graph. Its complexity in the best theoretical case for general graphs is $O(n(m+n \log n))$ (Section 3) where n is the number of vertices in the graph, m the number of arcs. In the particular case of de Bruijn graphs, the complexity is $O(n(m+n \log \alpha))$ (Section 4.1) where α is a constant related to the length of the skipped part in an alternative splicing event. In practice, an algorithmic solution in $O(nm \log n)$ (Section 4.2) appears to work better on de Bruijn graphs built from such data. We implemented the latter, show that it is more efficient than previous approaches and outline that it enables us to discover novel long alternative splicing events.

2 De Bruijn Graphs and Alternative Splicing

A *de Bruijn graph* (DBG) is a directed graph $G = (V, A)$ whose vertices V are labeled by words of length k over an alphabet Σ . An arc in A links a vertex u to a vertex v if the suffix of length $k-1$ of u is equal to the prefix of v . The out and the in-degree of any vertex are therefore bounded by the size of the alphabet Σ . In the case of NGS data, the k -mers correspond to all words of length k present in the reads of the input dataset, and only those. In relation to the classical de Bruijn graph for all possible words of size k , the DBG for NGS data may then not be complete. Given two vertices s and t in G , an (s, t) -path is a path from s to t . As defined in [3], by an (s, t) -bubble, we mean two vertex-disjoint (s, t) -paths. This definition is, of course, not restricted to de Bruijn graphs.

As was shown in [13], polymorphisms (i.e. variable parts) in a transcriptome (including alternative splicing (AS) events) correspond to recognizable patterns in the DBG that are precisely the (s, t) -bubbles. Intuitively, the variable parts correspond to alternative paths and the common parts correspond to the beginning and end points of those paths. More formally, any process generating

patterns awb and $aw'b$ in the sequences, with $a, b, w, w' \in \Sigma^*$, $|a| \geq k, |b| \geq k$ and w and w' not sharing any k -mer, creates a (s, t) -bubble in the DBG. In the special case of AS events excluding mutually exclusive exons, since w' is empty, one of the paths corresponds to the *junction* of ab , i.e. to k -mers that contain at least one letter of each sequence. Thus the number of vertices of this path in the DBG is predictable: it is at most¹ $k - 1$. An example is given in Fig. 2. In practice [13], an upper bound α to the other path and a lower bound β on both paths is also imposed. In other words, an AS event corresponds to a (s, t) -bubble with paths p_1 and p_2 such that p_1 has at most α vertices, p_2 at most $k - 1$ and both have at least β vertices.

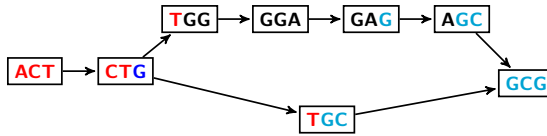


Fig. 1. DBG with $k = 3$ for the sequences: **ACTGGAGCG** (awb) and **ACTGCG** (ab). The pattern in the sequence generates a (s, t) -bubble, from **CTG** to **GCG**. In this case, $b = \mathbf{GCG}$ and $w = \mathbf{GGA}$ have their first letter **G** in common, so the path corresponding to the junction ab has $k - 1 - 1 = 1$ vertex.

Given a directed graph G with *non-negative* arc weights $w : E \mapsto \mathbb{Q}_{\geq 0}$, the length of the path $p = (v_0, v_1) \dots (v_{n-1}, v_n)$ is the sum of the weights of the edges in p and is denoted by $|p|$. The distance, length of the shortest path, from u to v is denoted by $d(u, v)$. We extend the definition of bubble given above.

Definition 1 ($(s, t, \alpha_1, \alpha_2)$ -bubble). A $(s, t, \alpha_1, \alpha_2)$ -bubble in a weighted directed graph is a (s, t) -bubble with paths p_1, p_2 satisfying $|p_1| \leq \alpha_1$ and $|p_2| \leq \alpha_2$.

In practice, when dealing with DBGs built from NGS data, in a lossless pre-processing step, all maximal non-branching linear paths of the graph (i.e. paths containing only vertices with in and out-degree 1) are compressed each into one single vertex, whose label corresponds to the label of the path (i.e. it is the concatenation of the labels of the vertices in the path without the overlapping part(s)). The resulting graph is the *compressed de Bruijn graph* (cDBG). In the cDBG, the vertices can have labels larger than k , but an arc still indicates a suffix-prefix overlap of size $k - 1$. Finally, since the only property of a bubble corresponding to an AS event is the constraint on the length of the path, we can disregard the labels from the cDBG and only keep for each vertex its label length². In this way, searching for bubbles corresponding to AS events in a cDBG

¹ The size is *exactly* $k - 1$ if w has no common prefix with b and no common suffix with a .

² Resulting in a graph with weights in the vertices. Here, however, we consider the weights in the arcs. Since this is more standard and, in our case, both alternatives are equivalent, we can transform one into another by splitting vertices or arcs.

can be seen as a particular case of looking for $(s, t, \alpha_1, \alpha_2)$ -bubbles satisfying the lower bound β in a non-negative weighted directed graph.

Actually, it is not hard to see that the enumeration of $(s, t, \alpha_1, \alpha_2)$ -bubbles, for all s and t , satisfying the lower bound β is NP-hard. Indeed, deciding the existence of at least one $(s, t, \alpha_1, \alpha_2)$ -bubble, for some s and t , with the lower bound β in a weighted directed graph where all the weights are 1 is NP-complete. It follows by a simple reduction from the Hamiltonian st -path problem [6]: given a directed graph $G = (V, E)$ and two vertices s and t , build the graph G' by adding to G the vertices s' and t' , the arcs (s, s') and (t, t') , and a new path from s' to t' with exactly $|V|$ nodes. There is a $(x, y, |V| + 2, |V| + 2)$ -bubble, for some x and y , satisfying the lower bound $\beta = |V| + 2$ in G' if and only if there is a Hamiltonian path from s to t in G .

From now on, we consider the enumeration of all $(s, t, \alpha_1, \alpha_2)$ -bubbles (without the lower bound) for a given source (fixed s) in a non-negative weighted directed graph G (not restricted to a cDBG). The number of vertices and arcs of G is denoted by n and m , respectively.

3 An $O(n(m + n \log n))$ Delay Algorithm

In this section, we present an $O(n(m + n \log n))$ delay algorithm to enumerate, for a fixed source s , all $(s, t, \alpha_1, \alpha_2)$ -bubbles in a general directed graph G with non-negative weights. In a *polynomial delay* enumeration algorithm, the time elapsed between the output of two solutions is polynomial in the instance size. The pseudocode is shown in Algorithm 1. It is important to stress that this pseudocode uses high-level primitives, e.g. the tests in lines 5, 11 and 19. An efficient implementation for the test in line 11, along with its correctness and analysis, is implicitly given in Lemma 3. This is a central result in this section. For its proof we need Lemma 1.

Algorithm 1 uses a recursive strategy, inspired by the binary partition method, that successively divides the solution space at every call until the considered subspace is a singleton. In order to have a more symmetric structure for the subproblems, we define the notion of a *pair of compatible paths*, which is an object that generalizes the definition of a $(s, t, \alpha_1, \alpha_2)$ -bubble. Given two vertices $s_1, s_2 \in V$ and upper bounds $\alpha_1, \alpha_2 \in \mathbb{Q}_{\geq 0}$, the paths $p_1 = s_1 \rightsquigarrow t_1$ and $p_2 = s_2 \rightsquigarrow t_2$ are a *pair of compatible paths* for s_1 and s_2 if $t_1 = t_2$, $|p_1| \leq \alpha_1$, $|p_2| \leq \alpha_2$ and the paths are internally vertex-disjoint. Clearly, every $(s, t, \alpha_1, \alpha_2)$ -bubble is also a pair of compatible paths for $s_1 = s_2 = s$ and some t .

Given a vertex v , the set of out-neighbors of v is denoted by $\delta^+(v)$. Let now $\mathcal{P}_{\alpha_1, \alpha_2}(s_1, s_2, G)$ be the set of all pairs of compatible paths for s_1, s_2, α_1 and α_2 in G . We have³ that:

$$\mathcal{P}_{\alpha_1, \alpha_2}(s_1, s_2, G) = \mathcal{P}_{\alpha_1, \alpha_2}(s_1, s_2, G') \bigcup_{v \in \delta^+(s_2)} (s_2, v) \mathcal{P}_{\alpha_1, \alpha_2'}(s_1, v, G - s_2), \quad (1)$$

³ The same relation is true using s_1 instead of s_2 .

where $\alpha'_2 = \alpha_2 - w(s_2, v)$ and $G' = G - \{(s_2, v) | v \in \delta^+(s_2)\}$. In other words, the set of pairs of compatible paths for s_1 and s_2 can be partitioned into: $\mathcal{P}_{\alpha_1, \alpha'_2}(s_1, v, G - s_2)$, the sets of pairs of paths containing the arc (s_2, v) , for each $v \in \delta^+(s_2)$; and $\mathcal{P}_{\alpha_1, \alpha_2}(s_1, s_2, G')$, the set of pairs of paths that do not contain any of them. Algorithm 1 implements this recursive partition strategy. The solutions are only output in the leaves of the recursion tree (line 3), where the partition is always a singleton. Moreover, in order to guarantee that every leaf in the recursion tree outputs at least one solution, we have to test if $\mathcal{P}_{\alpha_1, \alpha'_2}(s_1, v, G - s_2)$ (and $\mathcal{P}_{\alpha_1, \alpha_2}(s_1, s_2, G')$) is not empty before making the recursive call (lines 11 and 19).

Algorithm 1. `enumerate_bubbles`($s_1, \alpha_1, s_2, \alpha_2, B, G$)

```

1  if  $s_1 = s_2$  then
2  |   if  $B \neq \emptyset$  then
3  |   |   output(B)
4  |   |   return
5  |   else if there is no  $(s, t, \alpha_1, \alpha_2)$ -bubble, where  $s = s_1 = s_2$  then
6  |   |   return
7  |   end
8  end
9  choose  $u \in \{s_1, s_2\}$ , such that  $\delta^+(u) \neq \emptyset$ 
10 for  $v \in \delta^+(u)$  do
11 |   if there is a pair of compatible paths using  $(u, v)$  in  $G$  then
12 |   |   if  $u = s_1$  then
13 |   |   |   enumerate_bubbles( $v, \alpha_1 - w(s_1, v), s_2, \alpha_2, B \cup (s_1, v), G - s_1$ )
14 |   |   |   else
15 |   |   |   enumerate_bubbles( $s_1, \alpha_1, v, \alpha_2 - w(s_2, v), B \cup (s_2, v), G - s_2$ )
16 |   |   |   end
17 |   |   end
18 |   end
19 if there is a pair of compatible paths in  $G - \{(u, v) | v \in \delta^+(u)\}$  then
20 |   enumerate_bubbles( $v, \alpha_1, s_2, \alpha_2, B, G - \{(u, v) | v \in \delta^+(u)\}$ )
21 end
    
```

The correctness of Algorithm 1 follows directly from the relation given in Eq. 1 and the correctness of the tests performed in lines 11 and 19. In the remaining of this section, we describe a possible implementation for the tests, prove correctness and analyze the time complexity. Finally, we prove that Algorithm 1 has an $O(n(m + n \log n))$ delay.

Lemma 1. *There exists a pair of compatible paths for $s_1 \neq s_2$ in G if and only if there exists t such that $d(s_1, t) \leq \alpha_1$ and $d(s_2, t) \leq \alpha_2$.*

Proof. Clearly this is a necessary condition. Let us prove that it is also sufficient. Consider the paths $p_1 = s_1 \rightsquigarrow t$ and $p_2 = s_2 \rightsquigarrow t$, such that $|p_1| \leq \alpha_1$ and $|p_2| \leq \alpha_2$. Let t' be the first vertex in common between p_1 and p_2 . The sub-paths

$p'_1 = s_1 \rightsquigarrow t'$ and $p'_2 = s_2 \rightsquigarrow t'$ are internally vertex-disjoint, and since the weights are non-negative, they also satisfy $|p'_1| \leq |p_1| \leq \alpha_1$ and $|p'_2| \leq |p_2| \leq \alpha_2$.

Using this lemma, we can test for the existence of a pair of compatible paths for $s_1 \neq s_2$ in $O(m + n \log n)$ time. Indeed, let T_1 be a shortest path tree of G rooted in s_1 and truncated at distance α_1 , the same for T_2 , meaning that, for any vertex w in T_1 (resp. T_2), the tree path between s_1 and w (resp. s_2 and w) is a shortest one. It is not difficult to prove that the intersection $T_1 \cap T_2$ is not empty if and only if there is a pair of compatible paths for s_1 and s_2 in G . Moreover, each shortest path tree can be computed in $O(m + n \log n)$ time, using Dijkstra's algorithm [6]. Thus, in order to test for the existence of a $(s, t, \alpha_1, \alpha_2)$ -bubble for some t in G , we can test, for each arc (s, v) outgoing from s , the existence of a pair of compatible paths for $s \neq v$ and v in G . Since s has at most n out-neighbors, we obtain Lemma 2.

Lemma 2. *The test of line 5 can be performed in $O(n(m + n \log n))$.*

The test of line 11 could be implemented using the same idea. For each $v \in \delta^+(u)$, we test for the existence of a pair of compatible paths for, say, $u = s_2$ (the same would apply for s_1) and v in $G - u$, that is v is in the subgraph of G obtained by eliminating from G the vertex u and all the arcs incoming to or outgoing from u . This would lead to a total cost of $O(n(m + n \log n))$ for all tests of line 11 in each call. However, this is not enough to achieve an $O(n(m + n \log n))$ delay. In Lemma 3, we present an improved strategy to perform these tests in $O(m + n \log n)$ total time.

Lemma 3. *The test of line 11, for all $v \in \delta^+(u)$, can be performed in $O(m + n \log n)$ total time.*

Proof. Let us assume that $u = s_2$, the case $u = s_1$ is symmetric. From Lemma 1, for each $v \in \delta^+(u)$, we have that deciding if there exists a pair of compatible paths for s_1 and s_2 in G that uses (u, v) is equivalent to deciding if there exists t satisfying (i) $d(s_1, t) \leq \alpha_1$ and (ii) $d(v, t) \leq \alpha_2 - w(u, v)$ in $G - u$.

First, we compute a shortest path tree rooted in s_1 for $G - u$. Let V_{α_1} be the set of vertices at a distance at most α_1 from s_1 . We build a graph G' by adding a new vertex r to $G - u$, and for each $y \in V_{\alpha_1}$, we add the arcs (y, r) with weight $w(y, r) = 0$. We claim that there exists t in $G - u$ satisfying conditions (i) and (ii) if and only if $d(v, r) \leq \alpha_2 - w(u, v)$ in G' . Indeed, if t satisfies (i) we have that the arc (t, r) is in G' , so $d(t, r) = 0$. From the triangle inequality and (ii), $d(v, r) \leq d(v, t) + d(t, r) = d(v, t) \leq \alpha_2 - w(u, v)$. The other direction is trivial.

Finally, we compute a shortest path tree T_r rooted in r for the reverse graph G'^R , obtained by reversing the direction of the arcs of G' . With T_r , we have the distance from any vertex to r in G' , i.e. we can answer the query $d(v, r) \leq \alpha_2 - w(u, v)$ in constant time. Observe that the construction of T_r depends only on $G - u$, s_1 and α_1 , i.e. T_r is the same for all out-neighbors $v \in \delta^+(u)$. Therefore, we can build T_r only once in $O(m + n \log n)$ time, with two iterations of Dijkstra's algorithm, and use it to answer each test of line 11 in constant time.

Theorem 1. *Algorithm 1 has $O(n(m + n \log n))$ delay.*

Proof. The height of the recursion tree is bounded by $2n$ since at each call the size of the graph is reduced either by one vertex (lines 13 and 15) or all its out-neighborhood (line 20). After at most $2n$ recursive calls, the graph is empty. Since every leaf of the recursion tree outputs a solution and the distance between two leaves is bounded by $4n$, the delay is $O(n)$ multiplied by the cost per node (call) in the recursion tree. From Lemma 1, line 19 takes $O(m + n \log n)$ time, and from Lemma 3, line 11 takes $O(m + n \log n)$ total time. This leads to an $O(m + n \log n)$ time per call, excluding line 5. Lemma 2 states that the cost for the test in line 5 is $O(n(m + n \log n))$, but this line is executed only once, at the root of the recursion tree. Therefore, the delay is $O(n(m + n \log n))$.

4 Implementation and Experimental Results

We now discuss the details necessary for an efficient implementation of Algorithm 1 and the results on two sets of experimental tests. For the first set, our goal is to compare the running time of Dijkstra’s algorithm (for typical DBGs arising from applications) using several priority queue implementations. With the second set, our objective is to compare an implementation of Algorithm 1 to the KISSPLICE algorithm [13]. For both cases, we retrieved from the *Short Read Archive* (accession code ERX141791) 14M Illumina 79bp single-ended reads of a *Drosophila melanogaster* RNA-seq experiment. We then built the de Bruijn graph for this dataset with $k = 31$ using the MINIA algorithm [5]. In order to remove likely sequencing errors, we discarded all k -mers that are present less than 3 times in the dataset. The resulting graph contained 22M k -mers, which after compressing all maximal linear paths, corresponded to 600k vertices.

In order to perform a fair comparison with KISSPLICE, we pre-processed the graph as described in [13]. Namely, we decomposed the underlying undirected graph into biconnected components (BCCs) and compressed all non-branching bubbles with equal path lengths. In the end, after discarding all BCCs with less than 4 vertices (as they cannot contain a bubble), we obtained 7113 BCCs, the largest one containing 24977 vertices. This pre-processing is lossless, i.e. every bubble in the original graph is entirely contained in exactly one BCC. In KISSPLICE, the enumeration is then done in each BCC independently.

4.1 Dijkstra’s Algorithm with Different Priority Queues

Dijkstra’s algorithm is an important subroutine of Algorithm 1 that may have a big influence on its running time. Actually, the time complexity of Algorithm 1 can be written as $O(nc(n, m))$, where $c(n, m)$ is the complexity of Dijkstra’s algorithm. There are several variants of this algorithm [6], with different complexities depending on the priority queue used, including binary heaps ($O(m \log n)$) and Fibonacci heaps ($O(m + n \log n)$). In the particular case where all the weights are non-negative integers bounded by C , Dijkstra’s algorithm can be implemented

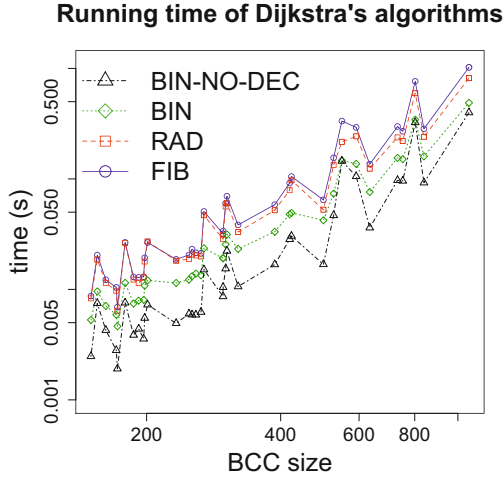


Fig. 2. Running times for each version of Dijkstra’s algorithm: using Fibonacci heaps (FIB), using radix heaps (RAD), using binary heaps (BIN) and using binary heaps without the decrease-key operation (BIN-NO-DEC). The tests were done including all BCCs with more than 150 vertices. Both axes are in logarithmic scale.

using radix heaps ($O(m + n \log C)$) [2]. As stated in Section 2, the weights of the de Bruijn graphs considered here are integer, but not necessarily bounded. However, we can remove from the graph all arcs with weights greater than α_1 since these are not part of any $(s, t, \alpha_1, \alpha_2)$ -bubble. This results in a complexity of $O(m + n \log \alpha_1)$ for Dijkstra’s algorithm.

We implemented four versions of Lemma 2 (for deciding whether there exists a $(s, t, \alpha_1, \alpha_2)$ -bubble for a given s) each using a different version of Dijkstra’s algorithm: with Fibonacci heaps (FIB), with radix heaps (RAD), with binary heaps (BIN) and with binary heaps without decrease-key operation (BIN-NO-DEC). The last version is Dijkstra’s modified in order not to use the decrease-key operation so that we can use a simpler binary heap that does not support such operation [4]. We then ran the four versions, using $\alpha_1 = 1000$ and $\alpha_2 = 2k - 2 = 60$, for each vertex in all the BCCs with more than 150 vertices. The results are shown⁴ in Fig. 2. Contrary to the theoretical predictions, the versions with the best complexities, FIB and RAD, have the worst results on this type of instances. It is clear that the best version is BIN-NO-DEC, which is at least 2.2 times and at most 4.3 times faster than FIB. One of the factors possibly contributing to a better performance of BIN and BIN-NO-DEC is the fact that cDBGs, as stated in Section 2, have bounded degree and are therefore sparse.

⁴ The results for the largest BCC were omitted from the plot to improve the visualization. It took 942.15s for FIB and 419.84s for BIN-NO-DEC.

4.2 Comparison with the KISSPLICE Algorithm

In this section, we compare Algorithm 1 to the KISSPLICE (version 1.8.1) enumeration algorithm [13]. To this purpose, we implemented Algorithm 1 using Dijkstra’s algorithm with binary heaps without the decrease-key operation for all shortest paths computation. In this way, the delay of Algorithm 1 becomes $O(nm \log n)$, which is worse than the one using Fibonacci or radix heaps, but is faster in practice. The goal of the KISSPLICE enumeration is to find all the potential alternative splicing events in a BCC, i.e. to find all $(s, t, \alpha_1, \alpha_2)$ -bubbles satisfying also the lower bound constraint (Section 2). In order to compare KISSPLICE to Algorithm 1, we (naively) modified the latter so that, whenever a $(s, t, \alpha_1, \alpha_2)$ -bubble is found, we check whether it also satisfies the lower bound constraints and output it only if it does.

In KISSPLICE, the upper bound α_1 is an open parameter, $\alpha_2 = k - 1$ and the lower bound is $k - 7$. Moreover, there are two stop conditions: either when more than 10000 $(s, t, \alpha_1, \alpha_2)$ -bubbles satisfying the lower bound constraint have been enumerated or a 900s timeout has been reached. We ran both KISSPLICE (version 1.8.1) and the modified Algorithm 1, with the stop conditions, for all 7113 BCCs, using $\alpha_2 = 60$, a lower bound of 54 and $\alpha_1 = 250, 500, 750$ and 1000. The running times for all BCCs with more than 150 vertices (there are 37) is shown⁵ in Fig. 3. For the BCCs smaller than 150 vertices, both algorithms have comparable (very small) running times. For instance, with $\alpha_1 = 250$, KISSPLICE runs in 17.44s for *all* 7113 BCCs with less than 150 vertices, while Algorithm 1 runs in 15.26s.

The plots in Fig. 3 show a trend of increasing running times for larger BCCs, but the graphs are not very smooth, i.e. there are some sudden decreases and increases in the running times observed. This is in part due to the fact that the time complexity of Algorithm 1 is output sensitive. The delay of the algorithm is $O(nm \log n)$, but the total time complexity is $O(|\mathcal{B}|nm \log n)$, where $|\mathcal{B}|$ is the number of $(s, t, \alpha_1, \alpha_2)$ -bubbles in the graph. The number of bubbles in the graph depends on its internal structure. A large graph does not necessarily have a large number of bubbles, while a small graph may have an exponential number of bubbles. Therefore, the value of $|\mathcal{B}|nm \log n$ can decrease by increasing the size of the graph.

Concerning now the comparison between the algorithms, as we can see in Fig. 3, Algorithm 1 is usually several times faster (keep in mind that the axes are in logarithmic scale) than KISSPLICE, with larger differences when α_1 increases (10 to 1000 times faster when $\alpha_1 = 1000$). In some instances however, KISSPLICE is faster than Algorithm 1, but (with only one exception for $\alpha_1 = 250$ and $\alpha_1 = 500$) they correspond either to very small instances or to cases where only 10000 bubbles were enumerated and the stop condition was met. Finally, using Algorithm 1, the computation finished within 900s for all but 3 BCCs, whereas using KISSPLICE, 11 BCCs remained unfinished after 900s. The improvement in

⁵ The BCCs where *both* algorithms reach the timeout were omitted from the plots to improve the visualization. For $\alpha_1 = 250, 500, 750$ and 1000 there are 1, 2, 3 and 3 BCCs omitted, respectively.

time therefore enables us to have access to bubbles that could not be enumerated with the previous approach.

4.3 On the Usefulness of Larger Values of α_1

In the implementation of KISSPLICE [1], the value of α_1 was experimentally set to 1000 due to performance issues, as indeed the algorithm quickly becomes impractical for larger values. On the other hand, the results of Section 4.2 suggest that Algorithm 1, that is faster than KISSPLICE, can deal with larger values of α_1 . From a biological point of view, it is a priori possible to argue that $\alpha_1 = 1000$ is a reasonable choice, because 87% of annotated exons in *Drosophila* indeed are shorter than 1000nt [11]. However, missing the top 13% may have a big impact on downstream analyses of AS, not to mention the possibility that not yet annotated AS events could be enriched in long skipped exons. In this section, we outline that larger values of α_1 indeed produces more results that are biologically relevant. For this, we exploit another RNA-seq dataset, with deeper coverage.

To this purpose, we retrieved 32M RNA-seq reads from the human brain and 39M from the human liver from the Short Read Archive (accession number ERP000546). Next, we built the de Bruijn graph with $k = 31$ for both datasets, then merged and decomposed the DBG into 5692 BCCs (containing more than 10 vertices). We ran Algorithm 1 for each BCC with $\alpha_1 = 5000$. It took 4min25s for Algorithm 1 to run on all BCCs, whereas KISSPLICE, even using $\alpha_1 = 1000$, took 31min45s, almost 8 times more. There were 59 BCCs containing at least one bubble with the length of the longest path strictly larger than 1000bp potentially corresponding to alternative splicing events. In Fig. 4.3, we show one of those bubbles mapped to the reference genome. It corresponds to an exon skipping in the PRRC2B human gene, the skipped exon containing 2069 bp. While the transcript containing the exon is annotated, the variant with the exon skipped is not annotated.

Furthermore, we ran TRINITY [7] (the most widely used transcriptome assembler) on the same dataset and found that it was unable to report this novel variant. Our method therefore enables us to find new AS events, reported by no other method. This is, of course, just an indication of the usefulness of our approach when compared to a full-transcriptome assembler. A more systematic comparison with TRINITY, as done in [13], is out of the scope of this work.

5 A Natural Generalization

For the sake of theoretical completeness, in this section, we extend the definition of $(s, t, \alpha_1, \alpha_2)$ -bubble to the case where the length constraints concern d vertex-disjoint paths, for an arbitrary but fixed d .

Definition 2 ((s, t, A) - d -bubble). *Let d be a natural number and $A = \{\alpha_1, \dots, \alpha_d\} \subset \mathbb{Q}_{\geq 0}$. Given a directed weighted graph G and two vertices s and t , an (s, t, A) - d -bubble is a set of d pairwise internally vertex-disjoint paths $\{p_1, \dots, p_d\}$, satisfying $p_i = s \rightsquigarrow t$ and $|p_i| \leq \alpha_i$, for all $i \in [1, d]$.*

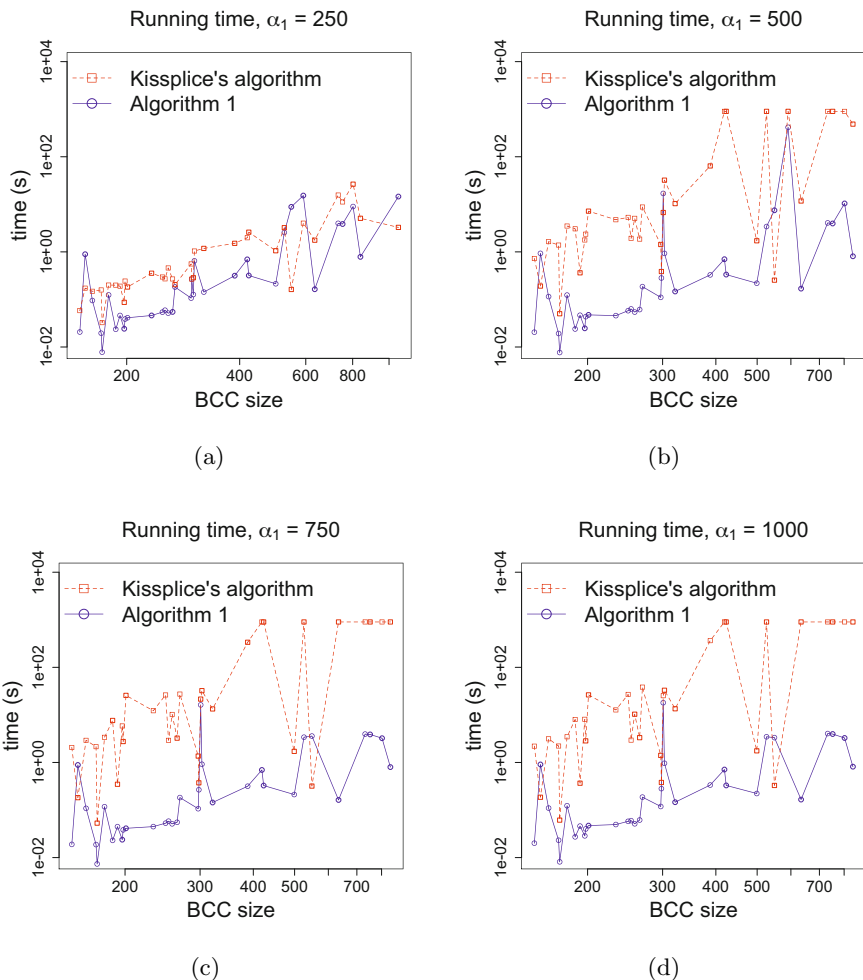


Fig. 3. Running times of Algorithm 1 and of the KISSPLICE algorithm [13] for all the BCCs with more than 150 vertices. Each graph (a), (b), (c) and (d) shows the running time of both algorithms for $\alpha_1 = 250, 500, 750$ and 1000, respectively.

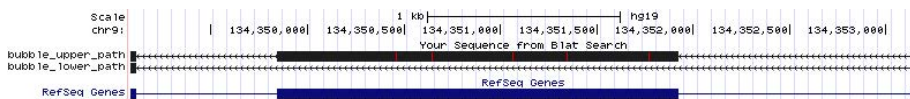


Fig. 4. One of the bubbles with longest path larger than 1000 bp found by Algorithm 1 with the corresponding sequences mapped to the reference genome and visualized using the UCSC Genome Browser. The first two lines correspond to the sequences of, respectively, the shortest (exon exclusion variant) and longest paths of the bubble mapped to the genome. The blue lines are the UCSC human transcript annotations.

Analogously to $(s, t, \alpha_1, \alpha_2)$ -bubbles, we can define two variants of the enumeration problem: all bubbles with a given source (s fixed) and all bubbles with a given source and target (s and t fixed). In both cases, the first step is to decide the existence of at least one (s, t, A) - d -bubble in the graph.

Problem 1 ((s, t, A)- d -bubble decision problem). Given a non-negatively weighted directed graph G , two vertices s, t , a set $A = \{\alpha_1, \dots, \alpha_d\} \subset \mathbb{Q}_{\geq 0}$ and $d \in \mathbb{N}$, decide if there exists a (s, t, A) - d -bubble.

This problem is a generalization of the two-disjoint-paths problem with a min-max objective function, which is NP-complete [9]. More formally, this problem can be stated as follows: given a directed graph G with non-negative weights, two vertices $s, t \in V$, and a maximum length M , decide if there exists a pair of vertex-disjoint paths such that the maximum of their lengths is less than M . The (s, t, A) - d -bubble decision problem, with $A = \{M, M\}$ and $d = 2$, is precisely this problem.

*Problem 2 (($s, *, A$)- d -bubble decision problem).* Given a non-negatively weighted directed graph G , a vertex s , a set $A = \{\alpha_1, \dots, \alpha_d\} \subset \mathbb{Q}_{\geq 0}$ and $d \in \mathbb{N}$, decide if there exists a (s, t, A) - d -bubble, for some $t \in V$.

The two-disjoint-path problem with a min-max objective function is NP-complete even for strictly positive weighted graphs. Let us reduce Problem 2 to it. Consider a graph G with strictly positive weights, two vertices $s, t \in V$, and a maximum length M . Construct the graph G' by adding an arc with weights 0 from s to t and use this as input for the $(s, *, \{M, M, 0\})$ -3-bubble decision problem. Since G has strictly positive weights, the only path with length 0 from s to t in G' is the added arc. Thus, there is a $(s, *, \{M, M, 0\})$ -3-bubble in G' if and only if there are two vertex-disjoint paths in G each with a length $\leq M$.

Therefore, the decision problem for fixed s (Problem 1) is NP-hard for $d \geq 2$, and for fixed s and t (Problem 2) is NP-hard for $d \geq 3$. In other words, the only tractable case is the enumeration of (s, t, A) -2-bubbles with fixed s , the one considered in Section 3.

6 Conclusion

We introduced a polynomial delay algorithm which enumerates all bubbles with length constraints in directed graphs. We show that it is faster than previous approaches and therefore enables us to enumerate more bubbles. These additional bubbles correspond to longer AS events, overseen previously, but biologically very relevant. As shown in [2], by combining radix and Fibonacci heaps in Dijkstra, we can achieve a complexity in $O(n(m + n\sqrt{\log \alpha_1}))$ for Algorithm 1 in cDGBs. The question whether this can be improved, either by improving Dijkstra's algorithm (exploiting more properties of a cDBG) or by using a different approach, remains open.

Acknowledgements. This work was funded by the ANR-12-BS02-0008 (Colib’read); the French project ANR MIRI BLAN08-1335497; and the European Research Council under the European Community’s Seventh Framework Programme (FP7 /2007-2013) / ERC grant agreement no. [247073]10.

References

1. KisSplice’s manual (2013), <http://kissplice.prabi.fr/documentation>
2. Ahuja, R.K., Mehlhorn, K., Orlin, J.B., Tarjan, R.E.: Faster algorithms for the Shortest Path Problem. *J. ACM* 37, 213–223 (1990)
3. Birmelé, E., Crescenzi, P., Ferreira, R.A., Grossi, R., Lacroix, V., Marino, A., Pisanti, N., Sacomoto, G.A.T., Sagot, M.-F.: Efficient bubble enumeration in directed graphs. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 118–129. Springer, Heidelberg (2012)
4. Chen, M., Chowdhury, R.A., Ramachandran, V., Roche, D.L., Tong, L.: Priority Queues and Dijkstra’s Algorithm. Technical Report TR-07-54 (2007)
5. Chikhi, R., Rizk, G.: Space-efficient and exact de bruijn graph representation based on a bloom filter. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS, vol. 7534, pp. 236–248. Springer, Heidelberg (2012)
6. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education (2001)
7. Grabherr, M.G., Haas, B.J., Yassour, M., Levin, J.Z., Thompson, D.A., et al.: Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nat. Biotechnol.* 29, 644–652 (2011)
8. Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G.: De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nat. Genetics* (2012)
9. Li, C.-L., McCormick, S.T., Simchi-Levi, D.: The complexity of finding two disjoint paths with min-max objective function. *Disc. Appl. Math.* (1990)
10. Mortazavi, A., Williams, B., McCue, K., Schaeffer, L., Wold, B.: Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nat. Methods* 5, 621–628 (2008)
11. Pruitt, K., Tatusova, T., Klimke, W., Maglott, D.: NCBI reference sequences: current status, policy and new initiatives. *Nucleic Acids Research* 37(Database-Issue), 32–36 (2009)
12. Robertson, G., Schein, J., Chiu, R., Corbett, R., et al.: De novo assembly and analysis of RNA-seq data. *Nat. Methods* 7, 909–912 (2010)
13. Sacomoto, G.A.T., Kielbassa, J., Chikhi, R., Uricaru, R., Antoniou, P., Sagot, M.-F., Peterlongo, P., Lacroix, V.: KISSPLICE: de-novo calling alternative splicing events from RNA-seq data. *BMC Bioinformatics* 13, S5 (2012)
14. Schulz, M.H., Zerbino, D.R., Vingron, M., Birney, E.: Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels. *Bioinformatics* 28, 1086–1092 (2012)
15. Wang, E., Sandberg, R., Luo, S., Khrebtkova, I., Zhang, L., Mayr, C., Kingsmore, S., Schroth, G.P., Burge, C.: Alternative isoform regulation in human tissue transcriptomes. *Nature* 456(7221), 470–476 (2008)

Distribution of Graph-Distances in Boltzmann Ensembles of RNA Secondary Structures

Rolf Backofen^{1,2}, Markus Fricke³, Manja Marz³,
Jing Qin^{4,*}, and Peter F. Stadler^{4,5,6,7,8}

¹ Department of Computer Science, Chair for Bioinformatics, University of Freiburg,
Georges-Koehler-Allee 106, 79110 Freiburg

² Center for Biological Signaling Studies (BIOSS), Albert-Ludwigs-Universität,
Freiburg, Germany

³ Bioinformatics/High Throughput Analysis Faculty of Mathematics und Computer
Science Friedrich-Schiller-University Jena Leutragraben 1, 07743 Jena

⁴ Max Planck Institute for Mathematics in the Sciences, Inselstraße 22,
04103 Leipzig, Germany

qin@bioinf.uni-leipzig.de

⁵ Bioinformatics Group, Department of Computer Science, and Interdisciplinary
Center for Bioinformatics, University of Leipzig, Härtelstrasse 16-18,
04107 Leipzig, Germany

⁶ Fraunhofer Institut for Cell Therapy and Immunology, Perlickstraße 1, 04103
Leipzig, Germany

⁷ Institute for Theoretical Chemistry, University of Vienna, Währingerstrasse 17,
1090 Vienna, Austria

⁸ Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM87501, USA

Abstract. Large RNA molecules often carry multiple functional domains whose spatial arrangement is an important determinant of their function. Pre-mRNA splicing, furthermore, relies on the spatial proximity of the splice junctions that can be separated by very long introns. Similar effects appear in the processing of RNA virus genomes. Albeit a crude measure, the distribution of spatial distances in thermodynamic equilibrium therefore provides useful information on the overall shape of the molecule can provide insights into the interplay of its functional domains. Spatial distance can be approximated by the graph-distance in RNA secondary structure. We show here that the equilibrium distribution of graph-distances between arbitrary nucleotides can be computed in polynomial time by means of dynamic programming. A naive implementation would yield recursions with a very high time complexity of $O(n^{11})$. Although we were able to reduce this to $O(n^6)$ for many practical applications a further reduction seems difficult. We conclude, therefore, that sampling approaches, which are much easier to implement, are also theoretically favorable for most real-life applications, in particular since these primarily concern long-range interactions in very large RNA molecules.

* Corresponding author.

1 Introduction

The distances distribution within an RNA molecule is of interest in various contexts. Most directly, the question arises whether panhandle-like structures (in which 3' and 5' ends of long RNA molecules are placed in close proximity) are the rule or an exception. Panhandles have been reported in particular for many RNA virus genomes. Several studies [28,8,2,13] agree based on different models that the two ends of single-stranded RNA molecules are typically not far apart. On a more technical level, the problem to compute the partition function over RNA secondary structures with given end-to-end distance d , usually measured as the number of external bases (plus possibly the number of structural domains) arises for instance when predicting nucleic acid secondary structure in the presence of single-stranded binding proteins [9] or in models of RNA subjected to pulling forces (e.g. in atom force microscopy or export through a small pore) [10,23,11]. It also plays a role for the effect of loop energy parameters [7].

In contrast to the end-to-end distance, the graph-distance between two *arbitrarily* prescribed nucleotides in a larger RNA structure does not seem to have been studied in any detail. However, this is of particular interest in the analysis of single-molecule fluorescence resonance energy transfer (smFRET) experiments [25]. This technique allows to monitor the distance between two dye-labeled nucleotides and can reveal details of the kinetics of RNA folding in real time. It measures the non-radiative energy transfer between the dye-labeled donor and acceptor positions. The efficiency of this energy transfer, E_{fret} , strongly depends on the spatial distance R according to $E_{fret} = (1 + (R/R_0)^6)^{-1}$. The Förster radius R_0 sets the length scale, e.g. $R_0 \approx 54 \text{ \AA}$ for the Cy3-Cy5 dye pair. A major obstacle is that, at present, there is no general and efficient way to link smFRET measurements to interpretations in terms of explicit molecular structures. To solve this problem, a natural first step to compute the distribution of spatial distances for an equilibrium ensemble of 3D structures. Since this is not feasible in practice despite major progress in the field of RNA 3D structure prediction [4], we can only resort to considering the graph-distances on the ensemble of RNA secondary structures instead. Although a crude approximation of reality, our initial results indicate that the graph distance can be related to the smFRET data such as those reported by [14]. From a computer science point of view, furthermore, we show here that the distance distribution can be computed exactly using a dynamic programming approach.

2 Theory

2.1 RNA Secondary Structures

An RNA secondary structure is a vertex labeled outerplanar graph $G(V, \xi, E)$, where $V = \{1, 2, \dots, n\}$ is a finite *ordered* set (of nucleotide positions) and $\xi : \{1, 2, \dots, n\} \rightarrow \{\text{A, U, G, C}\}$, $i \mapsto \xi_i$ assigns to each vertex at position i (along the RNA sequence from 5' to 3') the corresponding nucleotide ξ_i . We write $\xi = \xi_1 \dots \xi_n$ for the *sequence* underlying secondary structure and use $\xi[i \dots j] = \xi_i \dots \xi_j$ to denote the *subsequence* from i to j . The edge set E is subdivided into

backbone edges of the form $\{i, i + 1\}$ for $1 \leq i < n$ and a set B of base pairs satisfying the following conditions:

1. If $\{i, j\} \in B$ then $\xi_i \xi_k \in \{\text{GC, CG, AU, UA, GU, UG}\}$.
2. If $\{i, j\} \in B$ then $|j - i| > 3$.
3. If $\{i, j\}, \{i, k\} \in B$ then $j = k$.
4. If $\{i, j\}, \{k, l\} \in B$ and $i < k < j$ then $i < l < j$.

The first condition allows base pairs only for Watson-Crick and GU base pairs. The second condition implements the minimal steric requirement for an RNA to bend back on itself. The third condition enforces that B forms a matching in the secondary structure. The last condition (nesting condition) forbids crossing base pairs, i.e. pseudoknots.

The nesting condition results in a natural partial order in the set of base pairs B defined as $\{i, j\} \prec \{k, l\}$ if $k < i < j < l$. In particular, given an arbitrary vertex k , the set $B_k = \{\{i, j\} \in B \mid i \leq k \leq j\}$ of base pairs enclosing k is totally ordered. Note that k is explicitly allowed to be incident to its enclosing base pairs. A vertex k is *external* if $B_k = \emptyset$. A base pair $\{k, l\}$ is *external* if $B_k = B_l = \{\{k, l\}\}$.

Consider a fixed secondary structure G , for a given base pair $\{i, j\} \in B$, we say a vertex k is *accessible* from $\{i, j\}$ if $i < k < j$ and there is no other pair $\{i', j'\} \in B$ such that $i < i' < k < j' < j$. The unique subgraph $\mathcal{L}_{i,j}$ induced by i, j , and all the vertices accessible from $\{i, j\}$ is known as the *loop* of $\{i, j\}$. The *type* of a loop $\mathcal{L}_{i,j}$ is uniquely determined depending on whether $\{i, j\}$ is external or not, and the numbers of unpaired vertices and base pairs. For details, see [26]. Each secondary structure G has a unique set of loops $\{\mathcal{L}_{i,j} \mid \{i, j\} \in B\}$, which is called the *loop decomposition* of G . The free energy $f(G)$ of a given secondary structure, according to the standard energy model [20], is defined as the sum of the energies of all loops in its unique loop decomposition.

The relative location of two vertices v and w in G is determined by the base pairs B_v and B_w that enclose them. If $B_v \cap B_w \neq \emptyset$, there is a unique \prec -minimal base pair $\{i_{v,w}, j_{v,w}\}$ that encloses both vertices and thus a uniquely defined loop $\mathcal{L}_{\{i_{v,w}, j_{v,w}\}}$ in the loop associated with v and w . If $B_v \setminus B_w = \emptyset$ or $B_w \setminus B_v = \emptyset$ then v or w is unpaired and part of $\mathcal{L}_{\{i_{v,w}, j_{v,w}\}}$. Otherwise, i.e. $B_v \cap B_w = \emptyset$, there are uniquely defined \prec -maximal base pairs $\{k_v, l_v\} \in B_v \setminus B_w$ and $\{k_w, l_w\} \in B_w \setminus B_v$ that enclose v and w , respectively. This simple partition holds the key to computing distance distinguished partition functions below.

It will be convenient in the following to introduce edge weights $\omega_{i,j} = a$ if $j = i + 1$, i.e., for backbone edges, and $\omega_{i,j} = b$ for $\{i, j\} \in B$. Given a path p , we define the weight of the path $d(p)$ as the sum of the weights of edges in the path. The (weighted) *graph-distance* $d_{v,w}^G$ in G is defined as the weight of the path p connecting v and w with $d(p)$ being minimal. For the weights, we require the following condition:

- (W) If i and j are connected by an edge, then $\{i, j\} \in E$ is the unique shortest path between i and j .

This condition ensures that single edges cannot be replaced by detours of shorter weight. Condition (W) and property (ii) of the secondary structure graphs

implies $b < 3a$ because the closing base pair must be shorter than a hairpin loop. Furthermore, considering a stacked pair we need $b < b + 2a$, i.e. $a > 0$. We allow the degenerate case $b = 0$ that neglects the traversals of base pairs.

2.2 Boltzmann Distribution of Graph-Distances

For a fixed structure G , $d_{v,w}^G$ is easy to compute. Here, we are interested in the distribution $Pr[d_{v,w}^G|\xi]$ and its expected value $d_{v,w} = E[d_{v,w}^G|\xi]$ over the ensemble of all possible structures G for a given sequence ξ . Both quantities can be calculated from the Boltzmann distribution $Pr[G|\xi] = e^{-f(G)/RT}/Q$ where $Q = \sum_G e^{-f(G)/RT}$ denotes the partition function of the ensemble of structures. As first shown in [21], Q and related quantities can be computed in cubic time. A crucial quantity for our task is the restricted partition function

$$Z^{v,w}[d] = \sum_{G \text{ with } d_{v,w}^G = d} e^{-f(G)/RT}$$

for a given pair v, w of positions in a given RNA sequence ξ . A simple but tedious computation (Appendix A¹) verifies that the $Pr[d_{v,w}^G = d|\xi] = Z^{v,w}[d]/Q$ and $d_{v,w} = E[d_{v,w}^G|\xi] = \sum_d (Z^{v,w}[d]/Q)d$. Hence it suffices to compute $Z^{v,w}[d]$ for $d = 1, \dots, n$. In sections 2.3-2.5 we show that this can be achieved by a variant of McCaskill's approach [21].

For the ease of presentation we describe in the following only the recursion for the simplified energy model for the "circular maximum matching" matching, in which energy contributions are associated with individual base pairs rather than loops. Our approach easily extends to the full model by using separating the partition functions into distinct cases for the loop types. We use the letter Z to denote partition functions with distance constraints, while Q is used for quantities that appear in McCaskill's algorithm and are considered as pre-computed here.

Before we continue with the calculation of the partition function, let's first look into problem formulation in more detail. For the FRET application, it is well-known that the rate which with FRET occurs is correlated with the distance. Therefore, only a limited range of distance changes (e.g. $20\text{\AA} - 100\text{\AA}$ for Cy3-Cy5) can be reported by the FRET experiments. Thus the more useful formulation of our problem is not to use the full expected quantity for all positions. Instead, we are interested in the average for all distances within some threshold θ_d . As the space and time complexity will depend on the number of distances we consider, we will parametrise our complexity by the number of nucleotides n and the number of overall distances considered $D = \theta_d + 1$, as well.

2.3 Recursions of $Z^{v,w}[d]$: v and w Are External

An important special case assumes that both v and w are external. This is case e.g. when v and w are bound by proteins. In particular, the problem of computing

¹ The Appendix A-D of our paper are available from <http://www.rna.uni-jena.de/supplements/RNAGraphdist/supplement.pdf>

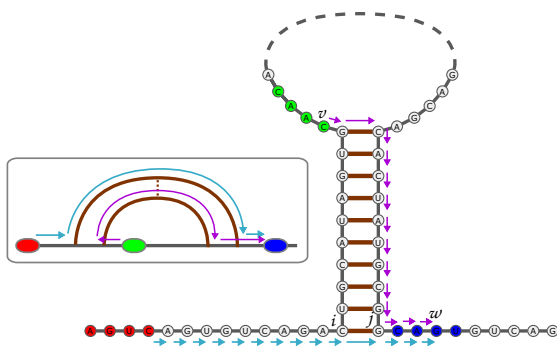


Fig. 1. Inside and outside paths. The shortest path (violet arrows) from v (green) to w (blue) is not an inside path: *inside* emphasizes that, in contrast to the shortest path (cyan arrows) between the red region and w , it is not contained in the interval determined by its end points.

By our initialization of Z^I , we can simply define $Z^{I'}$ by

$$Z_{i,j}^{I'}[d] = Z_{i+1,j-1}^I[d - 2a] \quad (2)$$

Note that if $Z_{i,j}^{I'}[d]$ is called with $j = i + 1$, then we call $Z_{i+1,i}^I[d - 2a]$. The only admissible value again is the correct value $d = a$.

This recursion requires $O(Dn^3)$ time and space. It is possible to reduce the complexity in this special case by a linear factor. The trick is to use conditional probabilities for arcs starting at i or the conditional probability for i to be single-stranded, which can be determined from the partition function for RNA folding [2], see Appendix B.

2.4 Recursions of $Z^{v,w}[d]$: The General Case

The minimal distance between two positions that are covered by an arc can be realized by *inside paths* and *outside paths*. This complicates the algorithmic approach, since both types of paths must be controlled simultaneously. Consider Fig. 1. The shortest path between the green and blue regions includes some vertices outside the interval between these two regions. The basic idea is to generalize Equation (1) to computing the partition function $Z^{v,w}[d]$. The main question now becomes how to recurse over decompositions of both the inside and the outside paths.

Fig. 1 shows that the outside paths are important for the green region, i.e., the region that is covered by an arc. Hence, we have to consider the different cases that the two positions v and w are covered by arcs. The set Ω of all secondary structures on ξ can be divided into two disjoint subclasses that have to be treated differently:

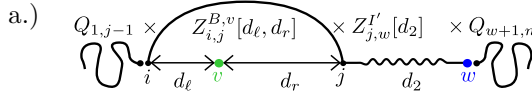
Ω_0 v and w are not enclosed in a common base pair, i.e., $B_v \cap B_w = \emptyset$.

Ω_1 there is a base pair enclosing both v and w , i.e., $B_v \cap B_w \neq \emptyset$.

Note that this bipartition explicitly depends on v and w . In the following, we will first introduce the recursions that are required in Ω_0 structures to compute $Z^{v,w}[d]$.

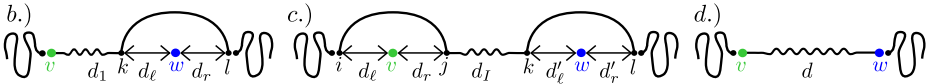
Contribution of Ω_0 structures to $Z^{v,w}[d]$ One example of this case is given in Fig. 1 with the red and blue region, where v (vertex in green region) is covered by an arc, and w (vertex in blue region) is external. Denote the \prec -maximal base pair enclosing v by $\{i, j\}$. Since at most one of v and w is covered by an arc, we know that $j < w$. Hence, every path p from v to w , and hence also the shortest paths (not necessarily unique) must run through the right end j of the arc $\{i, j\}$. More precisely, there must sub-paths p_1 and p_2 with $d(p) = d(p_1) + d(p_2) + a$ such that $v \xrightarrow{p} w \rightarrow v \xrightarrow{p_1} j - (j + 1) \xrightarrow{p_2} w$, where $i \xrightarrow{p} j$ denotes that p is a shortest path from i to j and $-$ denotes a single backbone edge. For the shortest path from v to j , it consists either of a shortest path $v \xrightarrow{p'} i$ and the arc $\{i, j\}$, or it goes directly to j without using the arc $\{i, j\}$.

How does this distinction translate to the partition function approach? If we want to calculate the contribution of this case to the partition function $Z^{v,w}[d]$, we have to split both the sequence $\xi[i, w]$ and distance d as follows



where $Z'_{j,w}[d_2]$ is the partition function starting and ending with a single-stranded base as defined in Equation (2), and $Z_{i,j}^{B,v}[d_l, d_r]$ is the partition function consisting of all structures of $\xi[i, j]$ containing the base pair $\{i, j\}$ with the property that the shortest path from v to i has length d_l and the shortest path from v to j has length d_r . In addition, d, d_r and d_2 must satisfy $d = d_r + d_2$.

The remaining cases for the contribution of the class Ω_0 to $Z^{v,w}[d]$ are given by all other possible combinations of v and w being single-stranded or being covered by an arc, i.e.,



To simplify, we extend the definition of $Z_{i,j}^{B,v}[d_l, d_r]$ by setting $Z_{v,v}^{B,v}[0, 0] = 1$ and $Z_{v,v}^{B,v}[d_l, d_r] = 0$ for $d_l + d_r > 0$. This allows us to conveniently model all cases where either v or w are external, i.e., a.), b.), and d.), as special cases of c.).

In case c.) we have to split the distance d for all combinations of i, j, v, w . This would result in an $O(n^6 D^5)$ algorithm. A careful inspection shows, however, that the split of the distances for the arcs into d_l and d_r is unnecessary. Since we want to know only distance to the left/right end overall, we can simply introduce two matrices $Z_{i,j}^{B,v,\ell}[d]$ and $Z_{i,j}^{B,v,r}[d]$ that store these values. These matrices can be generated from $Z_{i,j}^{B,v}[d_l, d_r]$ as follows:

$$Z_{i,j}^{B,v,\ell}[d] = \sum_{\substack{d_r \\ d_r + b \geq d}} Z_{i,j}^{B,v}[d, d_r] + \sum_{\substack{d_l \\ d_l > d}} Z_{i,j}^{B,v}[d_l, d - b]$$

Analogously, we compute $Z_{i,j}^{B,v,r}[d]$.

Overall, the contribution to $Z^{v,w}[d]$ for structures in Ω^0 is given by

$$Z_0^{v,w}[d] = \sum_{\substack{d_1, d_2 \\ d_1 + d_2 \leq d}} \sum_{\substack{i, j, k, l \\ i \leq v \leq j < k \leq w \leq l}} \begin{pmatrix} Q_{1, i-1} \cdot Z_{i, j}^{B, v, r}[d_1] \\ \cdot Z_{j, k}^{I'}[d - (d_1 + d_2)] \\ \cdot Z_{k, l}^{B, w, \ell}[d_2] \cdot Q_{l+1, n} \end{pmatrix} \quad (3)$$

Note that for splitting the distance, we reuse the same indices (e.g., the j in $Z_{i, j}^{B, v, r}[d_1] \cdot Z_{j, k}^{I'}[d - (d_1 + d_2)]$), where as for the remaining partition function, we use successive indices (e.g., the i in $Q_{1, i-1} \cdot Z_{i, j}^{B, v, r}[d_1]$). This difference comes from the fact that splitting a sequence into subsequences is done naturally between two successive indices, whereas splitting a distance is naturally done by splitting at an individual position. We have only to guarantee that the substructures which participate in the split do agree on the structural context of the split position. This is guaranteed by requiring that $Z^{I'}$ starts and ends with a backbone edge. We note that the incorporation of the full dangling end parameters makes is more tedious to handle the splitting positions.

This results in a complexity of $O(n^6 D^3)$ time and $O(n^3 D)$ space. However, we do not need to split in i, j, k, j simultaneously. Instead, we could split case (c) at position j and introduce for all $v \leq j$ and $k \leq w$ the auxiliary variables

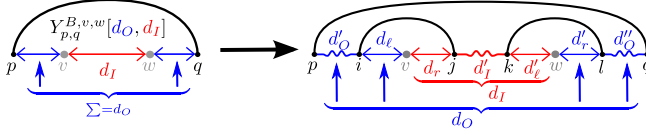
$$\begin{aligned} Z_{1, j}^{B, v, r}[d_1] &= \sum_{i \leq v} Q_{1, i-1} \cdot Z_{i, j}^{B, v, r}[d_1] & Z_{k, n}^{B, w, \ell}[d_2] &= \sum_{w \leq l} Z_{k, l}^{B, w, \ell}[d_2] \cdot Q_{l+1, n} \\ Z_{j, n}^{IB, w, \ell}[d'] &= \sum_{k > j} \sum_{d_2 \leq d'} Z_{j, k}^{I'}[d' - d_2] \cdot Z_{k, n}^{B, w, \ell}[d_2]. \end{aligned}$$

Finally, we can replace recursion (3) by

$$Z_0^{v,w}[d] = \sum_{v \leq j} \sum_{d_1 \leq d} Z_{1, j}^{B, v, r}[d_1] \cdot Z_{j, n}^{IB, w, \ell}[d - d_1] \quad (4)$$

We thus arrive at $O(n^4 D^2)$ time and $O(n^3 D)$ space complexity for the contribution of Ω_0 structures to $Z^{v,w}[d]$, excluding the complexity of computing $Z_{i, j}^{B, v}[d_\ell, d_r]$.

Contribution of Ω_1 Structures to $Z^{v,w}[d]$ Ω_1 contains all cases where v and w are covered by a base pair. In the following, let $\{p, q\}$ be the \prec -minimal base pair covering v and w . In principle, this case looks similar to the overall case for Ω_0 . However, we have now to deal not only with an inside distance, but also with an outside distance over the base pair $\{p, q\}$. Thus, we need to store the partition function for all inside and outside for each \prec -minimal arc $\{p, q\}$ that covers v and w , which we will call $Y_{p, q}^{B, v, w}[d_O, d_I]$. In principle, a similar recursion as defined for Z_0 in equation (3) can be derived, with the additional complication since we have to take care of the additional outside distance due to the arc (p, q) . Thus, we obtain the following splitting:



Again we can avoid the complexity of simultaneously splitting at $\{i, j\}$ and $\{k, l\}$ by doing a major split after j . Thus, we get the equivalent recursions as in eqns.(5–7):

$$Y_{p,j}^{B,v,r}[d, d_r] = \sum_{p < i \leq v} \sum_{d'_O \leq d} Z_{p,i}^{I'}[d'_O] \cdot Z_{i,j}^{B,v} \overbrace{[d - d'_O, d_r]}^{\cong d_\ell} \quad (5)$$

$$Y_{k,q}^{B,w,\ell}[d'_\ell, d] = \sum_{w \leq l < q} \sum_{d''_O \leq d} Z_{k,l}^{B,w} [d'_\ell, d - d''_O] \cdot Z_{l,q}^{I'} [d''_O] \quad (6)$$

$$Y_{j,q}^{IB,w,\ell}[d'_I, d] = \sum_{j < k < q} \sum_{d'_\ell \leq d'_I} Z_{j,k}^{I'} [d'_I - d'_\ell] \cdot Y_{k,q}^{B,w,\ell} [d'_\ell, d] \quad (7)$$

Overall, we get the following recursion:

$$Z_{p,q}^{v,w} [d_O, d_I] = \sum_{v \leq j} \sum_{\substack{d_r \leq d_I \\ d \leq d_O}} Y_{p,j}^{B,v,r} [d, d_r] \cdot Y_{q,j}^{IB,w,\ell} [d_I - d_r, d_O - d] \quad (8)$$

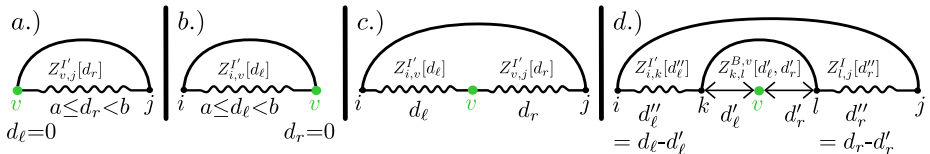
Overall, we can now define $Z^{v,w}[d]$ by

$$Z^{v,w} [d] = Z_0^{v,w} [d] + \sum_{\substack{\{p,q\} \neq \{v,w\} \\ d_I \geq d+b}} Z_{p,q}^{v,w} [d, d_I] + \sum_{\substack{\{p,q\} \neq \{v,w\} \\ d < d_O+b}} Z_{p,q}^{v,w} [d_O, d]$$

This part has now a complexity of $O(n^4 D^2)$ space and $O(n^5 D^4)$ time. For practical applications, however, we do not need to consider all possible $\{p, q\}$. Instead, there are only few base pairs that are likely to form *and* that cover v, w , especially for v, w where the internal distance of v, w is large enough such that an outside path has to be considered at all. If we assume a constant number of such long-range base-pairs, then the complexity is reduced by an n^2 -factor. For the complexity in terms of distance, recall that D is typically small.

2.5 Recursions for $Z_{i,j}^{B,v}[d_\ell, d_r]$

So far, we have used $Z_{i,j}^{B,v}[d_\ell, d_r]$ as a black box. In order to compute these terms, we distinguish the limiting cases a.) $v = i$, b.) $v = j$, c.) is external from the generic case d.):



Starting from the limiting cases, we initialize $Z_{v,j}^{B,v}[0, d_r]$ as follows:

$$Z_{v,j}^{B,v}[0, d_r] = \begin{cases} Z_{v,j}^{I'}[d_r] & \text{for } a \leq d_r < b \\ \sum_{d' \geq b} Z_{v,j}^{I'}[d'] & \text{for } d_r = b \\ 0 & \text{otherwise} \end{cases}$$

and analogously for $Z_{i,v}^{B,v}[d_\ell, 0]$. Furthermore, $Z_{i,j}^{B,v}[0, 0] = 0$ for $i \neq v \neq j$. Finally, we have the following recursion for $i \neq v \neq j$, $d_\ell > 0$ and $d_r > 0$:

$$Z_{i,j}^{B,v}[d_\ell, d_r] = \hat{Q}_{i,j}^b \cdot \sum_{\substack{k \neq l \\ i < k \leq v \\ v \leq l < j}} \sum_{\substack{d'_\ell \leq d_\ell \\ d'_r \leq d_r}} Z_{i,k}^{I'}[d_\ell - d'_\ell] \cdot Z_{k,l}^{B,v}[d'_\ell, d'_r] \cdot Z_{l,j}^{I'}[d_r - d'_r] \quad (9)$$

where $\hat{Q}_{i,j}^b$ is the external partition function over all structures on the union of the intervals $\xi[1..i] \cup \xi[j..n]$ so that $\{i, j\}$ is a base pair. This is equivalent to $\hat{Q}_{i,j}^b = Pr(\{i, j\}) \times Q/Q_{i,j}^b$. The base pair probability $Pr(\{i, j\})$, and the partition functions Q and $Q_{i,j}^b$ are computed by means of McCaskill's algorithm.

Recursion (9) apparently has complexity $O(n^5 D^4)$ in time and $O(n^3 D^2)$ in space. This can be reduced due to the strong dependency between d_ℓ and d_r , however. By construction we have $|d_\ell - d_r| \leq b$ since we can always use the bond $\{i, j\}$ to traverse from one end to the other. Furthermore, assuming integer values for a and b , we can have only $c_b = 2b/\text{lcd}(a, b) + 1$ different values for $(d_\ell - d_r)$. This implies that the space complexity of $Z_{i,j}^{B,v}[d_\ell, d_r]$ is $O(n^3 D c_b)$. Instead of $Z_{i,j}^{B,v}[d_\ell, d_r]$, we store $Z_{i,j}^{B,v}[d_\ell, d_\ell + d_{\text{add}}]$ for the c_b possible values of d_{add} .

The dependency between d_ℓ and d_r can also be used to reduce the time complexity in Equ.(9). The problematic case is (d). Instead of using the variables d_ℓ and d_r in $Z_{i,j}^{B,v}[d_\ell, d_r]$ we use the pair d_ℓ, d_{add} in $Z_{i,j}^{B,v}[d_\ell, d_\ell + d_{\text{add}}]$. Similarly, we use d'_ℓ, d'_{add} instead of d'_ℓ, d'_r for the inner base pair, which then determines completely the splitting the distances. The details are relegated to Appendix C. Overall, this results in an recursion for $Z_{i,j}^{B,v}[d_\ell, d_\ell + d_{\text{add}}]$ with complexity $O(n^5 c_b^2)$ time and $O(n^3 D c_b)$ space.

3 Discussion and Applications

The theoretical analysis of the distance distribution problem shows that, while polynomial-time algorithms exist, they probably cannot be improved to space and time complexities that make them widely applicable to large RNA molecules. Due to the unfavorable time complexity of the current algorithm and the associated exact implementation in C, a rather simple and efficient sampling algorithm has been implemented. We resort to sampling Boltzmann-weighted secondary structures with `RNAsubopt -p` [17], which uses the same stochastic backtracing approach as `sfold` [5]. As the graph-distance for a pair of nucleotides in a given secondary structure can be computed in $O(n \log n)$ time, even large samples can be evaluated efficiently².

² The C++ program `RNAgraphdist` is available from <http://www.rna.uni-jena.de/supplements/RNAgraphdist/RNAgraphdist1.0.tar.gz>

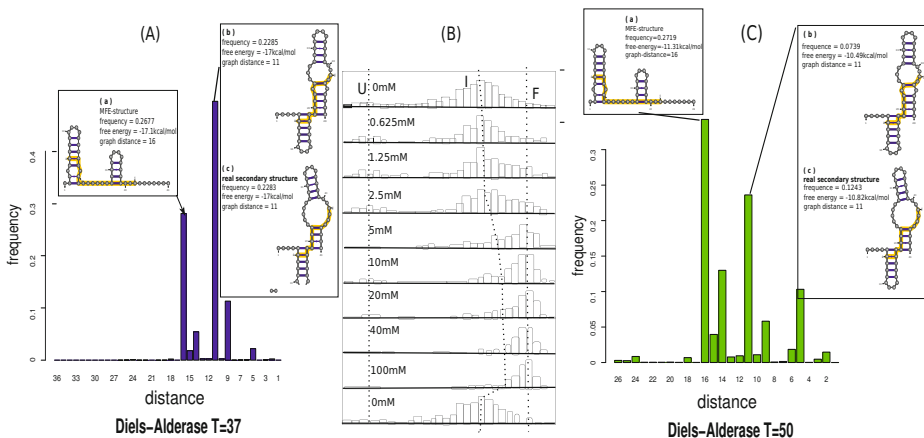


Fig. 2. Relation between graph distance distribution and smFRET data. (A) The graph distance distribution of a Diels-Alderase ribozyme at temperature 37°C. Structures (a), (b) and (c) are the top three secondary structures considering their free energy. In which, the minimum free energy structure is showed in (a), (c) is the real secondary structure which is ranked as the 3rd best sub-optimal structure with `RNAsubopt -e`. The graphic representations of these structures are produced with `VARNA` [3]. (B) The corresponding smFRET efficiency (E_{fret}) histograms are reported in [14]. From these data, three separate states of the DAse ribozyme can be distinguished, the unfolded (U), intermediate (I) and folded (F) states. (C) The graph distance distribution in the ensemble which is approximated with `RNAsubopt -p` at temperature 50°C.

As we pointed out in the introduction, the graph distance measure introduced in this paper can serve as a first step towards a structural interpretation of smFRET data. As an example, we consider the graph distance distribution of a Diels-Alderase (DAse) ribozyme (Fig. 2 (A)). Histograms of smFRET efficiency (E_{fret}) for this 49 nt long catalytic RNA are reported in [14] for a large number of surface-immobilized ribozyme molecules as a function of the Mg²⁺ concentration in the buffer solution. A sketch of their histograms is displayed in Fig. 2 (B). The dyes are attached to sequence positions 6 (Cy3) and 42 (Cy5) and hence do not simply reflect the end-to-end distance, Fig. 2 (A)(c). In this example, we observe the expected correspondence small graph distances with a strong smFRET signal. This is a particular interesting example, since the minimal free energy (mfe) structure (Fig. 2 (A)(a)) predicted with `RNAfold` is not identified with the real secondary structure (Fig. 2 (A)(c)). In fact, the ground state secondary structure is ranked as the 3rd best sub-optimal structure derived via `RNAsubopt -e`. The free energy difference between these two structures is only 0.1 kcal/mol . However, their graph distances show a relatively larger difference. The 2nd best sub-optimal structure (Fig. 2 (A)(b)) looks rather similar with the 3rd structure, in particular, they share the same graph distance value.

The smFRET data of [14] indicate the presence of three sub-populations, corresponding to three different structural states: folded molecules (state F),

intermediate conformation (state I) and unfolded molecules (state U). In the absence of Mg^{2+} , the I state dominates, and only small fractions are found in states U and F. Unfortunately, the salt dependence of RNA folding is complex [15,19] and currently is not properly modeled in the available folding programs. We can, however, make use of the qualitative correspondence of low salt concentrations with high temperature. In Fig. 2 (C) we therefore re-compute the graph distance distribution in the ensemble at an elevated temperature of $50^{\circ}C$. Here, the real structure becomes the second best structure with free energy $-10.82kcal/mol$ and we observe a much larger fraction of (nearly) unfolded structures with longer distances between the two beacon positions. Qualitatively, this matches the smFRET data showed in Fig. 2 (B).

Long-range interactions play an important role in pre-mRNA splicing and in the regulation of alternative splicing [1,22], bringing splice donor, acceptor, branching site into close spatial proximity. Fig. 3(A) shows for *D. melanogaster* pre-mRNAs that the distribution of graph-distances between donor and acceptor sites shifted towards smaller values compared to randomly selected pairs of positions with the same distance.³ Although the effect is small, it shows a clear difference between the real RNA sequences and artificial sequences that were randomized by di-nucleotide shuffling.

The spatial organization of the genomic and sub-genomic RNAs is important for the processing and functioning of many RNA viruses. This goes far beyond the well-known panhandle structures. In *Coronavirus* the interactions of the 5' TRS-L cis-acting element with body TRS elements has been proposed as an important determinant for the correct assembly of the *Coronavirus* genes in the host [6]. The matrix of expected graph-distances in Fig. 3(B) shows that TRS-L and TRS-B are indeed placed near each other. More detailed information is provided in Appendix (D).

Our first results show that the systematic analysis of the graph-distance distribution both for individual RNAs and their aggregation over ensembles of structures can provide useful insights into structural influences on RNA function. These may not be obvious directly from the structures due to the inherent difficulties of predicting long-range base pairs with sufficient accuracy and the many issues inherent in comparing RNA structures of very disparate lengths.

Due the complexity of algorithm we have refrained from attempting a direct implementation in an imperative programming language. Instead, we are aiming at an implementation in Haskell that allows us to make use of the framework of algebraic dynamic programming [12]. The graph distance measure and the associated algorithm can be extended in principle to of RNA secondary structures with additional tertiary structural elements such as pseudoknots [24] and G-quadruples [18]. RNA-RNA interaction structures [16] also form a promising area for future extensions. We note finally, that the Fourier transition method introduced in [27] could be employed to achieve a further speedup.

³ Due to the insufficiency of the spacial-distance information of structural elements in the secondary structures, we artificially choose $a = b = 1$ in our experiments.

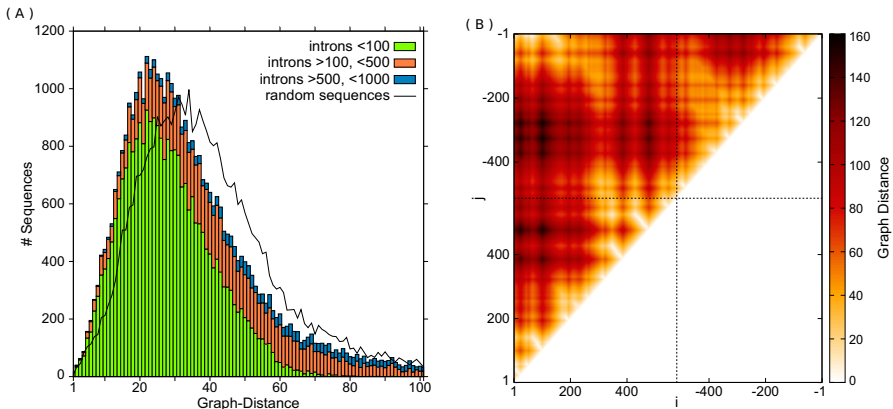


Fig. 3. (A): Distribution of graph-distances ($a = b = 1$) in *Drosophila melanogaster* pre-mRNAs between the first and last intron position. To save computational resources, pre-mRNAs were truncated to 100 nt flanking sequence. The black curve shows the graph-distance distribution computed for the corresponding pairs of positions on sequences that were randomized by di-nucleotide shuffling. **(B):** Graph-distances ($a = b = 1$) within and between the 5' and 3' regions of the genomic RNA of human *Coronavirus* 229E computed from a concatenation of position 1–576 and 25188–25688. Secondary structures bring the 5' TRS-L and 3' TRS-B elements into close proximity. More detailed information related to this example can be found in Supplemental Material D.

Acknowledgements. This work was supported in part by the *Deutsche Forschungsgemeinschaft* proj. nos. BA 2168/2-2, STA 850/10-2, SPP 1596 and MA5082/1-1.

References

1. Baraniak, A.P., Lasda, E.L., Wagner, E.J., Garcia-Blanco, M.A.: A stem structure in fibroblast growth factor receptor 2 transcripts mediates cell-type-specific splicing by approximating intronic control elements. *Mol. Cell Biol.* 23, 9327–9337 (2003)
2. Clote, P., Ponty, Y., Steyaert, J.M.: Expected distance between terminal nucleotides of RNA secondary structures. *J. Math. Biol.* 65, 581–599 (2012)
3. Darty, K., Denise, A., Ponty, Y. VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics* 25(15), 1974–1975 (2009)
4. Das, R., Baker, D.: Automated de novo prediction of native-like RNA tertiary structures. *Proc. Natl. Acad. Sci. USA* 104, 14664–14669 (2007)
5. Ding, Y., Lawrence, C.E.: A statistical sampling algorithm for RNA secondary structure prediction. *Nucl. Acids Res.* 31(24), 7280–7301 (2003)
6. Dufour, D., Mateos-Gomez, P.A., Enjuanes, L., Gallego, J., Sola, I.: Structure and functional relevance of a transcription-regulating sequence involved in coronavirus discontinuous RNA synthesis. *J. Virol.* 85(10), 4963–4973 (2011)
7. Einert, T.R., Näger, P., Orland, H., Netz, R.: Impact of loop statistics on the thermodynamics of RNA folding. *Phys. Rev. Lett.* 101, 048103 (2008)
8. Fang, L.T.: The end-to-end distance of RNA as a randomly self-paired polymer. *J. Theor. Biol.* 280, 101–107 (2011)

9. Forties, R.A., Bundschuh, R.: Modeling the interplay of single-stranded binding proteins and nucleic acid secondary structure. *Bioinformatics* 26, 61–67 (2010)
10. Gerland, U., Bundschuh, R., Hwa, T.: Force-induced denaturation of RNA. *Biophys. J.* 81, 1324–1332 (2001)
11. Gerland, U., Bundschuh, R., Hwa, T.: Translocation of structured polynucleotides through nanopores. *Phys. Biol.* 1, 19–26 (2004)
12. Giegerich, R., Meyer, C.: Algebraic dynamic programming. In: Kirchner, H., Ringeissen, C. (eds.) *AMAST 2002*. LNCS, vol. 2422, pp. 349–364. Springer, Heidelberg (2002)
13. Han, H.S., Reidys, C.M.: The 5'-3' distance of RNA secondary structures. *J. Comput. Biol.* 19, 867–878 (2012)
14. Kobitski, A., Nierth, A., Helm, M., Jaschke, A., Nienhaus, U.G.: Mg²⁺-dependent folding of a Diels-Alderase ribozyme probed by single-molecule FRET analysis. *Nucleic Acids Res.* 35(6), 2047–2059 (2007)
15. Leipply, D., Lambert, D., Draper, D.E.: Ion-RNA interactions thermodynamic analysis of the effects of mono- and divalent ions on RNA conformational equilibria. *Methods Enzymol.* 469, 433–463 (2009)
16. Li, A.X., Marz, M., Qin, J., Reidys, C.M.: RNA-RNA interaction prediction based on multiple sequence alignments. *Bioinformatics* 27(4), 456–463 (2011)
17. Lorenz, R., Bernhart, S.H., Höner zu Siederdisen, C., Tafer, H., Flamm, C., Stadler, P.F., Hofacker, I.L.: ViennaRNA Package 2.0. *Alg. Mol. Biol.* 6, 26 (2011)
18. Lorenz, R., Bernhart, S.H., Qin, J., Honer zu Siederdisen, C., Tanzer, A., Amman, F., Hofacker, I.L.: 2d meets 4g: G-quadruplexes in rna secondary structure prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 99(PrePrints), 1 (2013)
19. Mathews, D., Sabina, J., Zuker, M., Turner, D.H.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol.* 288, 911–940 (1999)
20. Mathews, D.H., Disney, M.D., Childs, J.L., Schroeder, S.J., Zuker, M., Turner, D.H.: Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proc. Natl. Acad. Sci. USA* 101, 7287–7292 (2004)
21. McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers* 29(6-7), 1105–1119 (1990)
22. McManus, C.J., Graveley, B.R.: RNA structure and the mechanisms of alternative splicing. *Curr. Opin. Genet. Dev.* 21, 373–379 (2011)
23. Müller, M., Krzakala, F., Mézard, M.: The secondary structure of RNA under tension. *Eur. Phys. J. E* 9, 67–77 (2002)
24. Reidys, C.M., Huang, F.W.D., Andersen, J.E., Penner, R.C., Stadler, P.F., Nebel, M.E.: Topology and prediction of RNA pseudoknots. *Bioinformatics* 27(8), 1076–1085 (2011)
25. Roy, R., Hohng, S., Ha, T.: A practical guide to single-molecule FRET. *Nature Methods* 5, 507–516 (2008)
26. Schuster, P., Fontana, W., Stadler, P.F., Hofacker, I.L.: From sequences to shapes and back: a case study in RNA secondary structures. *Proc. Royal Society London B* 255(1344), 279–284 (1994)
27. Senter, E., Sheikh, S., Dotu, I., Ponty, Y., Clote, P.: Using the Fast Fourier Transform to Accelerate the Computational Search for RNA Conformational Switches. *PLoS ONE* 7(12), e50506 (2012)
28. Yoffe, A.M., Prinsen, P., Gelbart, W.M., Ben-Shaul, A.: The ends of a large RNA molecule are necessarily close. *Nucl. Acids Res.* 39, 292–299 (2011)

Faster Algorithms for RNA-Folding Using the Four-Russians Method

Balaji Venkatachalam, Dan Gusfield, and Yelena Frid

Department of Computer Science, UC Davis
{balaji,gusfield,yafrid}@cs.ucdavis.edu

Abstract. The secondary structure that maximizes the number of non-crossing matchings between complementary bases of an RNA sequence of length n can be computed in $O(n^3)$ time using Nussinov's dynamic programming algorithm. The Four-Russians method is a technique that will reduce the running time for certain dynamic programming algorithms by a multiplicative factor after a preprocessing step where solutions to all smaller subproblems of a fixed size are exhaustively enumerated and solved. Frid and Gusfield designed an $O(\frac{n^3}{\log n})$ algorithm for RNA folding using the Four-Russians technique. In their algorithm the preprocessing is interleaved with the algorithm computation.

We simplify the algorithm and the analysis by doing the preprocessing once prior to the algorithm computation. We call this the *two-vector* method. We also show variants where instead of exhaustive preprocessing, we only solve the subproblems encountered in the main algorithm once and memoize the results. We give a simple proof of correctness and explore the practical advantages over the earlier method.

The Nussinov algorithm admits an $O(n^2)$ time parallel algorithm. We show a parallel algorithm using the two-vector idea that improves the time bound to $O(\frac{n^2}{\log n})$.

We have implemented the parallel algorithm on graphics processing units using the CUDA platform. We discuss the organization of the data structures to exploit coalesced memory access for fast running times. The ideas to organize the data structures also help in improving the running time of the serial algorithms. For sequences of length up to 6000 bases the parallel algorithm takes only about 2.5 seconds and the two-vector serial method takes about 57 seconds on a desktop and 15 seconds on a server. Among the serial algorithms, the two-vector and memoized versions are faster than the Frid-Gusfield algorithm by a factor of 3, and are faster than Nussinov by up to a factor of 20.

1 Introduction

Computational approaches to find the secondary structure of RNA molecules are used extensively in bioinformatics applications. The classic dynamic programming (DP) algorithm proposed in the 1970s has been central to most structure prediction algorithms. While the objective of the original algorithm was to maximize the number of non-crossing pairings between complementary bases, the

dynamic programming approach has been used for other models and approaches, including minimizing the free energy of a structure. The DP algorithm runs in cubic time and there have been many attempts at improving its running time. Here, we use the Four-Russians method for speeding up the computation.

The Four-Russians method, named after Aralazarov et al. [4], is a method to speed up certain dynamic programming algorithms. In a typical Four-Russians algorithm there is a preprocessing step that exhaustively enumerates and solves a set of subproblems and the results are tabled. In the main DP algorithm, instead of filling out or inspecting individual cells, the algorithm takes longer strides in the table. The computation for multiple cells is solved in constant time by utilizing the preprocessed solutions to the subproblems. The longer strides to fill the table reduce the runtime by a multiplicative factor. The size of the subproblems is chosen in a way that does not make the preprocessing too expensive.

Frid and Gusfield [11] showed the application of the Four-Russians approach for RNA folding. In their algorithm, the preprocessing is interleaved with the algorithm computation. They fill out a part of the DP table and use these entries to complete a part of the preprocessing. The preprocessed entries are used later in the computation.

We show a simpler algorithm where all the preprocessing is completed before the start of the main algorithm. This simplifies the correctness proof and the runtime analysis. This approach helps in obtaining a $\log n$ factor improvement for the parallel algorithm. In comparing various methods for RNA folding, Zakov and Frid (personal communication) had independently observed that the algorithm in [11] could be modified to do the preprocessing at once. It is essentially the idea as described here.

In this paper we explore the implications of the one-pass preprocessing idea. This description of the algorithm leads naturally to two other variants. We empirically evaluate these variants and also the implementation of the parallel algorithm.

The parallel architecture of general-purpose graphical processing units (GPUs) have been exploited for many real-world application in addition to applications in gaming and visualization problems. GPUs have also been used to speed up RNA folding algorithms [6,23,24]. Here we show how the Four-Russians method allows an organization of the data structures for fast memory accesses. We also describe the organization of the parallel hierarchy to exploit the inherent parallelism of the solution.

In the rest of the section, we describe the problem in relation to the other problems in RNA folding. To keep the paper self-contained, we will first describe the *two-vector algorithm*, our application of the Four-Russians method to the RNA folding problem. We will use that description to describe the original Four-Russians method for RNA folding by Frid and Gusfield [11]. This discussion leads to two other variants where the preprocessing is done on demand, instead of the exhaustive preprocessing in the two-vector method and the Frid-Gusfield algorithm. In section 4 we discuss the $O(n^2/\log n)$ parallel algorithm.

We will then describe the implementation of a parallel algorithm using CUDA. The final sections have discussion on empirical observations and conclusions. Due to space limitations, this manuscript focuses mostly on the theoretical aspects and describes the experimental results briefly. Detailed discussion can be found in [26].

Related Work. The $O(n^3)$ dynamic programming algorithm due to Nussinov et al. [21,20] maximizes the number of non-crossing matching complimentary bases. There have been many methods since Zuker and Stiegler [31] that infer the folding using thermodynamic parameters [25,19] which are more realistic than maximizing the number of base pairs. These methods have been implemented in many packages including UNAFold [18], Mfold [30], Vienna RNA Package [15], RNAstructure [22].

Probabilistic methods include stochastic context-free grammars [10,9], the maximum expected accuracy (MEA) method, where secondary structures are composed of pairs that have a maximal sum of pairing probabilities, eg., MaxExpect [17], Pfold [16], CONTRAfold [8] which maximize the posterior probabilities of base pairs; and Sfold [7], CentroidFold [14] that maximize the centroid estimator. There are also other methods that use a combination of thermodynamic and statistical parameters [2] and methods that use training sets of known folds to determine their parameters, eg., CONTRAfold [8], and Simfold[3] and ContextFold[28].

In addition to the Four-Russians method, other methods to improve the running time include Valiant's max-plus matrix multiplication by Akutsu [1] and Zakov et al. [29]; and sparsification, where the branch points are pruned to get an improved time bound [27,5].

CUDA, the programming platform for GPGPUs, has been used to solve many bioinformatics problems. Chang, Kimmer and Ouyang [6] and Stojanovski, Gjorgjevikj and Madjarov [24] show an implementation of the Nussinov algorithm on CUDA. Rizk et al. [23] describe the implementation for Zuker and Stiegler method involving energy parameters. These methods are discussed in section 5.2.

2 The Nussinov Algorithm

In this paper, we consider the basic RNA folding problem of maximizing the number of non-crossing complimentary base pair matchings. Complimentary bases can be paired, i.e., A with U and C with G. A set of disjoint pairs is a matching. The pairs in a matching must not cross, i.e., if bases in positions i and j are paired and if bases k and l are paired, then either they are nested, i.e., $i < k < l < j$ or they are non-intersecting, i.e., $i < j < k < l$. The objective is to maximize the number of pairings under these constraints.

The following algorithm, due to Nussinov [21] maximizes the number of non-crossing matchings. For an input sequence S of length n over the alphabet A, C, G, U, the recurrence is defined as follows. Let $D(i, j)$ denote the optimal cost

of folding for the subsequence from i to j . For all i , $D(i, i - 1) = D(i, i) = 0$ and for all $i < j$:

$$D(i, j) = \max \begin{cases} b(S(i), S(j)) + D(i + 1, j - 1) \\ \max_{i+1 \leq k \leq j} D(i, k - 1) + D(k, j) \end{cases} \quad (1)$$

where $b(., .) = 1$ for complimentary bases and 0 otherwise. The DP table is the upper triangular part of the $n \times n$ matrix. The optimal solution is given by $D(1, n)$. The table can be filled column-wise from the first column till the n^{th} . There are other ways of filling the table too, eg., along the diagonals — the (i, i) -diagonal first, $(i, i + 1)$ -diagonal next and so on, until the last diagonal with one entry, $D(1, n)$. To allow for traceback we need to store the bases that are paired to get the maximum value. Let $D^*(i, j)$ denote the corresponding indices. These are obtained by substituting $\arg \max$ in place of \max in the above recurrence and can be computed along with the \max value.

The first part of the recurrence can be solved in constant time. The second part is more expensive, incurring $\Theta(n)$ look ups and maximum computations. There are $O(n^2)$ entries in the DP table and each cell can be computed in $O(n)$ time, giving an $O(n^3)$ time algorithm.

3 The Four-Russians Algorithms

In this section we discuss three variants of the Four-Russians algorithm. We will first describe the *two-vector* approach. Since it is simpler than the other methods we will use the description to discuss two other variants.

3.1 Two-Vector Algorithm

To apply the Four-Russians technique we start with the following observation:

Lemma 1. *The values along a column from bottom to top and along a row from left to right are monotonically non-decreasing. Consecutive cells differ at most by 1.*

Proof. Consider neighboring cells (i, j) and $(i + 1, j)$. $D(i, j)$ represents the solution of a longer sequence than $D(i + 1, j)$. Therefore the former value should be at least as large as the latter. Suppose $D(i, j)$ differed from $D(i + 1, j)$ by more than one. Then we can remove any matching for i . This has at most one fewer base pair matching and is a valid solution for the subsequence $(i + 1, j)$ with a larger value than its current value, contradicting the optimality of $D(i + 1, j)$. An analogous argument holds along the columns.

Once the cells $D(i, l)$, $D(i, l + 1)$, \dots , $D(i, l + q - 1)$ are computed, for some $l \in \{i, \dots, j - q\}$, they can be represented by $D(i, l) + V_0$, $D(i, l) + V_1$, \dots , $D(i, l) + V_{q-1}$, where $V_p = D(i, l + p) - D(i, l)$, for $p \in \{0, \dots, q - 1\}$. Let us define, $v_0 = 0$ and $v_p = V_p - V_{p-1}$, for $p \in \{1, \dots, q - 1\}$. From lemma 1, $v_p \in \{0, 1\}$, for all

$p \in [0, q - 1]$. Let \mathbf{v} denote the binary vector v_0, v_1, \dots, v_{q-1} of differences and let \mathbf{V} denote the vector of running totals V_0, V_1, \dots, V_{q-1} .

Since the v_p 's are defined from V_p 's, the inverse function is well defined: $V_p = \sum_{k=0}^i v_k$. Thus $D(i, l)$ together with the vector \mathbf{v} represents q consecutive cells of the table.

Similarly, since the values are non-increasing down a column, $D(i + l + 1, j), \dots, D(i + l + q, j)$ be represented by the pair $D(i + l + 1, j), \bar{\mathbf{v}}$, where $\bar{\mathbf{v}} \in \{0, -1\}^q$. We call \mathbf{v} the *horizontal difference vector* or the *horizontal vector* and we call $\bar{\mathbf{v}}$ the *vertical difference vector* or the *column vector*. The corresponding vector of sums is denoted \bar{V} .

Consider q consecutive cells from $l + 1$ to $l + q$ used in computing $D(i, j)$:

$$D(i, j) \leftarrow \max_{l+1 \leq k \leq l+q} D(i, k - 1) + D(k, j) \quad (2)$$

$$\begin{aligned} &\leftarrow \max_{0 \leq k \leq q-1} D(i, l) + V_k + D(i + l + 1, j) + \bar{V}_k \\ &\leftarrow D(i, l) + D(i + l + 1, j) + \max_{0 \leq k \leq q-1} V_k + \bar{V}_k \quad (3) \end{aligned}$$

As before, we use $\arg \max$ in place of \max to obtain $D^*(i, j)$, which facilitates the traceback.

As noted above the second line of the recurrence (1), looping over elements, is more expensive and we will use (3) instead of (2) to compute the D and D^* values in the Four-Russians method. That is, we will use (3) for groups of q cells each instead of one loop of (1). Since the V vectors are in bijection with the \mathbf{v} vectors, we will do the preprocessing using \mathbf{v} . Let \mathbf{v} and $\bar{\mathbf{v}}$ be the corresponding vectors in (3). The following algorithm evaluates the max computation.

Input: horizontal difference vector \mathbf{v} and vertical difference vector $\bar{\mathbf{v}}$

```

1:  $max\text{-val} \leftarrow 0$  and  $max\text{-index} \leftarrow 0$ 
2:  $sum_1 \leftarrow 0$  and  $sum_2 \leftarrow 0$ 
3: for  $k = 0$  to  $q - 1$  do
4:    $sum_1 \leftarrow sum_1 + v_i$ 
5:    $sum_2 \leftarrow sum_2 + \bar{v}_i$ 
6:   if  $sum_1 + sum_2 > max\text{-val}$  then
7:      $max\text{-val} \leftarrow sum_1 + sum_2$ 
8:      $max\text{-index} \leftarrow k$ 
9:   end if
10: end for
11: return ( $max\text{-val}, max\text{-index}$ )

```

Using this instead of (2) is not advantageous in itself. However, if this algorithm is given as a black box, $D(i, j)$ can be computed in constant time by invoking the black box once. In the preprocessing stage, we will run the above algorithm for all possible vector pairs of length q and store the results in table R . Table R is indexed by a pair of numbers in the range $[2^q]$ to represent the two vectors $(\mathbf{v}, \bar{\mathbf{v}})$. Since there are two entries in the table, the lookup is a constant

time operation. We will show later that this exhaustive enumeration is not too expensive.

In the Nussinov algorithm described in the previous section, the recurrence is evaluated using (2) and it takes $O(q)$ time. In the Four-Russians method, using the preprocessing step, the max computation is available through a table lookup and the recurrence for q terms can be completed in constant time. This reduction in the computation time is the reason for the speedup by a factor of q .

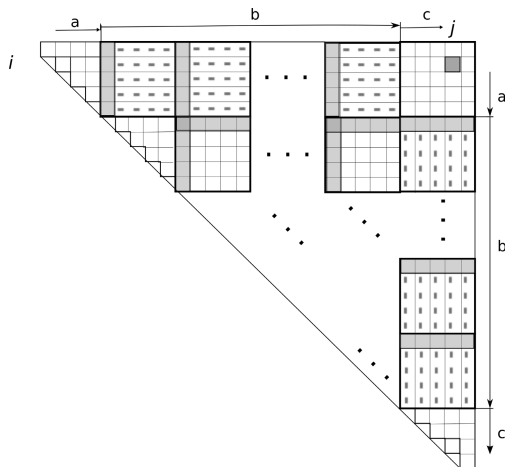


Fig. 1. A diagrammatic representation of the two-vector method. The row and column blocks are matched as labelled. The gray boxes and the gray dashes show the initial value and difference vectors. The group of cells in b correspond to the Four-Russians loop in lines 15–19 of Algorithm 1; the cells in a are used in the loop in lines 9–11 and the cells in c form the loop in lines 12–14.

The two-vector method modifies the Nussinov algorithm as follows. All the rows and columns of the table are grouped into groups of q cells each. The recurrence over these q cells is computed in constant time using the preprocessing table. The recurrence involves $D(i, k - 1) + D(k, j)$, i.e., the value in the $(k - 1)^{\text{st}}$ column is used with the k^{th} row. Therefore the row and column groupings differ by one. That is, the columns are grouped $(0, 1, \dots, q - 1)$, $(q, q + 1, \dots, 2q - 1)$ etc. The rows are grouped $(1, 2, \dots, q)$, $(q + 1, q + 2, \dots, 2q)$ etc. This ensures that the row and column groups are well characterized. That is, to fill the cell (i, j) , the k^{th} group along row i needs to be combined with the k^{th} group below (i, j) in column j .

The cells of the table are filled in the same order as before. When the last cell of a row- or a column- group is evaluated the corresponding row and column vectors are computed and stored. To fill cell (i, j) , we retrieve the first element and the horizontal vector of the group from row i and the first element and the column vector from the corresponding group in column j . The recurrence is solved using (3) by a table lookup. The final value for $D(i, j)$ is the maximum

Algorithm 1. Procedure for the two-vector Four-Russians speedup. The DP table is filled column-wise.

```

1:  $R \leftarrow$  preprocess all pairs of vectors of length  $q$ 
2: for  $j = 1$  to  $n$  do
3:    $D(j, j) \leftarrow 0$ 
4:   for  $i = j - 1$  down to 1 do
5:      $D(i, j) \leftarrow b(S[i], S[j]) + D(i + 1, j - 1)$ 
6:     Let  $(i, i)$  be in the  $I^{\text{th}}$  group in row  $i$ .
7:     Let  $(i, j)$  be in the  $J^{\text{th}}$  group horizontally in the  $i^{\text{th}}$  row and  $J'^{\text{th}}$  group
       vertically in the  $j^{\text{th}}$  column.
8:     Let  $i_q$  be the right-most entry of group  $I$  and  $j_q$  be the left-most entry
       in group  $J$ 
9:     for  $k = i + 1$  to  $i_q$  do           // For all cells in the first group
10:       $D(i, j) \leftarrow \max(D(i, j), D(i, k - 1) + D(k, j))$ 
11:    end for
12:    for  $k = j_q$  to  $j$  do           // For all cells in the last group
13:       $D(i, j) \leftarrow \max(D(i, j), D(i, k - 1) + D(k, j))$ 
14:    end for
15:    for  $k = 1$  to  $J - I$  do           // For all groups in between
16:      Let  $p$  be the left-most cell in the  $k^{\text{th}}$  group to the right of  $I$  and  $q$ 
       be the top-most cell in the  $k^{\text{th}}$  group below  $J'$ .
17:      Let  $v_p$  and  $v_q$  be the corresponding horizontal and vertical difference
       vectors.
18:       $D(i, j) \leftarrow \max(D(i, j), D(i, p) + D(q, j) + R(v_p, v_q))$ 
19:    end for
20:    if  $i \bmod q = 0$  then           // compute the vertical difference vector
21:      compute and store the  $\mathbf{v}$  vector  $i/q^{\text{th}}$  group for column  $j$ 
22:    end if
23:    if  $j \bmod q = q - 1$  then       // compute the horizontal difference
       vector
24:      compute and store the  $\mathbf{v}$  vector  $(j - 1)/q^{\text{th}}$  group for row  $i$ 
25:    end if
26:  end for
27: end for

```

value over all the groups. There might be residual elements in the row that do not fall in these groups. There are at most $2q$ such elements. These are solved separately using Nussinov's method. Algorithm 1 has the algorithm listing and Figure 1 describes the algorithm pictorially.

Runtime Analysis. In the precomputation phase, there are 2^q q -length vectors and 2^{2q} pairs of vectors. The precomputation takes $O(q)$ time per vector pair. Thus the total time for precomputation is $O(q2^{2q})$.

The main algorithm: There are $O(n^2)$ cells and to fill each cell it takes $O(n/q + q)$ time. That is, it takes $O(n/q)$ time to look up the initial value and the difference vector and the R table lookups for the $O(n/q)$ groups. It takes $O(q)$ time for the residual elements. Thus it takes $O(n^2 \times (n/q + q))$ time to

fill the table. Every cell is involved in at most two vector computations, where the difference to its neighbor is computed once for the row and for the column vector. This takes an amortized $O(n^2)$ time which is dominated by the rest of the algorithm.

When $q = \log n$, the total time for the entire algorithm is $O(\log n 2^{2 \log n} + n^2 + n^2 \times (\frac{n}{\log n} + \log n)) = O(n^2 \log n + n^3 / \log n) = O(n^3 / \log n)$.

3.2 Other Variants

FG Algorithm. Frid and Gusfield [11] first showed how the Four-Russians approach could be applied to the RNA-folding problem. We will call their algorithm the *FG* algorithm. *FG* and *two-vector* algorithms are variants of the same idea. We will highlight the differences in preprocessing and the maximum value computation by the Four-Russians technique. In particular, we will show the maximum computation in step 18 of Algorithm 1.

After computing the q -contiguous cells of a group in a row, the value in the initial cell $D(i, p)$ and the horizontal difference vector v_p are known. They run the preprocessing algorithm in page 130 for this fixed v_p vector together with all possible vertical difference vectors. They add the value of $D(i, p)$ to the maximum and table the result. This preprocessing step is computed for every block of every row. The preprocessing table R is indexed by row number, group number and a vector (which is a potential column vector). The horizontal vectors need not be stored.

To fill cell (i, j) , they iterate over all groups and find the q -length column vectors. The preprocessed value for this vector in the corresponding block is retrieved from the table and the result is added to $D(q, j)$.

The preprocessing is for horizontal vectors seen in the table. Since the horizontal vectors are not known beforehand, the precomputation cannot be done prior to the main algorithm. Instead, it is interleaved with the computation of the table. They fill part of the DP table and use the vectors to complete some preprocessing, which in turn is used fill another part of the table and so on.

Since the preprocessing is done for every group of every row, the same horizontal vector can be seen multiple times in the table. This leads to duplicated work and slower running time than the two-vector algorithm.

Memoization. The two-vector method computes the preprocessing over all possible vector pairs and the *FG* method for only the horizontal vectors that are seen in the table. Stated this way, a hybrid approach suggests itself.

In our next variants, we memoize the results for a pair of vectors. Like the two-vector approach, the preprocessing is done only once for a vector pair and like the *FG* algorithm, it is only for the vectors seen in the table and the preprocessing is interleaved with the main algorithm. Since the preprocessing table is indexed by two vectors, unlike the *FG* algorithm, the results are computed only once for every vector seen.

In the partially memoized version, upon completion of elements of a group, if a new horizontal vector is seen, we pair it with all possible 2^q column vectors and

the results are tabled. In the completely memoized version, the result for a pair of vectors is computed the first time the pair is observed and the result is stored in the table. The result for future occurrences of the same pair are obtained by a table lookup. the rest of the algorithm is identical to the two-vector method.

All these variants take $O(n^3/\log n)$ time but the memoized versions potentially store fewer vectors than the two vector method and will have a similar worst-case runtime in practice as the two-vector method. But, as argued before, the FG method does duplicated work and will be slower in practice.

4 Parallel Algorithm

The Nussinov DP algorithm can be parallelized with n processes to get an $O(n^2)$ parallel algorithm. We assign one parallel process to a column. In the i^{th} iteration, each process computes the value for the i^{th} diagonal entry. That is, the successive diagonals are solved in iterations and in each iteration the entries of the diagonal are solved in parallel. To compute the value for cell (i, j) , the entries in the row to its left and in the column below (i, j) are needed. Since these values are computed in earlier iterations, each diagonal cell can be filled independent of the other processes.

A process has to compute the value for $O(n)$ cells and for each cell it needs to access $O(n)$ other cells. Thus the total computation takes $O(n^2)$ time with n processes.

The parallel algorithm for process j for $j = 1, 2, \dots, n$:

```

1:  $D(j + 1, j) \leftarrow 0, D(j, j) \leftarrow 0$ 
2: for  $i = j$  down to 1 do
3:    $D(i, j) \leftarrow D(i + 1, j - 1) + b(S[i], S[j])$ 
4:   for  $k = i + 1$  to  $j$  do
5:      $D(i, j) \leftarrow \max\{D(i, j), D(i, k - 1) + D(k, j)\}$ 
6:   end for
7:   Synchronize with other processes
8: end for

```

We will describe the use the two-vector Four-Russians method to obtain an $O(n^2/\log n)$ algorithm below. The preprocessing step that enumerates the solution for $2^q \times 2^q$ difference vectors is embarrassingly parallel and we do not discuss the parallel algorithm for it.

As before, we have n processes one for each column. Each process solves the entries of the column from bottom to top. Instead of computing the maximum over each cell in the inner loop (lines 3 – 5 in the parallel algorithm above), we use the Four-Russians technique to solve q cells in one step by looking up the table computed in the preprocessing step.

Let $d_H(i, j)$ be the horizontal difference vector for cells $D(i, j), \dots, D(i + q - 1, j)$ and let $d_V(i, j)$ be the vertical difference for cells $D(i, j), \dots, D(i + q - 1, j)$. We modify the inner loop of the parallel algorithm as follows:

- 1: **for** $k' = 0$ to $\lfloor j/q \rfloor - 1$ **do**
- 2: $k = i + k' * q$
- 3: $D(i, j) = \max\{D(i, j), D(i, k) + D(k + 1, j) + R[d_H(i, k)][d_V(k + 1, j)]\}$
- 4: **end for**
- 5: **for** $k = \lfloor j/q \rfloor \times q$ to j **do**
- 6: $D(i, j) \leftarrow \max\{D(i, j), D(i, k) + D(k + 1, j)\}$
- 7: **end for**
- 8: Compute the horizontal and vertical differences and store them in $d_H(i - q + 1, j)$ and $d_V(i, j)$ respectively.

For each entry, the first loop takes $O(n/q)$ time and the second loop takes $O(q)$ time. Since all the processes are solving the k^{th} diagonal in the k^{th} iteration, all of them execute the same number of steps before synchronization. Note that we compute the horizontal and vertical differences for every node, unlike in section 3.1 where they are computed every q^{th} cell, to ensure that every process performs the same number of steps and simplify the analysis. The difference vectors can be computed in $O(q)$ time. These can also be computed in constant time by shifting the previous difference vector and appending the new difference. But we will not assume this simplification for the time bound computation.

Thus each entry can be computed in $O(n/q + q)$ time. There are $O(n)$ entries for each process, thus the total time taken for all processes to terminate is $O(n^2/q + nq)$. With $q = \log n$ as before, this gives an $O(n^2/\log n)$ algorithm.

5 Parallel Implementation

5.1 GPU Architecture

Graphics processing units (GPUs) are specialized processors designed for computationally intensive real-time graphics rendering. Compute Unified Device Architecture (CUDA) is the computing engine designed by NVIDIA for their GPUs.

The programmer can group threads in a *block*, which in turn can be organized in a *grid* hierarchy. Memory hierarchy includes thread-specific local memory, block-level shared memory for all threads in the block and global memory for the entire grid. The access times increases along the hierarchy from local to global memory.

Since the access to global memory is slower (more clock cycles than local memory access), it is efficient for the threads within a block to access contiguous memory locations. Then the hardware *coalesces* memory accesses for all threads in a block into one request. More specifically, in our application, if a matrix is stored in row-major order and if the threads in a block access contiguous elements of a row, then the accesses can be coalesced. However, accessing elements along a

column is inefficient as distant memory elements have to be fetched from different cache lines.

Programs that observe the hardware specifications can exploit the optimizations in the system and are fast in practice. We designed the program that exploits the parallel structure of the DP algorithm and the hardware features of the GPU.

5.2 Related Work

As mentioned earlier, the cells of a diagonal are independent of one another and can be computed in parallel. In Stojanovski et al. [24], elements of the diagonal are assigned to a block of threads. This design does not handle memory coalescence for either row or column accesses. Chang et al. [6] allocate an $n \times n$ table and reflect the upper-triangular part of the matrix on the main diagonal. Successive elements of a column are fetched from the row in the reflected part of the matrix. When threads of a block are assigned to elements of a diagonal, the successive column accesses for a thread are to consecutive memory cells. However, this does not allow coalesced access for threads within a block. Rizk and Lavenier [23] show an implementation for RNA folding under energy models. They show a tiling scheme where a group of cells are assigned to a block of threads to reuse the data values that are fetched from a column. In this paper, we show that storing the row and column vectors in different orders for two-vector method can further improve the efficiency.

5.3 Design of the Four-Russians CUDA Program

We briefly describe the design of the CUDA program; a longer discussion can be found in [26].

We group cells together into tiles, where each tile is a composite of $q \times q$ cells. The tiles along a diagonal can be computed independent of each other. Each tile is assigned to a block of threads and computed in parallel. After all the entries of the tile are computed, only the horizontal and vertical differences are stored.

To fill a tile, the horizontal differences of all the tiles to the left and vertical differences from the tiles underneath are accessed. These difference vectors are stored in different orders. The horizontal difference vectors of the rows of a tile are stored in contiguous memory locations and the tiles are stored in row-major order. The vertical difference vectors of the columns of a tile are stored together and the tiles are grouped in column-major order.

To fill a tile, the horizontal difference vectors from a tile to the left are fetched. When each thread retrieves one vector, the block of threads accesses contiguous memory locations and the memory accesses are coalesced. Successive iterations fetch the tiles along a row which are in contiguous memory locations. Similarly the vertical differences of a tile below are accessed in one coalesced memory access by the threads of the block.

6 Empirical Results

Prior to empirical evaluation, the FG algorithm was expected to be the slowest due to redundant work. The memoized versions were expected to be faster than the two-vector algorithm, as they preprocess only a subset of the 2^{2q} vectors seen in the table.

We ran the programs on complete mouse non-coding RNA sequences. We also tested the performance on random substrings on real RNA sequences and random strings over A,C,G,U.

The FG algorithm, while faster than Nussinov, was the slowest among the Four-Russians methods, as expected. The completely memoized version was slower than the other two variants. This is because every lookup of the preprocessing table includes a check to see if the pair of vectors has already been processed. There are 2^{2q} unique vector pairs but there are $O(\frac{n^3}{q})$ queries to the preprocessing table and each query involves checking if the vector pair has been processed plus the processing time for new pairs. There are $O(\frac{n^2}{q^2})$ vector pairs in the table. For larger n (eg., $n > 1000$ and $q = 8$), all the 2^{2q} vectors are expected to be present in the DP table. Generally, memoized subproblems are relatively expensive compared to the lookup. Since the preprocessing here has only q steps, the advantage of memoization is not seen.

The partially memoized version was slightly slower than the two vector algorithm. Again, the advantage of potentially less preprocessing than the two-vector method is erased by the need to check if a vector has been processed. The two-vector method was the fastest on all sequence lengths tested.

For short sequences the two vector method took negligible time (less than 0.2 seconds up to 1000 bases) and are not reported. For longer sequences, we noticed that using longer vector lengths reduced the running time and the improvement saturated beyond $q = 8$ or 9 . Beyond this, the extra work in preprocessing overshadowed the benefit. A similar trend was seen for the memoized versions too. However, for the FG method $q = 3$ gave the best speedup and longer vector lengths had a slower running time due to the extra preprocessing at every group.

All the programs were written in C++ compiled with the highest compiler optimizations. We only discuss the experimental results on a desktop and two GPU cards in this paper. Detailed notes on running times can be found in [26].

Speedup factors of the serial programs on the desktop

Length	Time	Speedup			
	(in secs)	Two-vector	Partially Memoized	Completely Memoized	FG
2000	16.5	7.7	7.3	5.6	3.0
3000	62.5	8.8	8.3	6.4	3.4
4000	196.6	11.9	11.4	8.8	4.7
5000	630.3	21.1	18.9	14.7	7.8
6150	1027.8	18.1	17.0	13.3	7.03

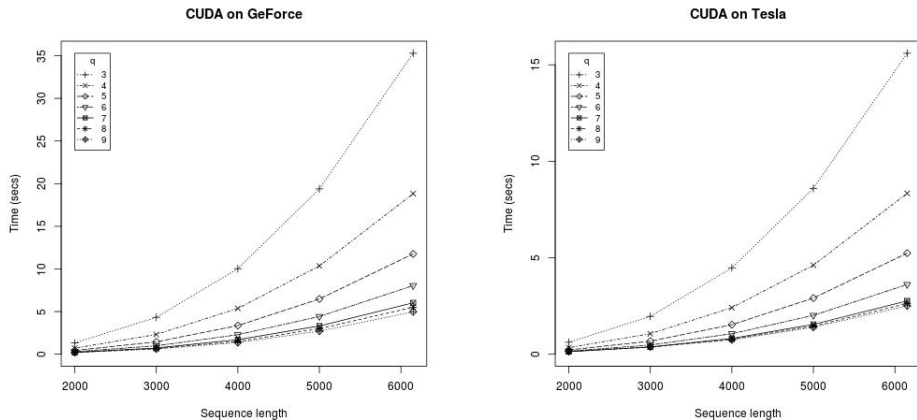


Fig. 2. Running time of the CUDA program on two GPUs. The programs run twice as fast on the Tesla card than the GeForce card.

We measured the running times of the different versions of our serial algorithms on a desktop machine with a Pentium II 3GHz processor and 1MB cache. The running times of Nussinov and the speedups of various programs compared to Nussinov are shown in the table below. For sequences of length 6000, the two-vector method takes close to a minute on the desktop.

Fig. 2 shows the execution times on two GPU cards – GeForce GTX 550 Ti card with 1GB on-card memory and Tesla C2070 with 5GB memory. The programs take about a second for sequences up to 4000 bases long, and takes about 5 seconds and 2.5 seconds for sequences of length 6000. The running times for various sequence lengths are shown in the table below.

Running times for the parallel program (in secs)		
Length	On GeForce	On Tesla
2000	0.20	0.14
3000	0.62	0.38
4000	1.36	0.74
5000	2.70	1.39
6000	4.97	2.50

7 Conclusions and Future Work

We described the two-vector method for using the Four-Russians technique for RNA folding. This method is simpler than the Frid-Gusfield method. It also improves the bound of the parallel algorithm by a $\log n$ factor to $O(\frac{n^2}{\log n})$. We

showed two other variants that memoize the preprocessing results. These methods are faster than Nussinov by up to a factor of 20 and the Frid-Gusfield method by a factor of 3.

In the future, it will be interesting to see the application of the Four-Russians technique for other methods that use energy models with thermodynamic parameters. The Frid-Gusfield method has been applied to RNA co-folding [12] and folding with pseudoknots [13] problems; the application of the two-vector method to those problems and its implications are also of interest. It will be interesting to compare our run time with the other improvements over Nussinov, like the boolean matrix multiplication method [1].

Acknowledgements. The first-listed author thanks Prof. Norm Matloff for the opportunity to lecture in his class; this project spawned out of that lecture. Thanks also to Prof. John Owens for access to the server with a Tesla card. Thanks to Jim Moersfelder and Vann Teves from Systems Support Staff for help in setting up the CUDA systems.

References

1. Akutsu, T.: Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages. *J. Comb. Optim.* 3(2-3), 321–336 (1999)
2. Andronescu, M., Condon, A., Hoos, H.H., Mathews, D.H., Murphy, K.P.: Efficient parameter estimation for RNA secondary structure prediction. *Bioinformatics* 23(13), i19–i28 (2007)
3. Andronescu, M., Condon, A., Hoos, H.H., Mathews, D.H., Murphy, K.P.: Computational approaches for RNA energy parameter estimation. *RNA* 16(12), 2304–2318 (2010)
4. Arlazarov, V., Dinic, E., Kronrod, M., Faradzev, I.: On economical construction of the transitive closure of a directed graph (in Russian). *Dokl. Akad. Nauk.* 194(11) (1970)
5. Backofen, R., Tsur, D., Zakov, S., Ziv-Ukelson, M.: Sparse RNA folding: Time and space efficient algorithms. *Journal of Discrete Algorithms* 9(1), 12–31 (2011)
6. Chang, D.-J., Kimmer, C., Ouyang, M.: Accelerating the nussinov RNA folding algorithm with CUDA/GPU. In: *ISSPIT*, pp. 120–125. IEEE (2010)
7. Ding, Y., Lawrence, C.E.: A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Research* 31(24), 7280–7301 (2003)
8. Do, C.B., Woods, D.A., Batzoglou, S.: CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics* 22(14), e90–e98 (2006)
9. Dowell, R., Eddy, S.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics* 5(1), 71 (2004)
10. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press (1998)
11. Frid, Y., Gusfield, D.: A simple, practical and complete $O(n^3)$ -time algorithm for RNA folding using the Four-Russians speedup. *Algorithms for Molecular Biology* 5, 13 (2010)

12. Frid, Y., Gusfield, D.: A worst-case and practical speedup for the RNA co-folding problem using the *four-russians* idea. In: Moulton, V., Singh, M. (eds.) WABI 2010. LNCS, vol. 6293, pp. 1–12. Springer, Heidelberg (2010)
13. Frid, Y., Gusfield, D.: Speedup of RNA pseudoknotted secondary structure recurrence computation with the four-russians method. In: Lin, G. (ed.) COCOA 2012. LNCS, vol. 7402, pp. 176–187. Springer, Heidelberg (2012)
14. Hamada, M., Kiryu, H., Sato, K., Mituyama, T., Asai, K.: Prediction of RNA secondary structure using generalized centroid estimators. *Bioinformatics* 25(4), 465–473 (2009)
15. Hofacker, I.L.: Vienna RNA secondary structure server. *Nucleic Acids Research* 31(13), 3429–3431 (2003)
16. Knudsen, B., Hein, J.: Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research* 31(13), 3423–3428 (2003)
17. Lu, Z.J., Gloor, J.W., Mathews, D.H.: Improved RNA secondary structure prediction by maximizing expected pair accuracy. *RNA* 15(10), 1805–1813 (2009)
18. Markham, N.R., Zuker, M.: UNAFold. *Bioinformatics* 453, 3–31 (2008)
19. Mathews, D.H., Disney, M.D., Childs, J.L., Schroeder, S.J., Zuker, M., Turner, D.H.: Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *PNAS* 101(19), 7287–7292 (2004)
20. Nussinov, R., Jacobson, A.B.: Fast algorithm for predicting the secondary structure of single-stranded RNA. *PNAS* 77(11), 6309–6313 (1980)
21. Nussinov, R., Pieczenik, G., Griggs, J.R., Kleitman, D.J.: Algorithms for loop matchings. *SIAM Journal on Applied Mathematics* 35(1), 68–82 (1978)
22. Reuter, J., Mathews, D.: RNAstructure: software for RNA secondary structure prediction and analysis. *BMC Bioinformatics* 11(1), 129 (2010)
23. Rizk, G., Lavenier, D.: GPU accelerated RNA folding algorithm. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009, Part I. LNCS, vol. 5544, pp. 1004–1013. Springer, Heidelberg (2009)
24. Stojanovski, M.Z., Gjorgjevikj, D., Madjarov, G.: Parallelization of dynamic programming in nussinov RNA folding algorithm on the CUDA GPU. In: Kocarev, L. (ed.) ICT Innovations 2011. AISC, vol. 150, pp. 279–289. Springer, Heidelberg (2012)
25. Tinoco, I., et al.: Improved Estimation Of Secondary Structure In Ribonucleic-Acids. *Nature-New Biology* 246(150), 40–41 (1973)
26. Venkatachalam, B., Frid, Y., Gusfield, D.: Faster algorithms for RNA-folding using the Four-Russians method. UC Davis Technical report (2013)
27. Wexler, Y., Zilberstein, C.B.-Z., Ziv-Ukelson, M.: A study of accessible motifs and RNA folding complexity. *Journal of Computational Biology* 14(6), 856–872 (2007)
28. Zakov, S., Goldberg, Y., Elhadad, M., Ziv-Ukelson, M.: Rich parameterization improves RNA structure prediction. In: Bafna, V., Sahinalp, S.C. (eds.) RECOMB 2011. LNCS, vol. 6577, pp. 546–562. Springer, Heidelberg (2011)
29. Zakov, S., Tsur, D., Ziv-Ukelson, M.: Reducing the worst case running times of a family of RNA and CFG problems, using valiant’s approach. In: Moulton, V., Singh, M. (eds.) WABI 2010. LNCS, vol. 6293, pp. 65–77. Springer, Heidelberg (2010)
30. Zuker, M.: Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Research* 31(13), 3406–3415 (2003)
31. Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research* 9(1), 133–148 (1981)

Algorithms for the Majority Rule (+) Consensus Tree and the Frequency Difference Consensus Tree

Jesper Jansson^{1,*}, Chuanqi Shen², and Wing-Kin Sung^{3,4}

¹ Laboratory of Mathematical Bioinformatics (Akutsu Laboratory),
Institute for Chemical Research,
Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan
jj@kuicr.kyoto-u.ac.jp

² Stanford University, 450 Serra Mall, Stanford, CA 94305-2004, USA
shencq@stanford.edu

³ School of Computing, National University of Singapore, 13 Computing Drive,
Singapore 117417
ksung@comp.nus.edu.sg

⁴ Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672

Abstract. This paper presents two new deterministic algorithms for constructing consensus trees. Given an input of k phylogenetic trees with identical leaf label sets and n leaves each, the first algorithm constructs the *majority rule (+) consensus tree* in $O(kn)$ time, which is optimal since the input size is $\Omega(kn)$, and the second one constructs the *frequency difference consensus tree* in $\min\{O(kn^2), O(kn(k + \log^2 n))\}$ time.

1 Introduction

A *consensus tree* is a phylogenetic tree that summarizes a given collection of phylogenetic trees having the same leaf labels but different branching structures. Consensus trees are used to resolve structural differences between two or more existing phylogenetic trees arising from conflicts in the raw data, to find strongly supported groupings, and to summarize large sets of candidate trees obtained by bootstrapping when trying to infer a new phylogenetic tree accurately [2, 10, 12, 27].

Since the first type of consensus tree was proposed by Adams III [1] in 1972, many others have been defined and analyzed. See, e.g., [5], Chapter 30 in [12], or Chapter 8.4 in [27] for some surveys. Which particular type of consensus tree to use in practice depends on the context. For example, the *strict consensus tree* [25] is very intuitive and easy to compute [9] and may be sufficient when there is not so much disagreement in the data, the *majority rule consensus tree* [21] is “the optimal tree to report if we view the cost of reporting an estimate of the phylogeny to be a linear function of the number of incorrect clades in the estimate and the number of true clades that are missing from the estimate and we

* Funded by The Hakubi Project and KAKENHI grant number 23700011.

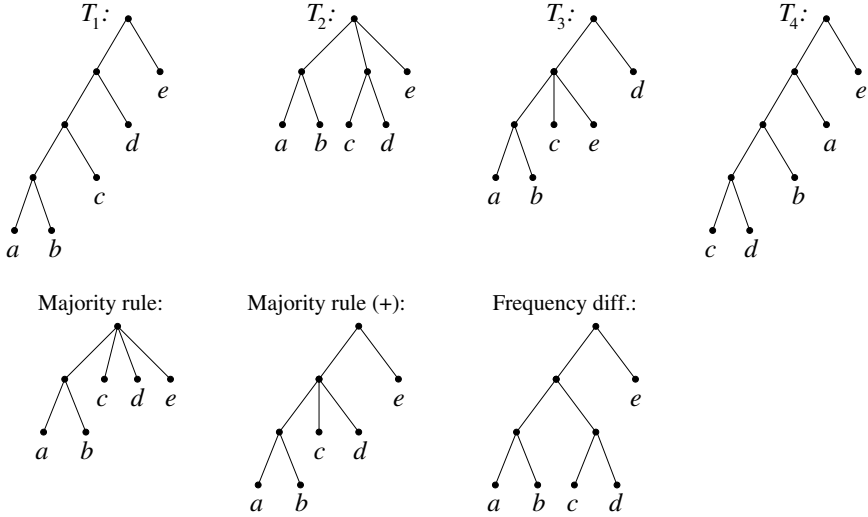


Fig. 1. Let $\mathcal{S} = \{T_1, T_2, T_3, T_4\}$ as shown above with $L = \Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \Lambda(T_4) = \{a, b, c, d, e\}$. The only non-trivial majority cluster of \mathcal{S} is $\{a, b\}$, the non-trivial majority (+) clusters of \mathcal{S} are $\{a, b\}$ and $\{a, b, c, d\}$, and the non-trivial frequency difference clusters of \mathcal{S} are $\{a, b\}$, $\{a, b, c, d\}$, and $\{c, d\}$. The majority rule, majority rule (+), and frequency difference consensus trees of \mathcal{S} are displayed.

view the reporting of an incorrect grouping as a more serious error than missing a clade” [16], and the R^* consensus tree [5] provides a statistically consistent estimator of the species tree topology when combining gene trees [10]. Therefore, scientists need efficient algorithms for constructing a broad range of different consensus trees.

In a recent series of papers [8, 17–19], we have developed fast algorithms for computing the *majority rule consensus tree* [21], the *loose consensus tree* [4] (also known in the literature as the *combinable component consensus tree* or the *semi-strict consensus tree*), a *greedy consensus tree* [5, 13], the R^* consensus tree [5], and consensus trees for so-called *multi-labeled phylogenetic trees (MUL-trees)* [20]. In this paper, we study two relatively new types of consensus trees called the *majority rule (+) consensus tree* [7, 11] and the *frequency difference consensus tree* [14], and give algorithms for constructing them efficiently.

1.1 Definitions and Notation

We shall use the following basic definitions. A *phylogenetic tree* is a rooted, unordered, leaf-labeled tree in which every internal node has at least two children and all leaves have different labels. (Below, phylogenetic trees are referred to as “trees” for short). For any tree T , the set of all nodes in T is denoted by $V(T)$ and the set of all leaf labels in T by $\Lambda(T)$. Any nonempty subset C of $\Lambda(T)$ is called a *cluster* of $\Lambda(T)$; if $|C| = 1$ or $C = \Lambda(T)$ then C is *trivial*, and otherwise,

C is *non-trivial*. For any $u \in V(T)$, $T[u]$ denotes the subtree of T rooted at the node u . Observe that if u is the root of T or if u is a leaf then $\Lambda(T[u])$ is a trivial cluster. The set $\mathcal{C}(T) = \bigcup_{u \in V(T)} \{\Lambda(T[u])\}$ is called the *cluster collection of T* , and any cluster $C \subseteq \Lambda(T)$ is said to *occur in T* if $C \in \mathcal{C}(T)$.

Two clusters $C_1, C_2 \subseteq \Lambda(T)$ are *compatible* if $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \cap C_2 = \emptyset$. If C_1 and C_2 are compatible, we write $C_1 \smile C_2$; otherwise, $C_1 \not\smile C_2$. A cluster $C \subseteq \Lambda(T)$ is *compatible with T* if $C \smile \Lambda(T[u])$ holds for every node $u \in V(T)$. In this case, we write $C \smile T$, and $C \not\smile T$ otherwise. If T_1 and T_2 are two trees with $\Lambda(T_1) = \Lambda(T_2)$ such that every cluster in $\mathcal{C}(T_1)$ is compatible with T_2 then it follows that every cluster in $\mathcal{C}(T_2)$ is compatible with T_1 , and we say that T_1 and T_2 are *compatible*. Any two clusters or trees that are not compatible are called *incompatible*.

Next, let $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ be a set of trees satisfying $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$ for some leaf label set L . For any cluster C of L , denote the set of all trees in \mathcal{S} in which C occurs by $K_C(\mathcal{S})$ and the set of all trees in \mathcal{S} that are incompatible with C by $Q_C(\mathcal{S})$. Thus, $K_C(\mathcal{S}) = \{T_i : C \in \mathcal{C}(T_i)\}$ and $Q_C(\mathcal{S}) = \{T_i : C \not\smile T_i\}$. Define three special types of clusters:

- If $|K_C(\mathcal{S})| > \frac{k}{2}$ then C is a *majority cluster of \mathcal{S}* .
- If $|K_C(\mathcal{S})| > |Q_C(\mathcal{S})|$ then C is a *majority (+) cluster of \mathcal{S}* .
- If $|K_C(\mathcal{S})| > \max\{|K_D(\mathcal{S})| : D \subseteq L \text{ and } C \not\smile D\}$ then C is a *frequency difference cluster of \mathcal{S}* .

(Informally, a frequency difference cluster is a cluster that occurs more frequently than each of the clusters that is incompatible with it.) Note that a majority cluster of \mathcal{S} is always a majority (+) cluster of \mathcal{S} and that a majority (+) cluster of \mathcal{S} is always a frequency difference cluster of \mathcal{S} , but not the other way around.

The *majority rule consensus tree of \mathcal{S}* [21] is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all majority clusters of \mathcal{S} . Similarly, the *majority rule (+) consensus tree of \mathcal{S}* [7, 11] is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all majority (+) clusters of \mathcal{S} , and the *frequency difference consensus tree of \mathcal{S}* [14] is the tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all frequency difference clusters of \mathcal{S} . See Fig. 1 for some examples.

From here on, \mathcal{S} is assumed to be an input set of identically leaf-labeled trees, and the leaf label set of \mathcal{S} is denoted by L . To express the size of the input, we define $k = |\mathcal{S}|$ and $n = |L|$.

1.2 Previous Work

Margush and McMorris [21] introduced the majority rule consensus tree in 1981, and a deterministic algorithm for constructing it in optimal $O(kn)$ worst-case running time was presented recently in [18]. (A randomized algorithm with $O(kn)$ expected running time and unbounded worst-case running time was given earlier by Amenta *et al.* [2].) The majority rule consensus tree has several desirable mathematical properties [3, 16, 22], and algorithms for constructing it

have been implemented in popular computational phylogenetics packages like PHYLIP [13], TNT [15], COMPONENT [23], MrBayes [24], SumTrees in DendroPy [26], and PAUP* [28]. Consequently, it is one of the most widely used consensus trees in practice [7, p. 450]. One drawback of the majority rule consensus tree is that it may be too harsh and discard valuable branching information. For example, in Fig. 1, even though the cluster $\{a, b, c, d\}$ is compatible with 75% of the input trees, it is not included in the majority rule consensus tree. For this reason, people have become interested in alternative types of consensus trees that include all the majority clusters and at the same time, also include other meaningful, well-defined kinds of clusters. The majority rule (+) consensus tree and the frequency difference consensus tree are two such consensus trees.

The majority rule (+) consensus tree was defined by Dong *et al.* [11] in 2010. It was obtained as a special case of an attempted generalization by Cotton and Wilkinson [7] of the majority rule consensus tree. According to [11], Cotton and Wilkinson [7] suggested two types of supertrees¹ called majority-rule (-) and majority-rule (+) that were supposed to generalize the majority rule consensus tree. Unexpectedly, only the first one did, and by restricting the second one to the consensus tree case, [11] arrived at the majority rule (+) consensus tree. Dong *et al.* [11] established some fundamental properties of this type of consensus tree and pointed out the existence of a polynomial-time algorithm for constructing it, but left the task of finding the best possible such algorithm as an open problem. As far as we know, no implementation for computing the majority rule (+) consensus tree is publicly available.

Goloboff *et al.* [14] initially proposed the frequency difference consensus tree as a way to improve methods for evaluating group support in parsimony analysis. Its relationships to other consensus trees have been studied in [11]. A method for constructing it has been implemented in the free software package TNT [15] but the algorithm used is not documented and its time complexity is unknown. We note that since the number of clusters occurring in \mathcal{S} may be $\Omega(kn)$, a naive algorithm that compares every cluster in \mathcal{S} to every other cluster in \mathcal{S} directly would require $\Omega(k^2n^2)$ time.

1.3 Organization of the Paper and New Results

Due to space constraints, some proofs have been omitted from the conference version of this paper. Please see the journal version for the complete proofs.

The paper is organized as follows. Section 2 summarizes some results from the literature that are needed later. In Section 3, we modify the techniques from [18] to obtain an $O(kn)$ -time algorithm for the majority rule (+) consensus tree. Its running time is optimal because the size of the input is $\Omega(kn)$; hence, we resolve the open problem of Dong *et al.* [11] mentioned above. Next, Section 4 gives a $\min\{O(kn^2), O(kn(k + \log^2 n))\}$ -time algorithm for constructing the frequency difference consensus tree (here, the second term is smaller than the first term

¹ A *supertree* is a generalization of a consensus tree that does not require the input trees to have identical leaf label sets.

if $k = o(n)$; e.g., if $k = O(1)$ then the running time reduces to $O(n \log^2 n)$. Our algorithms are fully deterministic and do not need to use hashing. Finally, Section 5 discusses implementations.

2 Preliminaries

2.1 The *delete* and *insert* Operations

The *delete* and *insert* operations are two operations that modify the structure of a tree. They are defined in the following way.

Let T be a tree and let u be any non-root, internal node in T . The *delete* operation on u makes all of u 's children become children of the parent of u , and then removes u and the edge between u and its parent. (See, e.g., Figure 2 in [17] for an illustration.) The time needed for this operation is proportional to the number of children of u , and the effect of applying it is that the cluster collection of T is changed to $\mathcal{C}(T) \setminus \{\Lambda(T[u])\}$. Conversely, the *insert* operation creates a new node u that becomes: (1) a child of an existing internal node v , and (2) the parent of a proper subset X of v 's children satisfying $|X| \geq 2$; the effect is that $\mathcal{C}(T)$ is changed to $\mathcal{C}(T) \cup \{\Lambda(T[u])\}$, where $\Lambda(T[u]) = \bigcup_{v_i \in X} \Lambda(T[v_i])$.

2.2 Subroutines

The new algorithms in this paper use the following algorithms from the literature as subroutines: Day's algorithm [9], Procedure `One-Way-Compatible` [17], and Procedure `Merge-Trees` [17]. Day's algorithm [9] is used to efficiently check whether any specified cluster that occurs in a tree T also occurs in another tree T_{ref} , and can be applied to find the set of all clusters that occur in both T and T_{ref} in linear time. Procedure `One-Way-Compatible` takes as input two trees T_A and T_B with identical leaf label sets and outputs a copy of T_A in which every cluster that is not compatible with T_B has been removed. (The procedure is asymmetric; e.g., if T_A consists of n leaves attached to a root node and $T_B \neq T_A$ then `One-Way-Compatible`(T_A, T_B) = T_A , while `One-Way-Compatible`(T_B, T_A) = T_B .) Procedure `Merge-Trees` takes as input two compatible trees with identical leaf label sets and outputs a tree that combines their cluster collections. Their properties are summarized below; for details, see references [9] and [17].

Lemma 1. (*Day [9]*) Let T_{ref} and T be two given trees with $\Lambda(T_{ref}) = \Lambda(T) = L$ and let $n = |L|$. After $O(n)$ time preprocessing, it is possible to determine, for any $u \in V(T)$, if $\Lambda(T[u]) \in \mathcal{C}(T_{ref})$ in $O(1)$ time.

Lemma 2. (*[17]*) Let T_A and T_B be two given trees with $\Lambda(T_A) = \Lambda(T_B) = L$ and let $n = |L|$. Procedure `One-Way-Compatible`(T_A, T_B) returns a tree T with $\Lambda(T) = L$ such that $\mathcal{C}(T) = \{C \in \mathcal{C}(T_A) : C \text{ is compatible with } T_B\}$ in $O(n)$ time.

Lemma 3. (*[17]*) Let T_A and T_B be two given trees with $\Lambda(T_A) = \Lambda(T_B) = L$ that are compatible and let $n = |L|$. Procedure `Merge-Trees`(T_A, T_B) returns a tree T with $\Lambda(T) = L$ and $\mathcal{C}(T) = \mathcal{C}(T_A) \cup \mathcal{C}(T_B)$ in $O(n)$ time.

3 Constructing the Majority Rule (+) Consensus Tree

This section presents an algorithm named `Maj_Rule_Plus` for computing the majority rule (+) consensus tree of \mathcal{S} in (optimal) $O(kn)$ time.

The pseudocode of `Maj_Rule_Plus` is given in Fig. 2. The algorithm has two phases. Phase 1 examines the input trees, one by one, to construct a set of candidate clusters that includes all majority (+) clusters. Then, Phase 2 removes all candidate clusters that are not majority (+) clusters.²

During Phase 1, the current candidate clusters are stored as nodes in a tree T . Every node v in T represents a current candidate cluster $\Lambda(T[v])$ and has a counter $count(v)$ that, starting from the iteration at which $\Lambda(T[v])$ became a candidate cluster, keeps track of the number of input trees in which it occurs minus the number of input trees that are incompatible with it. More precisely, while treating the tree T_j for any $j \in \{2, 3, \dots, k\}$ in Step 3.1, $count(v)$ for each current candidate cluster $\Lambda(T[v])$ is updated as follows: if $\Lambda(T[v])$ occurs in T_j then $count(v)$ is incremented by 1, if $\Lambda(T[v])$ does not occur in T_j and is not compatible with T_j then $count(v)$ is decremented by 1, and otherwise (i.e., $\Lambda(T[v])$ does not occur in T_j but is compatible with T_j) $count(v)$ is unchanged. Furthermore, if any $count(v)$ reaches 0 then the node v is deleted from T so that $\Lambda(T[v])$ is no longer a current candidate cluster. Next, in Step 3.3, every cluster occurring in T_j that is not a current candidate but compatible with T is inserted into T (thus becoming a current candidate cluster) and its counter is initialized to 1. Lemma 4 below proves that the set of majority (+) clusters of \mathcal{S} is contained in the set of candidate clusters at the end of Phase 1.

Lemma 4. *For any $C \subseteq L$, if C is a majority (+) cluster of \mathcal{S} then $C \in \mathcal{C}(T)$ at the end of Phase 1.*

Proof. Suppose that C is a majority (+) cluster of \mathcal{S} . Let T_x be any tree in $Q_C(\mathcal{S})$ and consider iteration x in Step 3: If C is a current candidate at the beginning of iteration x then its counter will be decremented, cancelling out the occurrence of C in one tree T_j where $1 \leq j < x$; otherwise, C may be prevented from being inserted into T in at most one later iteration j (where $x < j \leq k$ and $C \in \mathcal{C}(T_j)$) because of some cluster occurring in T_x . It follows from $|K_C(\mathcal{S})| - |Q_C(\mathcal{S})| > 0$ that C 's counter will be greater than 0 at the end of Phase 1, and therefore $C \in \mathcal{C}(T)$. \square

In Phase 2, Step 5 of the algorithm computes the values of $|K_C(\mathcal{S})|$ and $|Q_C(\mathcal{S})|$ for every candidate cluster C and stores them in $K(v)$ and $Q(v)$, respectively, where $C = \Lambda(T[v])$. Finally, Step 6 removes every candidate cluster C that does not satisfy the condition $|K_C(\mathcal{S})| > |Q_C(\mathcal{S})|$. By definition, the clusters that remain in T are the majority rule (+) clusters.

Theorem 1. *Algorithm `Maj_Rule_Plus` constructs the majority rule (+) consensus tree of \mathcal{S} in $O(kn)$ time.*

² This basic strategy was previously used in the $O(kn)$ -time algorithm in [18] for computing the majority rule consensus tree.

```

Algorithm  Maj_Rule_Plus
Input:    A set  $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$  of trees with  $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k)$ .
Output:  The majority rule (+) consensus tree of  $\mathcal{S}$ .

    /* Phase 1 */
  1   $T := T_1$ 
  2  for each  $v \in V(T)$  do  $count(v) := 1$ 
  3  for  $j := 2$  to  $k$  do
  3.1 for each  $v \in V(T)$  do
        if  $\Lambda(T[v])$  occurs in  $T_j$  then  $count(v) := count(v) + 1$ 
        else if  $\Lambda(T[v])$  is not compatible with  $T_j$  then  $count(v) := count(v) - 1$ 
      endfor
  3.2 for each  $v \in V(T)$  in top-down order do
        if  $count(v) = 0$  then delete node  $v$ .
      endfor
  3.3 for every  $C \in \mathcal{C}(T_j)$  that is compatible with  $T$  but does not occur in  $T$  do
        Insert  $C$  into  $T$ .
        Initialize  $count(v) := 1$  for the new node  $v$  satisfying  $\Lambda(T[v]) = C$ .
      endfor
  endfor

    /* Phase 2 */
  4  for each  $v \in V(T)$  do  $K(v) := 0$ ;  $Q(v) := 0$ 
  5  for  $j := 1$  to  $k$  do
  5.1 for each  $v \in V(T)$  do
        if  $\Lambda(T[v])$  occurs in  $T_j$  then  $K(v) := K(v) + 1$ 
        else if  $\Lambda(T[v])$  is not compatible with  $T_j$  then  $Q(v) := Q(v) + 1$ 
      endfor
  6  for each  $v \in V(T)$  in top-down order do
        if  $K(v) \leq Q(v)$  then perform a delete operation on  $v$ .
      endfor
  7  return  $T$ 
End Maj_Rule_Plus

```

Fig. 2. Algorithm Maj_Rule_Plus for constructing the majority rule (+) consensus tree

Proof. The correctness follows from Lemma 4 and the above discussion.

The time complexity analysis is analogous to the proof of Theorem 4 in [18]. First consider Phase 1. Step 3.1 takes $O(n)$ time by: (1) running Day’s algorithm with $T_{ref} = T_j$ and then checking each node v in $V(T)$ to see if $\Lambda(T[v])$ occurs in T_j (according to Lemma 1, this requires $O(n)$ time for preprocessing, and each of the $O(n)$ nodes in $V(T)$ may be checked in $O(1)$ time), and (2) computing $X := \text{One-Way-Compatible}(T, T_j)$ and then checking for each node v in $V(T)$ if v does not exist in X to determine if $\Lambda(T[v]) \not\sim T_j$ (this takes $O(n)$ time by Lemma 2). The *delete* operations in Step 3.2 take $O(n)$ time because the nodes are handled in top-down order, which means that for every node, its parent will change at most once in each iteration. In Step 3.3, define $Y := \text{One-Way-Compatible}(T_j, T)$ and $Z := \text{Merge-Trees}(Y, T)$. Then by Lemmas 2

and 3, the cluster collection of Y consists of the clusters occurring in T_j that are compatible with the set of current candidates, and Z is the result of inserting these clusters into T . Thus, Step 3.3 can be implemented by computing Y and Z , updating T 's structure according to Z , and setting the counters of all new nodes to 1, so Step 3.3 takes $O(n)$ time. The main loop in Step 3 consists of $O(k)$ iterations, and Phase 1 therefore takes $O(kn)$ time in total.

Next, Phase 2 also takes $O(kn)$ time because Step 5.1 can be implemented in $O(n)$ time with the same techniques as in Step 3.1, and Step 6 is performed in $O(n)$ time by handling the nodes in top-down order so that each node's parent is changed at most once, as in Step 3.2. \square

4 Constructing the Frequency Difference Consensus Tree

Here, we present an algorithm for finding the frequency consensus tree of \mathcal{S} in $\min\{O(kn^2), O(kn(k + \log^2 n))\}$ time. It is called `FrequencyDifference` and is described in Section 4.1 below. The algorithm uses the procedure `Merge_Trees` as well as a new procedure named `Filter_Clusters` whose details are given in Section 4.2.

For each tree $T_j \in \mathcal{S}$ and each node $u \in V(T_j)$, define the *weight of u* as the value $|K_{\Lambda(T_j[u])}(\mathcal{S})|$, i.e., the number of trees from \mathcal{S} in which the cluster $\Lambda(T_j[u])$ occurs, and denote it by $w(u)$. For convenience, also define $w(C) = w(u)$, where $C = \Lambda(T_j[u])$. The input to Procedure `Filter_Clusters` is two trees T_A, T_B with $\Lambda(T_A) = \Lambda(T_B) = L$ such that every cluster occurring in T_A or T_B also occurs in at least one tree in \mathcal{S} , and the output is a copy of T_A in which every cluster that is incompatible with some cluster in T_B with a higher weight has been removed. Formally, the output of `Filter_Clusters` is a tree T with $\Lambda(T) = L$ such that $\mathcal{C}(T) = \{\Lambda(T_A[u]) : u \in V(T_A) \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\prec \Lambda(T_B[x])\}$.

4.1 Algorithm `FrequencyDifference`

We first describe Algorithm `FrequencyDifference`. Refer to Fig. 3 for the pseudocode.

The algorithm starts by computing the weight $w(C)$ of every cluster C occurring in \mathcal{S} in a preprocessing step (Step 1). Next, let $\mathcal{C}(S)$ for any set S of trees denote the union $\bigcup_{T_i \in S} \mathcal{C}(T_i)$, and for any $j \in \{1, 2, \dots, k\}$, define a *forward frequency difference consensus tree of $\{T_1, T_2, \dots, T_j\}$* as any tree that includes every cluster C in $\mathcal{C}(\{T_1, T_2, \dots, T_j\})$ satisfying $w(C) > w(X)$ for all $X \in \mathcal{C}(\{T_1, T_2, \dots, T_j\})$ with $C \not\prec X$. Steps 2–3 use Procedure `Filter_Clusters` from Section 4.2 to build a tree T that, after any iteration $j \in \{1, 2, \dots, k\}$, is a forward frequency difference consensus tree of $\{T_1, T_2, \dots, T_j\}$, as proved in Lemma 5 below. After iteration k , $\mathcal{C}(T)$ contains all frequency difference clusters of \mathcal{S} but possibly some other clusters as well, so Step 4 applies `Filter_Clusters` again to remove all non-frequency difference clusters of \mathcal{S} from T .

```

Algorithm   Frequency_Difference
Input:     A set  $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$  of trees with  $A(T_1) = A(T_2) = \dots = A(T_k)$ .
Output:   The frequency difference consensus tree of  $\mathcal{S}$ .

    /* Preprocessing */
    1 Compute  $w(C)$  for every cluster  $C$  occurring in  $\mathcal{S}$ .

    /* Main algorithm */
    2  $T := T_1$ 
    3 for  $j := 2$  to  $k$  do
         $A := \text{Filter\_Clusters}(T, T_j)$ ;  $B := \text{Filter\_Clusters}(T_j, T)$ 
         $T := \text{Merge\_Trees}(A, B)$ 
    endfor
    4 for  $j := 1$  to  $k$  do  $T := \text{Filter\_Clusters}(T, T_j)$ 
    5 return  $T$ 
End   Frequency_Difference
    
```

Fig. 3. Algorithm Frequency_Difference for constructing the frequency difference consensus tree

Lemma 5. For any $j \in \{2, 3, \dots, k\}$, suppose that T is a forward frequency difference consensus tree of $\{T_1, T_2, \dots, T_{j-1}\}$. Let $A := \text{Filter_Clusters}(T, T_j)$ and $B := \text{Filter_Clusters}(T_j, T)$. Then $\text{Merge_Trees}(A, B)$ is a forward frequency difference consensus tree of $\{T_1, T_2, \dots, T_j\}$.

Proof. (Omitted from the conference version due to space constraints.) □

Theorem 2. Algorithm Frequency_Difference constructs the frequency difference consensus tree of \mathcal{S} in $\min\{O(kn^2), O(k^2n)\} + O(k \cdot f(n))$ time, where $f(n)$ is the running time of Procedure Filter_Clusters.

Proof. After completing iteration k of Step 3, $\mathcal{C}(T)$ is a superset of the set of all frequency difference clusters of \mathcal{S} by Lemma 5. Next, Step 4 removes all non-frequency difference clusters of \mathcal{S} , so the output will be the frequency difference consensus tree of \mathcal{S} .

To analyze the time complexity, first consider how to compute all the weights in Step 1. One method is to first fix an arbitrary ordering of L and represent every cluster C of L as a bit vector of length n (for every $i \in \{1, 2, \dots, n\}$, the i th bit is set to 1 if and only if the i th leaf label belongs to C). Then, spend $O(kn^2)$ time to construct a list of bit vectors for all $O(kn)$ clusters occurring in \mathcal{S} by a bottom-up traversal of each tree in \mathcal{S} , sort the resulting list of bit vectors by radix sort, and traverse the sorted list to identify the number of occurrences of each cluster. All this takes $O(kn^2)$ time. An alternative method, which uses $O(k^2n)$ time, is to initialize the weight of every node in \mathcal{S} to 1 and then, for $j \in \{1, 2, \dots, k\}$, apply Day’s algorithm (see Lemma 1) with $T_{ref} = T_j$ and T ranging over all T_i with $1 \leq i \leq k, i \neq j$ to find all clusters in T that also occur in T_j and increase the

weights of their nodes in T by 1. Therefore, Step 1 takes $\min\{O(kn^2), O(k^2n)\}$ time. Next, Steps 3 and 4 make $O(k)$ calls to the procedures `Merge_Trees` and `Filter_Clusters`. The running time of `Merge_Trees` is $O(n)$ by Lemma 3 and the running time of `Filter_Clusters` is $f(n) = \Omega(n)$, so Steps 3 and 4 take $O(k \cdot f(n))$ time. \square

Lemma 7 in the next subsection shows that $f(n) = O(n \log^2 n)$ is possible, which yields:

Corollary 1. *Algorithm `Frequency_Difference` constructs the frequency difference consensus tree of \mathcal{S} in $\min\{O(kn^2), O(kn(k + \log^2 n))\}$ time.*

4.2 Procedure `Filter_Clusters`

Recall that for any node u in any input tree T_j , its weight $w(u)$ is $|K_{\Lambda(T_j[u])}(\mathcal{S})|$. Also, $w(C) = w(u)$, where $C = \Lambda(T_j[u])$. We assume that all $w(u)$ -values have been computed in a preprocessing step and are available.

Let T be a tree. For every nonempty $X \subseteq V(T)$, $\text{lca}^T(X)$ denotes the lowest common ancestor of X in T . To obtain a fast solution for `Filter_Clusters`, we need the next lemma.

Lemma 6. *Let T be a tree, let X be any cluster of $\Lambda(T)$, and let $r_X = \text{lca}^T(X)$. For any $v \in V(T)$, it holds that $X \not\prec \Lambda(T[v])$ if and only if: (1) v lies on a path from a child of r_X to some leaf belonging to X ; and (2) $\Lambda(T[v]) \not\subseteq X$.*

Proof. Given T , X , r_X , and v as in the lemma statement, there are four possible cases: (i) v is a proper ancestor of r_X or equal to r_X ; (ii) v lies on a path from a child of r_X to some leaf in X and all leaf descendants of v belong to X ; (iii) v lies on a path from a child of r_X to some leaf in X and not all leaf descendants of v belong to X ; or (iv) v is a proper descendant of r_X that does not lie on any path from a leaf in X to r_X . In case (i), $X \subseteq \Lambda(T[v])$. In case (ii), $\Lambda(T[v]) \subseteq X$. In case (iii), $\Lambda(T[v]) \not\subseteq X$ while $X \cap \Lambda(T[v]) \neq \emptyset$. In case (iv), $X \cap \Lambda(T[v]) = \emptyset$. By the definition of compatible clusters, $X \not\prec \Lambda(T[v])$ if and only if case (iii) occurs. \square

Lemma 6 leads to an $O(n^2)$ -time method for `Filter_Clusters`, which we now briefly describe. For each node $u \in V(T_A)$ in top-down order, do the following: Let $X = \Lambda(T_A[u])$ and find all $v \in V(T_B)$ such that $X \not\prec \Lambda(T_B[v])$ in $O(n)$ time by doing bottom-up traversals of T_B to first mark all ancestors of leaves belonging to X that are proper descendants of the lowest common ancestor of X in T_B , and then unmarking all marked nodes that have no leaf descendants outside of X . By Lemma 6, $X \not\prec \Lambda(T_B[v])$ if and only if v is one of the resulting marked nodes. If $w(u) \leq w(v)$ for any such v then do a *delete* operation on u in T_A . Clearly, the total running time is $O(n^2)$. (This simple method gives $f(n) = O(n^2)$ in Theorem 2 in Section 4.1, and hence a total running time of $O(kn^2)$ for Algorithm `Frequency_Difference`.) Below, we refine this idea to get an even faster solution for `Filter_Clusters`.

High-Level Description. We use the *centroid path decomposition* technique [6] to divide the nodes of T_A into a so-called centroid path and a set of side trees. A *centroid path* of T_A is defined as a path in T_A of the form $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$, where p_α is the root of T_A , the node p_{i-1} for every $i \in \{2, \dots, \alpha\}$ is any child of p_i with the maximum number of leaf descendants, and p_1 is a leaf. Given a centroid path π , removing π and all its incident edges from T_A produces a set $\sigma(\pi)$ of disjoint trees whose root nodes are children of nodes belonging to π in T_A ; these trees are called the *side trees* of π . Importantly, $|\Lambda(\tau)| \leq n/2$ for every side tree τ of π . Also, $\{\Lambda(\tau) : \tau \in \sigma(\pi)\}$ forms a partition of $L \setminus \{p_1\}$. Furthermore, if π is a centroid path of T_A then the cluster collection $\mathcal{C}(T_A)$ can be written recursively as $\mathcal{C}(T_A) = \bigcup_{\tau \in \sigma(\pi)} \mathcal{C}(\tau) \cup \bigcup_{p_i \in \pi} \{\Lambda(T_A[p_i])\}$. Intuitively, this allows the cluster collection of T_A to be broken into smaller sets that can be checked more easily, and then put together again at the end.

The fast version of `Filter_Clusters` is shown in Fig. 4. It first computes a centroid path $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$ of T_A and the set $\sigma(\pi)$ of side trees of π in Step 1. Then, in Steps 2–3, it applies itself recursively to each side tree of π to get rid of any cluster in $\bigcup_{\tau \in \sigma(\pi)} \mathcal{C}(\tau)$ that is incompatible with some cluster in T_B with a higher weight than itself, and the remaining clusters are inserted into a temporary tree R_s . Next, Steps 4–5 check all clusters in $\bigcup_{p_i \in \pi} \{\Lambda(T_A[p_i])\}$ to determine which of them are not incompatible with any cluster in T_B with a higher weight, and create a temporary tree R_c whose cluster collection consists of all those clusters that pass this test. Finally, Step 6 combines the cluster collections of R_s and R_c by applying the procedure `Merge_Trees`. The details of Procedure `Filter_Clusters` are discussed next.

Steps 2–3 (Handling the Side Trees). For every nonempty $C \subseteq \Lambda(T)$, define $T|C$ (“the subtree of T induced by C ”; see, e.g., [6]) as the tree T' with leaf label set C and internal node set $\{lca^T(\{a, b\}) : a, b \in C\}$ which preserves the ancestor relations from T , i.e., which satisfies $lca^T(C') = lca^{T'}(C')$ for all nonempty $C' \subseteq C$. Now, let $\sigma(\pi)$ be the set of side trees of the centroid path π computed in Step 1. For each $\tau \in \sigma(\pi)$, define a weighted tree $T_B|\Lambda(\tau)$ as follows. First, construct $T_B|\Lambda(\tau)$ and let the weight of each node in this tree equal its weight in T_B . Next, for each edge (u, v) in $T_B|\Lambda(\tau)$, let P be the path in T_B between u and v , excluding u and v ; if P is not empty then create a new node z in $T_B|\Lambda(\tau)$, replace the edge (u, v) by the two edges (u, z) and (z, v) , and set the weight of z to the maximum weight of all nodes belonging to P . Each such z is called a *special node* and has exactly one child. See Fig. 5 for an example. We extend the concept of “compatible” to special nodes as follows: if $C \subseteq L$ and z is a special node in $T_B|\Lambda(\tau)$ then $C \smile z$ if and only if C and $\Lambda((T_B|\Lambda(\tau))[z])$ are disjoint or $(T_B|\Lambda(\tau))[z]$ has no proper descendant that is a special node. The obtained tree $T_B|\Lambda(\tau)$ satisfies $\Lambda(\tau) = \Lambda(T_B|\Lambda(\tau))$ and has the property that for every cluster C in $\mathcal{C}(\tau)$, $\max\{w(X) : X \in \mathcal{C}(T_B) \text{ and } C \not\smile X\}$ is equal to $\max\{w(X) : X \in \mathcal{C}(T_B|\Lambda(\tau)) \text{ and } C \not\smile X\}$.

After constructing $T_B|\Lambda(\tau)$, `Filter_Clusters` is applied to $(\tau, T_B|\Lambda(\tau))$ recursively to remove all bad clusters from τ . For each $\tau \in \sigma(\pi)$, the resulting tree is denoted by τ' . All the clusters of τ' are inserted into the tree R_s by

Algorithm Filter_Clusters

Input: Two trees T_A, T_B with $\Lambda(T_A) = \Lambda(T_B) = L$ such that every cluster occurring in T_A or T_B also occurs in at least one tree in \mathcal{S} .

Output: A tree T with $\Lambda(T) = L$ such that $\mathcal{C}(T) = \{\Lambda(T_A[u]) : u \in V(T_A) \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\prec \Lambda(T_B[x])\}$.

1 Compute a centroid path $\pi = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$ of T_A , where p_α is the root of T_A and p_1 is a leaf, and compute the set $\sigma(\pi)$ of side trees of π .

/* Handle the side trees. */

2 Let R_s be a tree consisting only of a root node and a single leaf labeled by p_1 .

3 **for** each side tree $\tau \in \sigma(\pi)$ **do**

$\tau' := \text{Filter_Clusters}(\tau, T_B | \Lambda(\tau))$

 Attach the root of τ' to the root of R_s .

endfor

/* Handle the centroid path. */

4 Let R_c be a tree with $\Lambda(R_c) = L$ where every leaf is directly attached to the root. Let BT be an empty binary search tree. For every $x \in V(T_B)$, initialize $counter(x) := 0$. Do a bottom-up traversal of T_B to precompute $|\Lambda(T_B[x])|$ for every $x \in V(T_B)$. Preprocess T_B for answering *lca*-queries. Let $\beta_1 := 0$.

5 **for** $i := 2$ **to** α **do**

5.1 Let D be the set of leaves in $\Lambda(T_A[p_i]) \setminus \Lambda(T_A[p_{i-1}])$.

5.2 Compute $r_i := \text{lca}^{T_B}(\{r_{i-1}\} \cup D)$. /* r_i now equals $\text{lca}^{T_B}(\Lambda(T_A[p_i])$. */

5.3 Insert every node belonging to the path from r_i to r_{i-1} , except r_i , into BT .

5.4 **for** each $x \in D$ **do**

 Insert x into BT .

while ($\text{parent}(x)$ is not in BT and $\text{parent}(x) \neq r_i$) **do**

$x := \text{parent}(x)$; insert x into BT .

endfor

5.5 **for** each $x \in D$ **do**

$counter(x) := counter(x) + 1$

while ($counter(x) = |\Lambda(T_B[x])|$) **do**

$counter(\text{parent}(x)) := counter(\text{parent}(x)) + |\Lambda(T_B[x])|$

 Remove x from BT ; $x := \text{parent}(x)$

endwhile

endfor

5.6 Let $M :=$ maximum weight of a node in BT ; **if** BT is empty **then** $M := 0$.

5.7 Compute β_i , and **if** $\beta_i > M$ **then** let $M := \beta_i$.

5.8 **if** ($w(\Lambda(T_A[p_i])) > M$) **then** put $\Lambda(T_A[p_i])$ in R_c by an *insert* operation.

endfor

/* Combine the surviving clusters. */

6 $T := \text{Merge_Trees}(R_s, R_c)$

7 **return** T

End Filter_Clusters

Fig. 4. The procedure Filter_Clusters

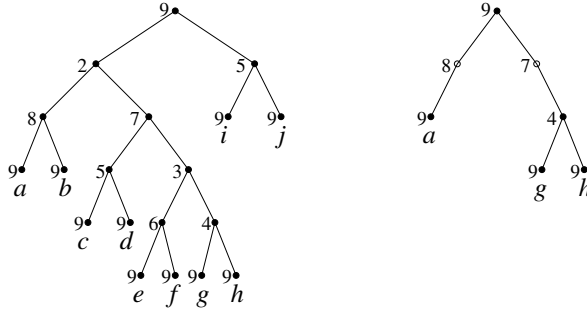


Fig. 5. Let T_B be the tree (with node weights) on the left. The tree $T_B||\{a, g, h\}$ is shown on the right.

directly attaching τ' to the root of R_s . Since $\{\Lambda(\tau') : \tau \in \sigma(\pi)\}$ forms a partition of $L \setminus \{p_1\}$, every leaf label in L appears exactly once in R_s and we have $\mathcal{C}(R_s) = \{\Lambda(T_A[u]) : u \in V(\tau) \text{ for some } \tau \in \sigma(\pi) \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\subseteq \Lambda(T_B[x])\} \cup \{L\}$ after Step 3 is finished.

Steps 4–5 (Handling the Centroid Path). The clusters $\bigcup_{p_i \in \pi} \{\Lambda(T_A[p_i])\}$ on the centroid path are nested because p_i is the parent of p_{i-1} , so $\Lambda(T_A[p_{i-1}]) \subseteq \Lambda(T_A[p_i])$ for every $i \in \{2, 3, \dots, \alpha\}$. The main loop (Step 5) checks each of these clusters in order of increasing cardinality.

The algorithm maintains a binary search tree BT that, right after Step 5.5 in any iteration i of the main loop is complete, contains every node x from T_B with $\Lambda(T_A[p_i]) \not\subseteq \Lambda(T_B[x])$. Whenever a node x is inserted into BT , its key is set to the weight $w(T_B[x])$. Using BT , Step 5.6 retrieves the weight M of the heaviest cluster in T_B that is incompatible with $\Lambda(T_A[p_i])$ (if any). Then, Step 5.7 computes a value β_i , defined as the maximum weight of all special nodes in T_B (if any) that are incompatible with the current $T_A[p_i]$; if $\beta_i > M$ then M is set to β_i . Step 5.8 saves $\Lambda(T_A[p_i])$ by inserting it into the tree R_c if its weight is strictly greater than M . After Step 5 is done, $\mathcal{C}(R_c) = \{\Lambda(T_A[u]) : u \in \pi \text{ and } w(u) > w(x) \text{ for every } x \in V(T_B) \text{ with } \Lambda(T_A[u]) \not\subseteq \Lambda(T_B[x])\}$.

In order to update BT correctly while moving upwards along π in Step 5, the algorithm relies on Lemma 6. In each iteration $i \in \{2, 3, \dots, \alpha\}$ of Step 5, r_i is the lowest common ancestor in T_B of $\Lambda(T_A[p_i])$. By Lemma 6, the clusters in T_B that are incompatible with $\Lambda(T_A[p_i])$ are of the form $T_B[v]$ where: (1) v lies on a path in T_B from a child of r_i to a leaf in $\Lambda(T_A[p_i])$; and (2) $\Lambda(T[v]) \not\subseteq \Lambda(T_A[p_i])$. Accordingly, BT is updated in Steps 5.3–5.5 as follows. Condition (1) is taken care of by first inserting all nodes from T_B between r_{i-1} and r_i except r_i into BT in Step 5.3 and then inserting all leaf descendants of p_i that are not descendants of p_{i-1} , along with any of their ancestors in T_B that were not already in BT , into BT in Step 5.4. Finally, Step 5.5 enforces condition (2) by using counters to locate and remove all nodes from BT (if any) whose clusters are proper subsets of $\Lambda(T_A[p_i])$. To do this, $counter(x)$ for every node x in T_B is updated so that

it stores the number of leaves in $\Lambda(T_B[x]) \cap \Lambda(T_A[p_i])$ for the current i , and if $\text{counter}(x)$ reaches the value $|\Lambda(T_B[x])|$ then x is removed from BT .

To compute β_i in Step 5.7, take the maximum of: (i) β_{i-1} ; (ii) the weights of all special nodes on the path between r_i and r_{i-1} in T_B ; and (iii) the weights of all special nodes that belong to a path between r_i and a leaf in D .

Lemma 7. *Procedure `Filter_Clusters` runs in $O(n \log^2 n)$ time.*

Proof. (Omitted from the conference version due to space constraints.) □

5 Implementations

As noted in Section 1.2, there does not seem to be any publicly available implementation for the majority rule (+) consensus tree. To fill this void, we implemented Algorithm `Maj_Rule_Plus` from Section 3 in C++ and included it in the source code of the FACT (Fast Algorithms for Consensus Trees) package [17] at:

<http://compbio.ddns.comp.nus.edu.sg/~consensus.tree/>

To test the implementation, we repeatedly applied it to 10 random sets of trees for various specified values of (k, n) , generated with the method described in Section 6.2 of [17]. The following worst-case running times (in seconds) were obtained using Ubuntu Nutty Narwhal, a 64-bit operating system with 8.00 GB RAM, and a 2.20 GHz CPU:

(k, n)	(100, 500)	(100, 1000)	(100, 2000)	(100, 5000)	(500, 100)	(1000, 100)	(2000, 100)	(5000, 100)	(1000, 2000)
Time	0.63	1.51	2.99	6.78	0.65	1.29	2.72	6.66	27.29

The situation for the frequency difference consensus tree is less critical as there already exist implementations, e.g., in the software package TNT [15]. Nevertheless, it could be useful to implement our algorithm `Frequency_Difference` from Section 4 in the future and compare its practical performance to TNT. Before doing that, one should try to simplify the procedure `Filter_Clusters`.

References

- Adams III, E.N.: Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology* 21(4), 390–397 (1972)
- Amenta, N., Clarke, F., St. John, K.: A linear-time majority tree algorithm. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 216–227. Springer, Heidelberg (2003)
- Barthélemy, J.-P., McMorris, F.R.: The median procedure for n -trees. *Journal of Classification* 3(2), 329–334 (1986)
- Bremer, K.: Combinable component consensus. *Cladistics* 6(4), 369–372 (1990)
- Bryant, D.: A classification of consensus methods for phylogenetics. In: Janowitz, M.F., Lapointe, F.-J., McMorris, F.R., Mirkin, B., Roberts, F.S. (eds.) *Bioconsensus*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 61, pp. 163–184. American Mathematical Society (2003)
- Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., Thorup, M.: An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing* 30(5), 1385–1404 (2000)

7. Cotton, J.A., Wilkinson, M.: Majority-rule supertrees. *Systematic Biology* 56(3), 445–452 (2007)
8. Cui, Y., Jansson, J., Sung, W.-K.: Polynomial-time algorithms for building a consensus MUL-tree. *Journal of Computational Biology* 19(9), 1073–1088 (2012)
9. Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification* 2(1), 7–28 (1985)
10. Degnan, J.H., DeGiorgio, M., Bryant, D., Rosenberg, N.A.: Properties of consensus methods for inferring species trees from gene trees. *Systematic Biology* 58(1), 35–54 (2009)
11. Dong, J., Fernández-Baca, D., McMorris, F.R., Powers, R.C.: Majority-rule (+) consensus trees. *Mathematical Biosciences* 228(1), 10–15 (2010)
12. Felsenstein, J.: *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland (2004)
13. Felsenstein, J.: PHYLIP, version 3.6. Software package, Department of Genome Sciences. University of Washington, Seattle (2005)
14. Goloboff, P.A., Farris, J.S., Källersjö, M., Oxelman, B., Ramírez, M.J., Szumik, C.A.: Improvements to resampling measures of group support. *Cladistics* 19(4), 324–332 (2003)
15. Goloboff, P.A., Farris, J.S., Nixon, K.C.: TNT, a free program for phylogenetic analysis. *Cladistics* 24(5), 774–786 (2008)
16. Holder, M.T., Sukumaran, J., Lewis, P.O.: A justification for reporting the majority-rule consensus tree in Bayesian phylogenetics. *Systematic Biology* 57(5), 814–821 (2008)
17. Jansson, J., Shen, C., Sung, W.-K.: Improved algorithms for constructing consensus trees. In: *Proceedings of SODA 2013*, pp. 1800–1813. SIAM (2013)
18. Jansson, J., Shen, C., Sung, W.-K.: An optimal algorithm for building the majority rule consensus tree. In: Deng, M., Jiang, R., Sun, F., Zhang, X. (eds.) *RECOMB 2013*. LNCS, vol. 7821, pp. 88–99. Springer, Heidelberg (2013)
19. Jansson, J., Sung, W.-K.: Constructing the R^* consensus tree of two trees in sub-cubic time. *Algorithmica* 66(2), 329–345 (2013)
20. Lott, M., Spillner, A., Huber, K.T., Petri, A., Oxelman, B., Moulton, V.: Inferring polyploid phylogenies from multiply-labeled gene trees. *BMC Evolutionary Biology* 9, 216 (2009)
21. Margush, T., McMorris, F.R.: Consensus n -Trees. *Bulletin of Mathematical Biology* 43(2), 239–244 (1981)
22. McMorris, F.R., Powers, R.C.: A characterization of majority rule for hierarchies. *Journal of Classification* 25(2), 153–158 (2008)
23. Page, R.: COMPONENT, version 2.0. Software package, University of Glasgow, U.K. (1993)
24. Ronquist, F., Huelsenbeck, J.P.: MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics* 19(12), 1572–1574 (2003)
25. Sokal, R.R., Rohlf, F.J.: Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology* 30(3), 309–325 (1981)
26. Sukumaran, J., Holder, M.T.: DendroPy: A Python library for phylogenetic computing. *Bioinformatics* 26(12), 1569–1571 (2010)
27. Sung, W.-K.: *Algorithms in Bioinformatics: A Practical Introduction*. Chapman & Hall/CRC (2010)
28. Swofford, D.L.: PAUP*, version 4.0. Software package. Sinauer Associates, Inc., Sunderland (2003)

The Generalized Robinson-Foulds Metric^{*}

Sebastian Böcker^{1,**}, Stefan Canzar^{2,**}, and Gunnar W. Klau^{3,**}

¹ Chair for Bioinformatics, Friedrich Schiller University Jena, Germany
`sebastian.boecker@uni-jena.de`

² Center for Computational Biology, McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University School of Medicine, Baltimore, Maryland, USA
`canzar@jhu.edu`

³ Life Sciences Group, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
`gunnar.klau@cwi.nl`

Abstract. The Robinson-Foulds (RF) metric is arguably the most widely used measure of phylogenetic tree similarity, despite its well-known shortcomings: For example, moving a single taxon in a tree can result in a tree that has maximum distance to the original one; but the two trees are identical if we remove the single taxon. To this end, we propose a natural extension of the RF metric that does not simply count *identical* clades but instead, also takes *similar* clades into consideration. In contrast to previous approaches, our model requires the matching between clades to respect the structure of the two trees, a property that the classical RF metric exhibits, too. We show that computing this generalized RF metric is, unfortunately, NP-hard. We then present a simple Integer Linear Program for its computation, and evaluate it by an all-against-all comparison of 100 trees from a benchmark data set. We find that matchings that respect the tree structure differ significantly from those that do not, underlining the importance of this natural condition.

1 Introduction

In 1981, Robinson and Foulds introduced an intriguingly simple yet intuitively well-motivated metric, which is nowadays known as *Robinson-Foulds (RF) metric* [18]. Given two phylogenetic trees, this metric counts the number of splits or clades induced by one of the trees but not the other. The RF metric is highly conservative, as only perfectly conserved splits or clades do not count towards the distance. The degree of conservation between any pair of clades that is not perfectly conserved, does not change the RF distance. See Fig. 1 for an example of two trees that are structurally similar but have maximum RF distance.

Other measures for comparing phylogenetic trees do capture that the trees in Fig. 1 are structurally similar: The Maximum Agreement Subtree (MAST)

^{*} This work is supported in part by the National Institutes of Health under grant R01 HG006677.

^{**} Equal contribution.

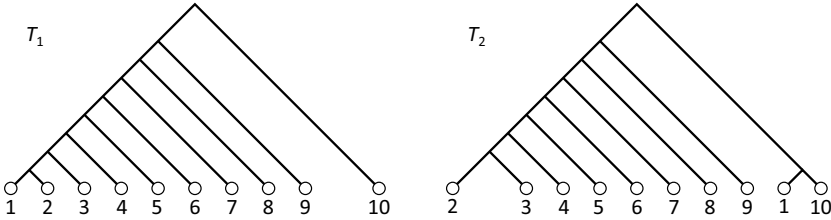


Fig. 1. Two rooted phylogenetic trees. Despite their high similarity, the RF distance of these two trees is 16, the maximum distance of two rooted trees with ten leaves.

score [11, 13] of the two trees is 9, where 10 is the highest possible score of two trees with 10 leaves. Secondly, the triplet distance counts the number of induced triplet trees on three taxa that are not shared by the two trees [2, 6]. Both measures are less frequently applied than the RF metric, and one may argue that this is due to certain “issues” of these measures: For example, if the trees contain (soft) polytomies or arbitrarily resolved polytomies, then we may have to exclude large parts of the trees from the MAST due to a single polytomy. Lastly, there are distance measures based on the number of branch-swapping operations to transform one tree into another; many of these measures are computationally hard to compute [1]. Such tree modifications are routinely used in local search optimization procedures, but rarely to compute distances in practice.

From an applied view, the comparison of two phylogenetic trees with identical taxa set has been frequently addressed in the literature [12, 16, 17]. This is of interest for comparing phylogenetic trees computed using different methods, output trees of an (MC)MCMC method, or host-parasite comparisons. Mutzner *et al.* [16] introduced the “best corresponding node” concept which, unfortunately, is not symmetric: Node a in the first tree may correspond to node b in the second, whereas b corresponds to a different node c in the first tree, and so on. Nye *et al.* [17] suggested to compute a matching between the inner nodes of the two trees, thereby enforcing symmetry. Later, Bogdanowicz [3] and, independently, Lin *et al.* [15] proposed to use these matchings to introduce a “generalized” version of the RF distance, see also [4]. Using matchings for comparing trees as part of MAST computations, was pioneered by Kao *et al.* [13].

Here, we present a straightforward generalization of the RF distance that allows us to relax its highly conservative behavior. At the same time, we can make this distance “arbitrarily similar” to the original RF distance. Unfortunately, computing this new distance is NP-hard, as we will show in Section 3. Our work generalizes and formalizes that of Nye *et al.* [17]: Their clade matching does not respect the structure of the two trees, see Fig. 1 and below. As a consequence, the matching distances from [3, 15] are no proper generalization of the RF distance: These distances treat the two input trees as collections of (unrelated) clades but *ignore the tree topologies*. In contrast, the RF distance does respect tree topologies, and so does our generalization.

In the following, we will concentrate on rooted phylogenetic trees.

2 The Generalized Robinson-Foulds Distance

Let $T = (V, E)$ be a rooted phylogenetic tree over the set of taxa X : That is, the leaves of T are (labeled by) the taxa X . We assume that T is arboreal, so all edges of T are pointing away from the root. In the following, we assume that any tree is an arboreal, rooted phylogenetic tree, unless stated otherwise. A set $Y \subseteq X$ is a *clade* of T if there exists some vertex $v \in V$ such that Y is the set of leaves below v . We call Y *trivial* if $|Y| = 1$ or $Y = X$. Since the trivial clades are identical for any two trees with taxa set X , we will restrict ourselves to the set $\mathcal{C}(T)$ of non-trivial clades of T . Let $\mathcal{P}(X)$ be the set of subsets of X .

Let T_1, T_2 be two phylogenetic trees over the set of taxa X , and let $\mathcal{C}_j := \mathcal{C}(T_j)$ for $j = 1, 2$ be the corresponding sets of non-trivial clades. The original RF distance counts zero whenever we can find a clade in both trees, and one if we find it in exactly one tree. We want to relax this by computing a matching between the clades of the two trees, and by assigning a cost function that measures the dissimilarity between the matched clades. To this end, we define a *cost function*

$$\delta : (\mathcal{P}(X) \cup \{-\}) \times (\mathcal{P}(X) \cup \{-\}) \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\} . \quad (1)$$

Now, $\delta(Y_1, Y_2)$ measures the dissimilarity of two arbitrary clades $Y_1, Y_2 \subseteq X$. The symbol ‘ $-$ ’ is the gap symbol, and we define $\delta(Y_1, -) > 0$ to be the cost of leaving some clade Y_1 of the first tree without a counterpart in the second tree; analogously, we define $\delta(-, Y_2) > 0$.

2.1 Matchings and Arboreal Matchings

Let $m \subseteq \mathcal{C}_1 \times \mathcal{C}_2$ be a *matching* between \mathcal{C}_1 and \mathcal{C}_2 : That is, $(Y_1, Y_2), (Y'_1, Y'_2) \in m$ implies $Y_1 = Y'_1$, and $(Y_1, Y_2), (Y_1, Y'_2) \in m$ implies $Y_2 = Y'_2$. We say that $Y_1 \in \mathcal{C}_1$ (or $Y_2 \in \mathcal{C}_2$) is *unmatched* if there is no $(Y'_1, Y'_2) \in m$ with $Y_1 = Y'_1$ (or $Y_2 = Y'_2$, respectively). We define the *cost* $d(m)$ of the matching m as:

$$d(m) := \sum_{(Y_1, Y_2) \in m} \delta(Y_1, Y_2) + \sum_{\substack{Y_1 \in \mathcal{C}_1 \\ Y_1 \text{ unmatched}}} \delta(Y_1, -) + \sum_{\substack{Y_2 \in \mathcal{C}_2 \\ Y_2 \text{ unmatched}}} \delta(-, Y_2) \quad (2)$$

Now, we could define a generalization of the Robinson-Foulds distance between T_1, T_2 (with respect to δ) to be the minimum cost of any matching between \mathcal{C}_1 and \mathcal{C}_2 . One can easily see that for $\delta(Y, Y) = 0$, $\delta(Y, Y') = \infty$ for $Y \neq Y'$, and $\delta(Y, -) = \delta(-, Y) = 1$ we reach the original RF distance.

How can we compute a matching of minimum cost? This is actually straightforward: We define a complete bipartite graph G with vertex set $\mathcal{C}_1 \cup \mathcal{C}_2$, and for any pair $C_1 \in \mathcal{C}_1, C_2 \in \mathcal{C}_2$ we define the weight of the edge (C_1, C_2) as $w(C_1, C_2) := \delta(C_1, -) + \delta(-, C_2) - \delta(C_1, C_2)$. Now, finding a matching with minimum cost corresponds to finding a maximum matching in G . In case δ is a metric, all edges in G have non-negative weight.

Unfortunately, finding a minimum cost matching will usually result in an unexpected—and undesired—behavior: Consider the two trees from Fig. 1 together with the cost function

$$\delta(Y_1, Y_2) = |Y_1 \cup Y_2| - |Y_1 \cap Y_2| = |Y_1 \triangle Y_2| \quad (3)$$

which is the cardinality of the *symmetric difference* $Y_1 \triangle Y_2$ of Y_1, Y_2 . In addition, we define $\delta(Y, -) = \delta(-, Y) = |Y|$. We note that δ is a metric. One can easily see that the matching with minimum cost matches clade $\{1, \dots, j\}$ from T_1 to $\{2, \dots, j\}$ from T_2 for all $j = 3, \dots, 10$. But in addition, clade $\{1, 2\}$ from T_1 is matched to clade $\{1, 10\}$ from T_2 , since

$$\delta(\{1, 2\}, \{1, 10\}) = 2 < 4 = \delta(\{1, 2\}, -) + \delta(-, \{1, 10\}) .$$

This means that the matching with minimum cost does not respect the structure of the two trees T_1, T_2 : Clade $\{1, 2\}$ in T_1 is a subclade of all $\{1, \dots, j\}$ whereas clade $\{1, 10\}$ in T_2 is no subclade of any $\{2, \dots, j\}$, for $j = 3, \dots, 10$. To this end, clades $\{1, 2\}$ and $\{1, 10\}$ should not be matched in a “reasonable” matching.

We say that a matching m is *arboreal* if no pair of matched clades is in *conflict*, that is, for any $(Y_1, Y_2), (Y'_1, Y'_2) \in m$, one of the three cases holds:

- (i) $Y_1 \subseteq Y'_1$ and $Y_2 \subseteq Y'_2$;
- (ii) $Y_1 \supseteq Y'_1$ and $Y_2 \supseteq Y'_2$; or
- (iii) $Y_1 \cap Y'_1 = \emptyset$ and $Y_2 \cap Y'_2 = \emptyset$.

This allows us to define the *generalized Robinson-Foulds distance* between T_1, T_2 (with respect to δ) to be the minimum cost of a arboreal matching between \mathcal{C}_1 and \mathcal{C}_2 . Whereas it is straightforward to compute a bipartite matching of minimum cost, it is less clear how to obtain an minimum cost arboreal bipartite matching. The formal problem statement is as follows:

Minimum Cost Arboreal Bipartite Matching. Given two rooted phylogenetic trees T_1, T_2 on X and a cost function δ , find a arboreal matching between $\mathcal{C}(T_1)$ and $\mathcal{C}(T_2)$ of minimum cost, as defined in (2).

This problem differs from the NP-complete *tree-constrained bipartite matching problem introduced in [5] in that cases (i) and (ii) are considered infeasible in [5]*. Unfortunately, the problem remains NP-complete, as we will show in Sec. 3.

For arbitrary cost functions δ we cannot draw conclusions about the resulting generalized Robinson-Foulds distance. But in case δ is a metric, this distance is a metric, too:

Lemma 1. *Given a metric δ as defined in (1); then, the induced generalized Robinson-Foulds distance d_{GRF} is a metric on the set of phylogenetic rooted trees on X .*

For the proof, the central point is that the combination of two arboreal matchings is also a arboreal matching; we defer the details to the full version of this paper.

2.2 The Jaccard-Robinson-Foulds Metric

Up to this point, we have assumed that δ can be an arbitrary metric. Now, we suggest one particular type that, again, appears quite naturally as a generalization of the original Robinson-Foulds metric: Namely, we will concentrate on a

measure that is motivated by the Jaccard index $J(A, B) = |A \cap B| / |A \cup B|$ of two sets A, B . For two clades Y, Y' , we define the *Jaccard weights of order k* as

$$\delta_k(Y, Y') := 2 - 2 \cdot \left(\frac{|Y \cap Y'|}{|Y \cup Y'|} \right)^k \quad (4)$$

where $k \geq 1$ is an arbitrary (usually integer) constant. In addition, we define $\delta_k(Y, -) = \delta_k(-, Y') = 1$ and, for completeness, $\delta_k(\emptyset, \emptyset) = 0$. The factor “2” in eq. (4) is chosen to guarantee compatibility with the original Robinson-Foulds metric. Nye *et al.* [17] suggested a similar metric without the exponent k . It is straightforward to check that (4) defines a metric, see [8] and the full version of this paper. We call the generalized Robinson-Foulds metric using δ_k from (4) the *Jaccard-Robinson-Foulds (JRF) metric of order k* , and denote it by $d_{\text{JRF}}^{(k)}$. More precisely, for two trees T_1, T_2 , $d_{\text{JRF}}^{(k)}(T_1, T_2)$ denotes the minimum cost of any matching between $\mathcal{C}(T_1)$ and $\mathcal{C}(T_2)$, using δ_k from (4) in (2).

For any two trees and any $k \geq 1$ we clearly have $d_{\text{JRF}}^{(k)}(T_1, T_2) \leq d_{\text{RF}}(T_1, T_2)$, as the matching of the RF metric is clearly arboreal. For $k \rightarrow \infty$ we reach $\delta_k(Y, Y) \rightarrow 0$ and $\delta_k(Y, Y') \rightarrow 1$ for $Y \neq Y'$, the inverse Kronecker delta. To this end, the JRF metric $d_{\text{JRF}}^{(k)}$ also converges to the original Robinson-Foulds metric d_{RF} . Furthermore, for any two trees T_1, T_2 there exists some k' such that for all $k \geq k'$, the matchings for d_{RF} and $d_{\text{JRF}}^{(k)}$ are “basically identical”: All exact clade matches will be contained in the matching of $d_{\text{JRF}}^{(k)}$. We defer the details to the full version of this paper.

3 Complexity of the Problem

In this section we prove hardness of the minimum arboreal matching problem, even if δ (and thus the induced RF distance, see Lemma 1) is a metric.

In the following we devise a polynomial-time reduction τ from (3, 4)-SAT, the problem of deciding whether a Boolean formula in which every clause is a disjunction of exactly 3 literals and every variable occurs 4 times, has a satisfying assignment. This problem was shown to be NP-hard in [10]. Given a formula φ with m clauses over n variables, we construct a minimum arboreal matching instance I under metric (3), such that φ is satisfiable if and only if I admits a matching of cost $d(M_0) - 10n - 26m - 5 \cdot 2^4 - (k + 1)2^k - q$, where M_0 is the empty matching.

For each variable x_i we construct a gadget as shown in Figure 2. The next lemma shows that, under certain assumptions, there are precisely two optimal solutions to the variable gadgets. We will use these two matchings to represent a truth assignment to variable x_i .

Lemma 2. *Consider the gadget of a variable x_i as depicted in Figure 2. Under the restriction that none of the ancestors of nodes v and v' is matched, there are two optimal matchings of trees T_{1i} and T_2 of cost $d(M_0) - 10$. M_{1i} contains (v, v') and (u, u') and matches leaves labeled l_i and α_i , and $M_{\bar{1}i}$ contains (v, v'') and (u, u'') and matches leaves labeled \bar{l}_i and $\bar{\alpha}_i$.*

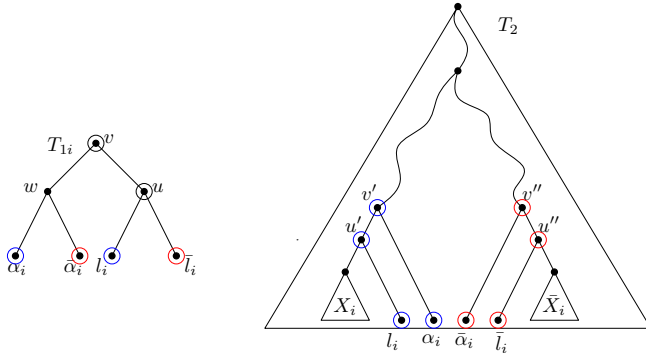


Fig. 2. Variable Gadget. Vertices covered by optimal matchings M_l and $M_{\bar{l}}$ are marked in blue and red, respectively. Vertices marked in black are covered in both optimal matchings.

Proof. In the following we assume that none of the ancestors of v and v' can be matched. Let $M_0 = \emptyset$ be the empty matching between T_{1i} and T_2 , and let M_a denote the matching that matches all leaves with identical labels. Then, M_a is maximal and $d(M_a) = d(M_0) - 8$. If we match u to either u' or to u'' , a feasible matching cannot match leaves labeled \bar{l}_i or leaves labeled l_i , respectively. Similarly, matching w to v' or to v'' invalidates the matching of leaves labeled $\bar{\alpha}_i$ or leaves labeled α_i , respectively. In both cases the overall cost remains unchanged compared to M_a . If we match v to v' , only leaves labeled l_i and α_i can be matched to corresponding leaves in T_2 . A feasible matching of node w to any node in T_{2i} does not reduce the total cost, since none of the labels of descendants of v' contains α_i or $\bar{\alpha}_i$. However, matching u to u' does not introduce any conflict and further decreases the cost. The resulting matching (see Figure 2), M_l , has cost $d(M_l) = d(M_0) - 10$. By a symmetric argument, a maximum matching $M_{\bar{l}}$ containing (v, v'') matches u to u'' and leaves labeled \bar{l}_i and $\bar{\alpha}_i$, with $d(M_{\bar{l}}) = d(M_0) - 10$.

For each clause C_j we construct a clause gadget as shown in Figure 3.

Lemma 3. *Consider the gadget of a clause C_j as depicted in Figure 3. Under the restriction that no common ancestor of w_i , w_k , or w_l is matched, there exists an optimal matching M of C_j and T_2 that matches all vertices in one of the subtrees rooted at w_i , w_k , or w_l and none of the remaining vertices, and has cost $d(M) = d(M_0) - 26$.*

Proof. Let $M_0 = \emptyset$ be the empty matching between C_j and T_2 , and let M_a denote the matching that matches all leaves with identical labels. Then M_a is maximal and $d(M_a) = d(M_0) - 24$. Matching any non-leaf node below u , v , or w in C_j to a node in T_2 that is not an ancestor of w_i , w_k , or w_l , yields a matching of cost at least as high as $d(M_a)$: At most one leaf in the subtree rooted at such

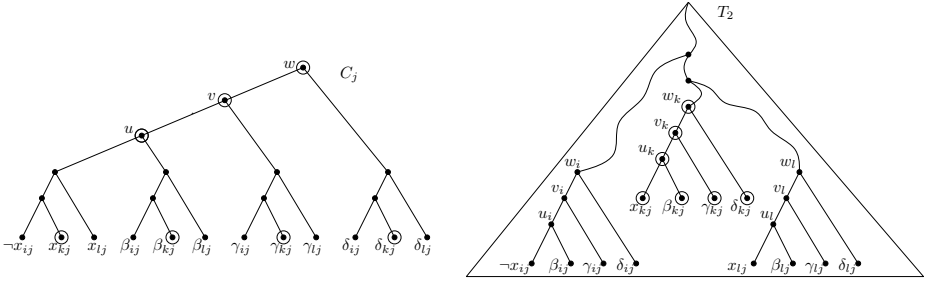


Fig. 3. Clause gadget for clause $C_j = (-x_i \vee x_k \vee x_l)$. Vertices covered by an optimal matching are marked.

a node u' can be matched to its corresponding leaf in T_2 , while the label overlap of u' with nodes in T_2 that are not ancestors of $w_i, w_k,$ or w_l , is at most 1.

If node u is matched to a node in T_2 with maximal label overlap that is not an ancestor of $w_i, w_k,$ or w_l , only 2 leaves in the subtree rooted at u can be matched to the corresponding leaves in T_2 . If the remaining nodes in T_1 are matched according to M_a the resulting matching has cost $d(M_0) - 20$.

Matching node v to a node in T_2 with maximal overlap that is not an ancestor of $w_i, w_k,$ or w_l , allows only 3 leaves in the subtree rooted at v to be matched to the corresponding leaves in T_2 . Additionally node u can be matched to a node in T_2 with label overlap of size 2. Matching the remaining nodes in C_j according to M_a yields a matching of cost $d(M_0) - 22$.

Finally, if node w is matched to a node in T_2 with maximal label overlap that is not an ancestor of $w_i, w_k,$ or w_l , in total 4 leaves in C_j can be matched to the corresponding leaves in T_2 . At the same time, u and v can be matched to nodes with maximal label overlap, yielding a matching M of cost $d(M) = d(M_0) - 26$ (see Figure 3). Since all edges in M have maximum label overlap under the assumption that no common ancestor of $w_i, w_k,$ or w_l is matched, M is optimal.

Next, we show how variable and clause gadgets together form $\tau(\varphi)$. For each occurrence of a positive or negative literal l_i or \bar{l}_i in a clause j we denote the subtrees rooted at $w_i, w_k,$ and w_l in T_2 (Figure 3) by L_{ij} or \bar{L}_{ij} , respectively. T_2 in Figure 3 show trees $\bar{L}_{ij}, L_{kj},$ and L_{lj} . Let j_1, \dots, j_h be the indices of clauses in which positive literal l_i occurs. Then, module X_i in Figure 2 is constructed as shown in Figure 4. Module \bar{X}_i is analogously composed of trees \bar{L} .

From variables gadgets (Figure 2) and clause gadgets (Figure 3) we construct two rooted trees T_1 and T_2 as depicted in Figure 5, where trees T_{2i} and \bar{T}_{2i} denote subtrees rooted at v' and v'' , respectively, in T_2 (Figure 2). T_1 and T_2 , together with cost function (3), form our instance $\tau(\varphi)$. Both trees connect subtrees of variable and clause gadgets in linear chains, augmented by two separator trees S_0 and S_1 . S_1 represents a complete binary trees of depth 4, and S_0 a complete binary tree of depth $k = \lceil \log(40n^2 + 141) \rceil$.

We assign leaves of separator trees arbitrary but unique taxa in a way, such that tree S_i in T_1 is an identical copy of S_i in $T_2, i \in \{1, 2\}$.

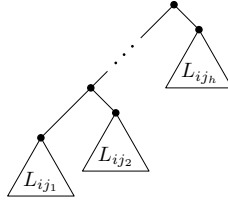


Fig. 4. Module X_i in the variable gadget for x_i is composed of one tree L_{ij} for each occurrence j of positive literal x_i

Lemma 4. Consider the construction $\tau(\varphi)$ in Figure 5. In an optimal matching of trees T_1 and T_2 , nodes in the backbone of T_1 , $\mathcal{B}_1 := \{c_1, \dots, c_m, t_{11}, \dots, t_{1n}\}$, and nodes in the backbone of T_2 , $\mathcal{B}_2 := \{t_{22}, \dots, t_{2n}, t'_{22}, \dots, t'_{2n}\}$, are unmatched.

Proof. First, an optimal solution must match roots r_1, r_2 , since edge (r_1, r_2) does not introduce any constraint on the remaining vertices and has maximum label overlap. Therefore, matching any node in \mathcal{B}_1 invalidates the matching of nodes in S_0 . According to conditions (i)-(iii), a feasible matching cannot match nodes from different subtrees in $\mathcal{T}_2 = \{T_{21}, \dots, T_{2n}, T'_{21}, \dots, T'_{2n}, S_0, S_1\}$ to nodes in \mathcal{B}_1 . Replacing all edges incident to nodes in \mathcal{B}_1 by a full matching of nodes in S_0 reduces the cost by at least

$$2 \left(\sum_{u \in S_0} |Y(u)| - \sum_{i=1}^n (|Y(t_{2i})| + |Y(t'_{2i})|) - \max_{T \in \mathcal{T}_2 \setminus \{S_0\}} \sum_{v \in T} |Y(v)| \right) \geq 2((k+1) \cdot 2^k - 16 - 40n^2 - 125), \tag{5}$$

where k is the depth of S_0 . The upper bound of 125 on $\sum_{v \in T_{2j}} |Y(u)|$ assumes that each variable occurs in at most 4 clauses, and $125 > \sum_{v \in S_1} |Y(u)|$. Note that the taxa assigned to the 16 leaves of S_1 are contained only in $Y(r_1)$ and that for each i , $|Y(t_{2i})| + |Y(t'_{2i})| \leq 20$. For the above chosen k it holds $(k+1) \cdot 2^k > 40n^2 + 141$.

Similarly, a feasible matching cannot match nodes from different subtrees in $\mathcal{T}_1 := \{C_1, \dots, C_m, T_{11}, \dots, T_{1n}, S_0, S_1\}$ to nodes in \mathcal{B}_2 . Assume the optimal solution matches nodes in a subtree C_i to nodes in \mathcal{B}_2 . Since every node in \mathcal{B}_2 is ancestor of S_1 , the nodes of S_1 are unmatched. Replacing the edges between C_i and \mathcal{B}_2 by a full matching of nodes in S_1 reduces the cost by at least

$$2 \left(\sum_{u \in S_1} |Y(u)| - \sum_{v \in C_i} |Y(u)| \right) = 2(80 - 59) > 0,$$

a contradiction. An analog argument applies to matching nodes in one of the trees T_{1i} to nodes in \mathcal{B}_2 , with $\sum_{u \in T_{1i}} Y(u) = 12 < \sum_{u \in S_1} Y(u)$. As the optimal matching of trees S_0 has cost 0, matching at least one node in S_0 in T_1 to a node in \mathcal{B}_2 strictly increases the overall cost.

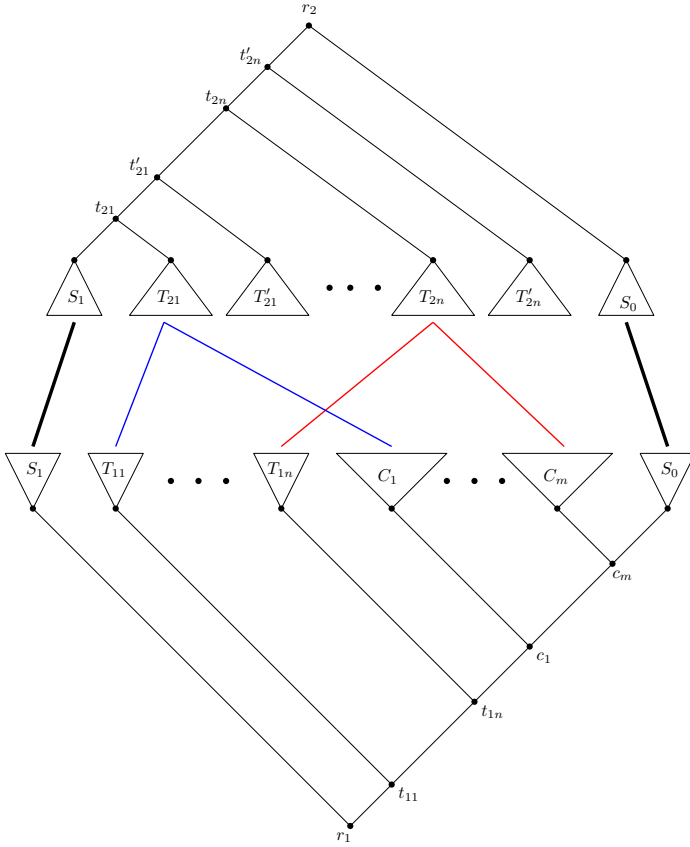


Fig. 5. Trees T_1 with root r_1 and T_2 with root r_2 in instance I , obtained from $\tau(\varphi)$. In an optimal solution trees S_0 and S_1 are fully aligned to each other (black lines). If a variable gadget is in M_l configuration (blue line), a clause in which the corresponding negative literal occurs can be matched optimally (blue line). The same holds for configuration $M_{\bar{l}}$ and positive literal occurrences (red lines).

Now we are ready to state the main theorem.

Theorem 1. *For an instance of the minimum arboreal matching problem with cost function (3) and an integer k , it is NP-complete to decide whether there exists an arboreal matching of cost at most k .*

Proof. First, we show that if φ is satisfiable, then $\tau(\varphi)$ admits a matching M of cost $d(M_0) - 10n - 26m - 5 \cdot 2^4 - (k + 1)2^k - q$, where k is the depth of tree S_0 and q is total number of leaves of tree T_1 or, equivalently, tree T_2 . For this, let ν be a satisfying assignment for φ . We start from $M = \emptyset$. For each variable x_i we set the corresponding variable gadget to configuration M_l if $\nu(x_i) = \text{false}$ and to configuration $M_{\bar{l}}$ if $\nu(x_i) = \text{true}$, each having cost $d(M_0) - 10$ (Lemma 2).

Additionally, we match each subtree representing a clause C_j to subtree T_{2i} or T'_{2i} following the construction in Lemma 3, where literal x_i or $\neg x_i$, respectively, is contained in C_j and evaluates to true under the assignment ν . Note that none of the ancestors of subtree X_i or \bar{X}_i (see Figure 2), respectively, is matched in this case (Lemma 2 and Lemma 4). Each clause therefore contributes $d(M_0) - 26$ to the overall cost (Lemma 3). Finally, trees S_0 and S_1 are covered by full matchings of their nodes and the roots r_1, r_2 are matched, yielding a matching of total cost

$$d(M_0) - 10n - 26m - 5 \cdot 2^4 - (k + 1)2^k - q \tag{6}$$

As an optimal solution matches roots r_1 and r_2 but none of the nodes in \mathcal{B}_1 or \mathcal{B}_2 (Lemma 4), any optimal matching must match subtrees in $\mathcal{T}_1 := \{C_1, \dots, C_m, T_{11}, \dots, T_{1n}, S_0, S_1\}$ and $\mathcal{T}_2 = \{T_{21}, \dots, T_{2n}, T'_{21}, \dots, T'_{2n}, S_0, S_1\}$ optimally. Since an optimal matching of any tree in \mathcal{T}_1 to \mathcal{T}_2 and vice versa is given by Lemmas 2 and 3, one can always derive a satisfying assignment of φ from M . Therefore, if φ is not satisfiable, the weight of a maximum matching in $\tau(\varphi)$ is strictly larger than (6).

4 An Integer Linear Program

In this section we introduce a simple integer linear programming formulation for the problem of finding a minimum cost arboreal matching between $\mathcal{C}(T_1)$ and $\mathcal{C}(T_2)$, given two rooted phylogenetic trees $T_1 = (V_1, E_1)$, $T_2 = (V_2, E_2)$, and a cost function δ . We number clades C in $\mathcal{C}(T_1)$ from 1 to $|V_1|$ and clades \bar{C} in $\mathcal{C}(T_1)$ from 1 to $|V_2|$. An indicator variable $x_{i,j}$ denotes whether $(C_i, \bar{C}_j) \in m$ ($x_{i,j} = 1$) or not ($x_{i,j} = 0$). Set \mathcal{I} contains pairs of matched clades $\{(i, j), (k, l)\}$ that are *incompatible* according to conditions (i)-(iii). With $w(C_1, C_2) := \delta(C_1, -) + \delta(-, C_2) - \delta(C_1, C_2)$ (see Section 2.1) a minimum cost arboreal matching is represented by the optimal solution to:

$$\max \sum_{i=1}^{|V_1|} \sum_{j=1}^{|V_2|} w(C_i, \bar{C}_j) x_{i,j} \tag{7}$$

$$\text{s. t. } \sum_{j=1}^{|V_2|} x_{i,j} \leq 1 \quad \forall i = 1 \dots |V_1|, \tag{8}$$

$$\sum_{i=1}^{|V_1|} x_{i,j} \leq 1 \quad \forall j = 1 \dots |V_2|, \tag{9}$$

$$x_{i,j} + x_{k,l} \leq 1 \quad \forall \{(i, j), (k, l)\} \in \mathcal{I}, \tag{10}$$

$$x_{i,j} \in \{0, 1\} \tag{11}$$

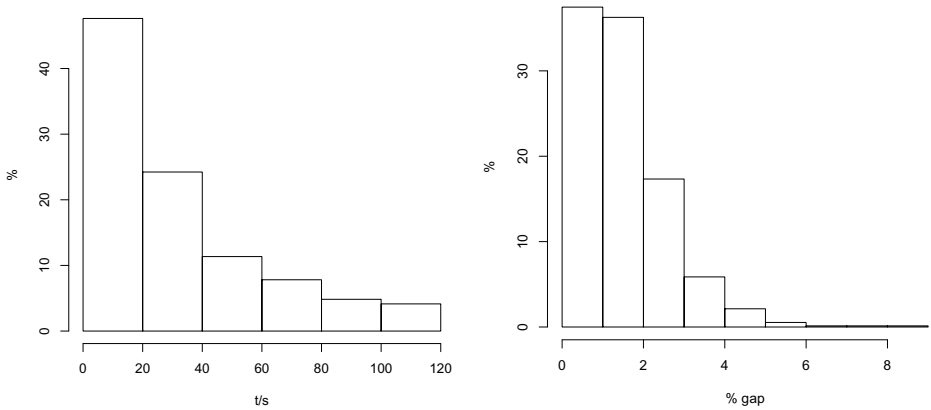


Fig. 6. Running time and optimality gap statistics of the 5050 benchmark instances. Left: histogram of running times of the 4300 instances that were solved to optimality within 2 CPU minutes. Right: histogram of the optimality gap in percent of the remaining 750 instances. This value is defined as $100 \cdot (u - l)/l$, where u and l are the upper and lower bounds of the arboreal matching, respectively.

5 Evaluation

We use a real-world dataset provided by Sul and Williams [19] as part of the HashRF program.¹ It contains 1000 phylogenetic trees from a Bayesian analysis of 150 green algae [14]. For the purpose of this comparison we performed an all-against-all comparison of the first hundred trees in the benchmark set as a proof-of-concept study, resulting in 5050 problem instances. We compute the values of the Robinson-Foulds metric as well as the minimum arboreal matching using the Jaccard weights of order $k = 1$, that is, the JRF metric $d_{\text{JRF}}^{(1)}$. We limit the computation to two CPU minutes per comparison and record the times for computing each value as well as the best upper and lower bounds for $d_{\text{JRF}}^{(1)}$.

From the 5050 instances, 4300 (85 %) could be computed to optimality within the time limit on an Intel Xeon CPU E5-2620 with 2.00 GHz. Most of these instances could be solved within 40 CPU s. See Fig. 6 (left) for a histogram of running times. The remaining 750 instances (15 %) were solved close to optimality. Fig. 6 (right) shows a histogram of the relative optimality gap in percent. This value is defined as $100 \cdot (u - l)/l$, where u and l are the upper and lower bounds of the arboreal matching, respectively. Overall, the majority (3578 instances, 71 %) could be solved to optimality within a minute. Note that these results are obtained the quite simple Integer Linear Programming formulation presented in this paper. Improvements on the formulation will likely lead to a drastic reduction of the running time.

Figure 7 shows typical characteristics of the arboreal JRF distances over increasing k for a randomly picked instance (tree 34 vs. tree 48). We observe that

¹ Trees can be downloaded from <https://code.google.com/p/hashrf/>

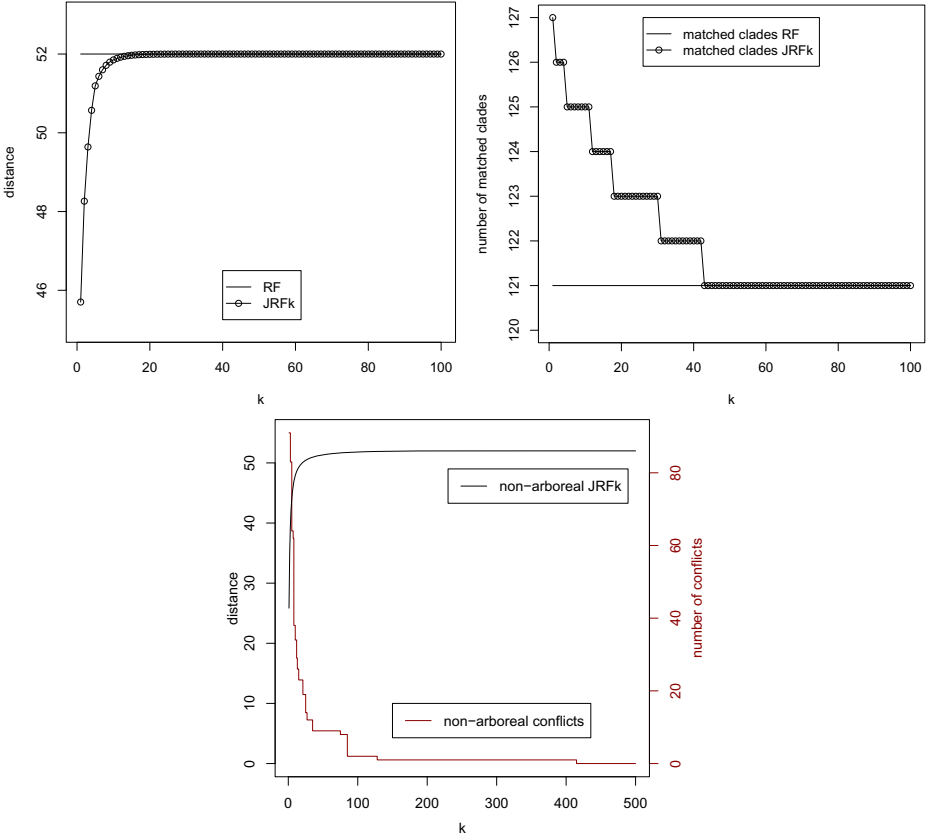


Fig. 7. Typical characteristics of the distances over increasing k . In the randomly chosen example (tree 34 vs. tree 48), RF equals 52 and $d_{\text{JRF}}^{(1)}$ is depicted by circles connected by lines (left plot). In the right plot we see that the number of matched clades decreases with increasing k . The plot below shows the development of distance and number of conflicts of the non-arboreal matching for increasing k .

RF and $d_{\text{JRF}}^{(k)}$ distances differ considerably for $k = 1$ and that $d_{\text{JRF}}^{(k)}$ converges quickly to RF (Fig. 7, left). A similar converging behavior can be observed for the number of matched clades (Fig. 7, right). The bottom plot in Fig. 7 illustrates the difference to non-arboreal matchings. For $k = 1$, the distances differ significantly from the RF distance (25.8 versus 52), however, at the prize of a large number of violations of the arboreal property (91). As k increases, the distance converges quickly to the RF distance and the number of violations decreases. Note that zero violations occur only when the non-arboreal distance is equal to the RF distance.

6 Conclusion

We have introduced a tree metric that naturally extends the well-known Robinson-Foulds metric. Different from previous work, our metric is a true generalization, as it respects the structure of the trees when comparing clades. Besides the theoretical amenities of such a generalization, our methods naturally allows for a manual comparison of two trees, using the arboreal matching that has been computed. This allows us to compute “best corresponding nodes” that respect the tree structures, and to inform the user when other node correspondences disagree with the optimal matching. We believe that such a feature will be very useful for the manual comparison of two trees, for example, in host-parasite comparison.

An open question is the parameterized complexity of the problem, where natural parameters are the size of the matching or, more relevant in applications, the discrepancy between the size of the maximum arboreal matching and a regular maximum matching. The Maximum Independent Set problem is $W[1]$ -hard [9] but, obviously, this does not imply that our more restricted problem cannot be approached by a parameterized algorithm [7].

We have come up with a generalization that retains the advantages of the widely-used Robinson-Foulds metric, but simultaneously overcomes some of its shortcomings. Our results are a first step to make the GRF and JRF metrics applicable to practical problems. In the future, faster algorithms are needed for this purpose; we believe that such algorithms can and will be developed. Furthermore, we want to generalize our results for unrooted trees, along the lines of [17, Sec. 2.1]. Here, the main challenge lies in adapting the notion of an arboreal matching.

In the full version of this paper, we will evaluate the JRF metric following ideas of Lin *et al.* [15]: That is, we will compare distributions of distances with arboreal and non-arboreal matchings; and, we will estimate the power of the new distance with regards to clustering similar trees.

Acknowledgments. We thank W. T. J. White for helpful discussions.

References

1. Allen, B.L., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. *Annals Combinatorics* 5, 1–15 (2001)
2. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. *Theor. Comput. Sci.* 412(48), 6634–6652 (2011)
3. Bogdanowicz, D.: Comparing phylogenetic trees using a minimum weight perfect matching. In: *Proc. of Information Technology (IT 2008)*, pp. 1–4 (2008)
4. Bogdanowicz, D., Giaro, K.: Matching split distance for unrooted binary phylogenetic trees. *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 9(1), 150–160 (2012)
5. Canzar, S., Elbassioni, K., Klau, G., Mestre, J.: On tree-constrained matchings and generalizations. *Algorithmica*, 1–22 (2013)

6. Critchlow, D.E., Pearl, D.K., Qian, C.: The triples distance for rooted bifurcating phylogenetic trees. *Syst. Biol.* 45(3), 323–334 (1996)
7. Dabrowski, K., Lozin, V.V., Müller, H., Rautenbach, D.: Parameterized algorithms for the independent set problem in some hereditary graph classes. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 1–9. Springer, Heidelberg (2011)
8. Deza, M., Laurent, M.: *Geometry of Cuts and Metrics*. Springer, New York (1997)
9. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Berlin (1999)
10. Dubois, O.: On the r, s -SAT satisfiability problem and a conjecture of Tovey. *Discrete Applied Mathematics* 26(1), 51–60 (1990)
11. Finden, C., Gordon, A.: Obtaining common pruned trees. *J. Classif.* 2(1), 255–276 (1985)
12. Griebel, T., Brinkmeyer, M., Böcker, S.: EPoS: A modular software framework for phylogenetic analysis. *Bioinformatics* 24(20), 2399–2400 (2008)
13. Kao, M.-Y., Lam, T.W., Sung, W.-K., Ting, H.-F.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. Algorithms* 40(2), 212–233 (2001)
14. Lewis, L.A., Lewis, P.O.: Unearthing the molecular phylodiversity of desert soil green algae (Chlorophyta). *Syst. Biol.* 54(6), 936–947 (2005)
15. Lin, Y., Rajan, V., Moret, B.M.E.: A metric for phylogenetic trees based on matching. *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 9(4), 1014–1022 (2012)
16. Munzner, T., Guimbertière, F., Tasiran, S., Zhang, L., Zhou, Y.: TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. *ACM Trans. Graph.* 22(3), 453–462 (2003)
17. Nye, T.M.W., Liò, P., Gilks, W.R.: A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. *Bioinformatics* 22(1), 117–119 (2006)
18. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. *Math. Biosci.* 53(1-2), 131–147 (1981)
19. Sul, S.-J., Williams, T.L.: An experimental analysis of robinson-foulds distance matrix algorithms. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 793–804. Springer, Heidelberg (2008)

Computing the Skewness of the Phylogenetic Mean Pairwise Distance in Linear Time

Constantinos Tsirogiannis and Brody Sandel

MADALGO* and Department of Bioscience, Aarhus University, Denmark
{constant, brody.sandel}@cs.au.dk

Abstract. The phylogenetic Mean Pairwise Distance (MPD) is one of the most popular measures for computing the phylogenetic distance between a given group of species. More specifically, for a phylogenetic tree \mathcal{T} and for a set of species R represented by a subset of the leaf nodes of \mathcal{T} , the MPD of R is equal to the average cost of all possible simple paths in \mathcal{T} that connect pairs of nodes in R .

Among other phylogenetic measures, the MPD is used as a tool for deciding if the species of a given group R are closely related. To do this, it is important to compute not only the value of the MPD for this group but also the expectation, the variance, and the skewness of this metric. Although efficient algorithms have been developed for computing the expectation and the variance the MPD, there has been no approach so far for computing the skewness of this measure.

In the present work we describe how to compute the skewness of the MPD on a tree \mathcal{T} optimally, in $\Theta(n)$ time; here n is the size of the tree \mathcal{T} . So far this is the first result that leads to an exact, let alone efficient, computation of the skewness for any popular phylogenetic distance measure. Moreover, we show how we can compute in $\Theta(n)$ time several interesting quantities in \mathcal{T} that can be possibly used as building blocks for computing efficiently the skewness of other phylogenetic measures.

1 Introduction

Communities of co-occurring species may be described as “clustered” if species in the community tend to be close phylogenetic relatives of one another, or “overdispersed” if they are distant relatives [7]. To define these terms we need a function that measures the phylogenetic relatedness of a set of species, and also a point of reference for how this function should behave in the absence of ecological and evolutionary processes. One such function is the mean pairwise distance (MPD); given a phylogenetic tree \mathcal{T} and a subset of species R that are represented by leaf nodes of \mathcal{T} , the MPD of the species in R is equal to average cost of all possible simple paths that connect pairs of nodes in R .

To decide if the value of the MPD for a specific set of species R is large or small, we need to know the average value (expectation) of the MPD for all sets

* Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

of species in \mathcal{T} that consist of exactly $r = |R|$ species. To judge how much larger or smaller is this value from the average, we also need to know the standard deviation of the MPD for all possible sets of r species in \mathcal{T} . Putting all these values together, we get the following index that expresses how clustered are the species in R [7]:

$$\text{NRI} = \frac{\text{MPD}(\mathcal{T}, R) - \text{expec}_{\text{MPD}}(\mathcal{T}, r)}{sd_{\text{MPD}}(\mathcal{T}, r)},$$

where $\text{MPD}(\mathcal{T}, R)$ is the value of the MPD for R in \mathcal{T} , and $\text{expec}(\mathcal{T})$ and $sd_{\text{MPD}}(\mathcal{T}, r)$ are the expected value and the standard deviation respectively of the MPD calculated over all subsets of r species in \mathcal{T} .

In a previous paper we presented optimal algorithms for computing the expectation and the standard deviation of the MPD of a phylogenetic tree \mathcal{T} in $\Theta(n)$ time, where n is the number of the edges of \mathcal{T} [5]. This enabled exact computations of these statistical moments of the MPD on large trees, which were previously infeasible using traditional slow and inexact resampling techniques. However, one important problem remained unsolved; quantifying our degree of confidence that the NRI value observed in a community reflects non-random ecological and evolutionary processes.

This degree of confidence is a statistical P value, that is the probability that we would observe an NRI value as extreme or more so if the community were randomly assembled. Traditionally, estimating P is accomplished by ranking the observed MPD against the distribution of randomized MPD values [3]. If the MPD falls far enough into one of the tails of the distribution (generally below the 2.5 percentile or above the 97.5 percentile, yielding $P < 0.05$), the community is said to be significantly overdispersed or significantly clustered. However, this approach relies on sampling a large number of random subsets of species in \mathcal{T} , and recomputing the MPD for each random subset. Therefore, this method is slow and imprecise.

We can approximate the P value of an observed NRI by assuming a particular distribution of the possible MPD values and evaluating its cumulative distribution function at the observed MPD. Because the NRI measures the difference between the observed values and expectation in units of standard deviations, this yields a very simple rule if we assume that possible MPD values are normally distributed: any NRI value larger than 1.96 or smaller than -1.96 is significant. Unfortunately, the distribution of MPD values is often skewed, such that this simple rule will lead to incorrect P value estimates [1,6]. Of particular concern, this skewness introduces a bias towards detecting either significant clustering or significant overdispersion [2]. Calculating this skewness analytically would enable us to remove this bias and improve the accuracy of P value estimates obtained analytically. However, so far there has been no result in the related literature that shows how to compute this skewness value.

Hence, given a phylogenetic tree \mathcal{T} and an integer r there is the need to design an efficient and exact algorithm that can compute the skewness of the MPD for r species in \mathcal{T} . This would provide the last critical piece required for the adoption

of a fully analytical and efficient approach for analysing ecological communities using the MPD and the NRI.

Our Results. In the present work we tackle the problem of computing efficiently the skewness of the MPD. More specifically, given a tree \mathcal{T} that consists of n edges and a positive integer r , we prove that we can compute the skewness of the MPD over all subsets of r leaf nodes in \mathcal{T} optimally, in $\Theta(n)$ time. For the calculation of this skewness value we consider that every subset of exactly r species in \mathcal{T} is picked uniformly out of all possible subsets that have r species. The main contribution of this paper is a constructive proof that leads straightforwardly to an algorithm that computes the skewness of the MPD in $\Theta(n)$ time. This is clearly very efficient, especially if we consider that it outperforms the best algorithms that are known so far for computing lower-order statistics for other phylogenetic measures; for example the most efficient known algorithm for computing the variance of the popular Phylogenetic Distance (PD) runs in $O(n^2)$ time [5].

More than that, we prove how we can compute in $\Theta(n)$ time several quantities that are related with groups of paths in the given tree; these quantities can be possibly used as building blocks for computing efficiently the skewness (and other statistical moments) of phylogenetic measures that are similar to the MPD. Such an example is the measure which is the equivalent of the MPD for computing the distance between two subsets of species in \mathcal{T} [4].

The rest of this paper is, in its entirety, an elaborate proof for computing the skewness of the MPD on a tree \mathcal{T} in $\Theta(n)$ time. In the next section we define the problem that we want to tackle, and we present a group of quantities that we use as building blocks for computing the skewness of the MPD. We prove that all of these quantities can be computed in linear time with respect to the size of the input tree. In Section 3 we provide the main proof of this paper; there we show how we can express the value of the skewness of the MPD in terms of the quantities that we introduced earlier. The proof implies a straightforward linear time algorithm for the computation of the skewness as well.

2 Description of the Problem and Basic Concepts

Definitions and Notation. Let \mathcal{T} be a phylogenetic tree, and let E be the set of its edges. We denote the number of the edges in \mathcal{T} by n , that is $n = |E|$. For an edge $e \in E$, we use w_e to indicate the weight of this edge. We use S to denote the set of the leaf nodes of \mathcal{T} . We call these nodes the *tips* of the tree, and we use s to denote the number of these nodes.

Since a phylogenetic tree is a rooted tree, for any edge $e \in E$ we distinguish the two nodes adjacent to e into a *parent* node and a *child* node; among these two, the parent node of e is the one for which the simple path from this node to the root does not contain e . We use $\text{Ch}(e)$ to indicate the set of edges whose parent node is the child node of e , which of course implies that $e \notin \text{Ch}(e)$. We indicate the edge whose child node is the parent node of e by $\text{parent}(e)$. For any

edge $e \in E$, tree $\mathcal{T}(e)$ is the subtree of \mathcal{T} whose root is the child node of edge e . We denote the set of tips that appear in $\mathcal{T}(e)$ as $S(e)$, and we denote the number of these tips by $s(e)$.

Given any edge $e \in E$, we partition the edges of \mathcal{T} into three subsets. The first subset consists of all the edges that appear in the subtree of e . We denote this set by $\text{Off}(e)$. The second subset consists of all edges $e' \in E$ for which e appears in the subtree of e' . We use $\text{Anc}(e)$ to indicate this subset. For the rest of this paper, we define that $e \in \text{Anc}(e)$, and that $e \notin \text{Off}(e)$. The third subset contains all the tree edges that do not appear neither in $\text{Off}(e)$, nor in $\text{Anc}(e)$; we indicate this subset by $\text{Ind}(e)$.

For any two tips $u, v \in S$, we use $p(u, v)$ to indicate the simple path in \mathcal{T} between these nodes. Of course, the path $p(u, v)$ is unique since \mathcal{T} is a tree. We use $\text{cost}(u, v)$ to denote the cost of this path, that is the sum of the weights of all the edges that appear on the path. Let u be a tip in S and let e be an edge in E . We use $\text{cost}(u, e)$ to represent the cost of the shortest simple path between u and the child node of e . Therefore, if $u \in S(e)$ this path does not include e , otherwise it does. For any subset $R \subseteq S$ of the tips of the tree \mathcal{T} , we denote the set of all pairs of elements in R , that is the set of all combinations that consist of two distinct tips in R , by $\Delta(R)$. Given a phylogenetic tree \mathcal{T} and a subset of its tips $R \subseteq S$, we denote the Mean Pairwise Distance of R in \mathcal{T} by $\text{MPD}(\mathcal{T}, R)$. Let $r = |R|$. This measure is equal to:

$$\text{MPD}(\mathcal{T}, R) = \frac{2}{r(r-1)} \sum_{\{u,v\} \in \Delta(R)} \text{cost}(u, v) .$$

2.1 Aggregating the Costs of Paths

Let \mathcal{T} be a phylogenetic tree that consists of n edges and s tips, and let r be a positive integer such that $r \leq s$. We use $\text{sk}(\mathcal{T}, r)$ to denote the skewness of the MPD on \mathcal{T} when we pick a subset of r tips of this tree with uniform probability. In the rest of this paper we describe in detail how we can compute $\text{sk}(\mathcal{T}, r)$ in $O(n)$ time, by scanning \mathcal{T} only a constant number of times. Based on the formal definition of the concept of skewness, the value of $\text{sk}(\mathcal{T}, r)$ is equal to:

$$\begin{aligned} \text{sk}(\mathcal{T}, r) &= E_{R \in \text{Sub}(S,r)} \left[\left(\frac{\text{MPD}(\mathcal{T}, R) - \text{expec}(\mathcal{T}, r)}{\text{var}(\mathcal{T}, r)} \right)^3 \right] \\ &= \frac{E_{R \in \text{Sub}(S,r)} [\text{MPD}^3(\mathcal{T}, R)] - 3 \cdot \text{var}(\mathcal{T}, r)^2 - \text{expec}(\mathcal{T}, r)^3}{\text{var}(\mathcal{T}, r)^3} , \end{aligned} \quad (1)$$

where $\text{expec}(\mathcal{T}, r)$ and $\text{var}(\mathcal{T}, r)$ are the expectation and the variance of the MPD for subsets of exactly r tips in \mathcal{T} , and $E_{R \in \text{Sub}(S,r)}[\cdot]$ denotes the function of the expectation over all subsets of exactly r tips in S . In a previous paper, we showed how we can compute the expectation and the variance of the MPD on \mathcal{T} in $O(n)$ time [5]. Therefore, in the rest of this work we focus on analysing

the value $E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)]$ and expressing this quantity in a way that can be computed efficiently, in linear time with respect to the size of \mathcal{T} .

To make things more simple, we break the description of our approach into two parts; in the first part, we define several quantities that come from adding and multiplying the costs of specific subsets of paths between tips of the tree. We also present how we can compute all these quantities in $O(n)$ time in total by scanning \mathcal{T} a constant number of times. Then, in Section 3, we show how we can express the skewness of the MPD on \mathcal{T} based on these quantities, and hence compute the skewness in $O(n)$ time as well. Next we provide the quantities that we want to consider in our analysis; these quantities are described in Table 1.

Table 1. The quantities that we use for expressing the skewness of the MPD

I) $\text{TC}(\mathcal{T}) = \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u, v)$	II) $\text{CB}(\mathcal{T}) = \sum_{\{u,v\} \in \Delta(S)} \text{cost}^3(u, v)$
III) $\forall e \in E, \text{TC}(e) = \sum_{\substack{\{u,v\} \in \Delta(S) \\ e \in p(u,v)}} \text{cost}(u, v)$	IV) $\forall e \in E, \text{SQ}(e) = \sum_{\substack{\{u,v\} \in \Delta(S) \\ e \in p(u,v)}} \text{cost}^2(u, v)$
V) $\forall e \in E, \text{Mult}(e) = \sum_{\substack{\{u,v\} \in \Delta(S) \\ e \in p(u,v)}} \text{TC}(u) \cdot \text{TC}(v)$	VI) $\forall u \in S, \text{SM}(u) = \sum_{v \in S \setminus \{u\}} \text{cost}(u, v) \cdot \text{TC}(v)$
VII) $\forall e \in E, \text{TC}_{\text{sub}}(e) = \sum_{u \in S(e)} \text{cost}(u, e)$	VIII) $\forall e \in E, \text{SQ}_{\text{sub}}(e) = \sum_{u \in S(e)} \text{cost}^2(u, e)$
IX) $\forall e \in E, \text{PC}(e) = \sum_{u \in S} \text{cost}(u, e)$	X) $\forall e \in E, \text{PSQ}(e) = \sum_{u \in S} \text{cost}^2(u, e)$
XI) $\forall e \in E, \text{QD}(e) = \sum_{u \in S(e)} \left(\sum_{v \in S(e) \setminus \{u\}} \text{cost}(u, v) \right)^2$	

For any tip $u \in S$, we define that $\text{SQ}(u) = \text{SQ}(e)$, and $\text{TC}(u) = \text{TC}(e)$, where e is the edge whose child node is u . The proof of the following lemma is provided in the full version of this paper.

Lemma 1. *Given a phylogenetic tree \mathcal{T} that consists of n edges, we can compute all the quantities that are presented in Table 1 in $O(n)$ time in total.*

3 Computing the Skewness of the MPD

In the previous section we defined the problem of computing the skewness of the MPD for a given phylogenetic tree \mathcal{T} . Given a positive integer $r \leq s$, we showed that to solve this problem efficiently it remains to find an efficient algorithm for computing $E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)]$; this is the mean value of the cube of the MPD among all possible subsets of tips in \mathcal{T} that consist of exactly r elements. To compute this efficiently, we introduced in Table 1 ten different quantities which we want to use in order to express this mean value. In Lemma 1 we proved that these quantities can be computed in $O(n)$ time, where n is the size of \mathcal{T} .

Next we prove how we can calculate the value for the mean of the cube of the MPD based on the quantities in Table 1. In particular, in the proof of

the following lemma we show how the value $E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)]$ can be written analytically as an expression that contains the quantities in Table 1. This expression can then be straightforwardly evaluated in $O(n)$ time, given that we have already computed the aforementioned quantities ¹.

Lemma 2. *For any given natural $r \leq s$, we can compute $E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)]$ in $\Theta(n)$ time.*

Proof. The expectation of the cube of the MPD is equal to:

$$E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)] = \frac{8}{r^3(r-1)^3} \cdot E_{R \in \text{Sub}(S,r)} \left[\sum_{\{u,v\} \in \Delta(R)} \sum_{\{x,y\} \in \Delta(R)} \sum_{\{c,d\} \in \Delta(R)} \text{cost}(u,v) \cdot \text{cost}(x,y) \cdot \text{cost}(c,d) \right].$$

From the last expression we get:

$$\begin{aligned} & E_{R \in \text{Sub}(S,r)} \left[\sum_{\{u,v\} \in \Delta(R)} \sum_{\{x,y\} \in \Delta(R)} \sum_{\{c,d\} \in \Delta(R)} \text{cost}(u,v) \cdot \text{cost}(x,y) \cdot \text{cost}(c,d) \right] \\ &= \sum_{\{u,v\} \in \Delta(S)} \sum_{\{x,y\} \in \Delta(S)} \sum_{\{c,d\} \in \Delta(S)} \text{cost}(u,v) \cdot \text{cost}(x,y) \cdot \text{cost}(c,d) \cdot \\ & E_{R \in \text{Sub}(S,r)}[AP_R(u,v,x,y,c,d)], \end{aligned} \tag{2}$$

where $AP_R(u,v,x,y,c,d)$ is a random variable whose value is equal to one in the case that $u,v,x,y,c,d \in R$, otherwise it is equal to zero. For any six tips $u,v,x,y,c,d \in S$, which may not be all of them distinct, we use $\theta(u,v,x,y,c,d)$ to denote the number of distinct elements among these tips. Let t be an integer, and let $(t)_k$ denote the k -th falling factorial power of t , which means that $(t)_k = t(t-1) \dots (t-k+1)$. For the expectation of the random variables that appear in the last expression it holds that:

$$E_{R \in \text{Sub}(S,r)} [AP_R(u,v,x,y,c,d)] = \frac{\binom{r}{\theta(u,v,x,y,c,d)}}{\binom{s}{\theta(u,v,x,y,c,d)}} \tag{3}$$

Notice that in (3) we have $2 \leq \theta(u,v,x,y,c,d) \leq 6$. The value of the function $\theta(\cdot)$ cannot be smaller than two in the above case because we have that $u \neq v$, $x \neq y$, and $c \neq d$. Thus, we can rewrite (2) as:

$$\sum_{\{u,v\} \in \Delta(S)} \sum_{\{x,y\} \in \Delta(S)} \sum_{\{c,d\} \in \Delta(S)} \frac{\binom{r}{\theta(u,v,x,y,c,d)}}{\binom{s}{\theta(u,v,x,y,c,d)}} \cdot \text{cost}(u,v) \cdot \text{cost}(x,y) \cdot \text{cost}(c,d) \tag{4}$$

¹ Because the full form of this expression is very long (it consists of a large number of terms), we have chosen not to include it in the definition of the following lemma. We chose to do so because we considered that including the entire expression would not make this work more readable. In any case, the full expression can be easily inferred from the proof of the lemma

Hence, our goal now is to compute a sum whose elements are the product of costs of triples of paths. Recall that for each of these paths, the end-nodes of the path are a pair of distinct tips in the tree. Although the end-nodes of each path are distinct, in a given triple the paths may share one or more end-nodes with each other. Therefore, the distinct tips in any triple of paths may vary from two up to six tips. Indeed, in (4) we get a sum where the triples of paths in the sum are partitioned in five groups; a triple of paths is assigned to a group depending on the number of distinct tips in this triple. In (4) the sum for each group of triples is multiplied by the same factor $(r)^{\theta(u,v,x,y,c,d)}/(s)^{\theta(u,v,x,y,c,d)}$, hence we have to calculate the sum for each group of triples separately.

However, when we try to calculate the sum for each of these groups of triples we see that this calculation is more involved; some of these groups of triples are divided into smaller subgroups, depending on which end-nodes of the paths in each triple are the same. To explain this better, we can represent a triple of paths schematically as a graph; let $\{u, v\}, \{x, y\}, \{c, d\} \in \Delta(S)$ be three pairs of tips in \mathcal{T} . As mentioned already, the tips within each pair are distinct, but tips between different pairs can be the same. We represent the similarity between tips of these three pairs as a graph of six vertices. Each vertex in the graph corresponds to a tip of these three pairs. Also, there exists an edge in this graph between two vertices if the corresponding tips are the same. Thus, this graph is tripartite; no vertices that correspond to tips of the same pair can be connected to each other with an edge. Hence, we have a tripartite graph where each partite set of vertices consists of two vertices—see Fig. 1 for an example.

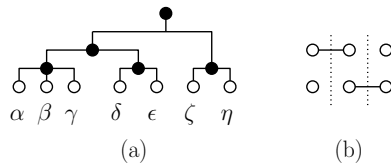


Fig. 1. (a) A phylogenetic tree \mathcal{T} and (b) an example of the tripartite graph induced by the triplet of its tip pairs $\{\alpha, \gamma\}, \{\delta, \gamma\}, \{\epsilon, \delta\}$, where $\{\alpha, \gamma, \delta, \epsilon\} \subset S$. The dashed lines in the graph distinguish the partite subsets of vertices; the vertices of each partite subset correspond to tips of the same pair.

For any triple of pairs of tips $\{u, v\}, \{x, y\}, \{c, d\} \in \Delta(S)$ we denote the tripartite graph that corresponds to this triple by $G[u, v, x, y, c, d]$. We call this graph the *similarity* graph of this triple. Based on the way that similarities may occur between tips in a triple of paths, we can partition the five groups of triples in (4) into smaller subgroups. Each of these subgroups contains triples whose similarity graphs are isomorphic. For a tripartite graph that consists of three partite sets of two vertices each, there can be eight different isomorphism classes. Therefore, the five groups of triples are partitioned into eight subgroups. Figure 2 illustrates the eight isomorphism classes that exist for the specific kind of tripartite graphs that we consider. Since we refer to isomorphism classes, each

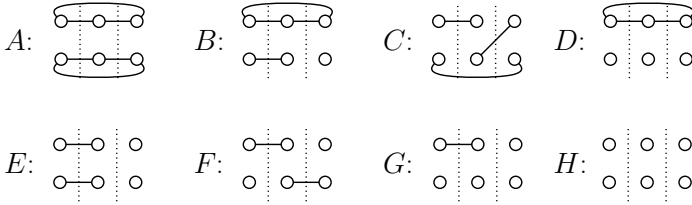


Fig. 2. The eight isomorphism classes of a tripartite graph of 3×2 vertices that represent schematically the eight possible cases of similarities between tips that we can have when we consider three paths between pairs of tips in a tree \mathcal{T}

of the graphs in Fig. 2 represents the combinatorial structure of the similarities between three pairs of tips, and it does not correspond to a particular planar embedding, or ordering of the tips.

Let X be any isomorphism class that is illustrated in Figure 2. We denote the set of all triples of pairs in $\Delta(S)$ whose similarity graphs belong to this class by \mathcal{B}_X . More formally, the set \mathcal{B}_X can be defined as follows :

$$\mathcal{B}_X = \{ \{ \{u, v\}, \{x, y\}, \{c, d\} \} : \{u, v\}, \{x, y\}, \{c, d\} \in \Delta(S) \text{ and } G[u, v, x, y, c, d] \text{ belongs to class } X \text{ in Figure 2} \} .$$

We introduce also the following quantity:

$$\text{TRS}(X) = \sum_{\{ \{u, v\}, \{x, y\}, \{c, d\} \} \in \mathcal{B}_X} \text{cost}(u, v) \cdot \text{cost}(x, y) \cdot \text{cost}(c, d) .$$

Hence, we can rewrite (4) as follows:

$$\begin{aligned} & \frac{\binom{r}{s}_2}{\binom{r}{s}_2} \cdot \text{TRS}(A) + 3 \cdot \frac{\binom{r}{s}_3}{\binom{r}{s}_3} \cdot \text{TRS}(B) + 6 \cdot \frac{\binom{r}{s}_3}{\binom{r}{s}_3} \cdot \text{TRS}(C) + 6 \cdot \frac{\binom{r}{s}_4}{\binom{r}{s}_4} \cdot \text{TRS}(D) \\ & + 3 \cdot \frac{\binom{r}{s}_4}{\binom{r}{s}_4} \cdot \text{TRS}(E) + 6 \cdot \frac{\binom{r}{s}_4}{\binom{r}{s}_4} \cdot \text{TRS}(F) + 6 \cdot \frac{\binom{r}{s}_5}{\binom{r}{s}_5} \cdot \text{TRS}(G) + 6 \cdot \frac{\binom{r}{s}_6}{\binom{r}{s}_6} \cdot \text{TRS}(H) \end{aligned} \quad (5)$$

Notice that some of the terms $\frac{\binom{r}{s}_i}{\binom{r}{s}_i} \cdot \text{TRS}(X)$ in (5) are multiplied with an extra constant factor. This happens for the following reason; the sum in $\text{TRS}(X)$ counts each triple once for every different combination of three pairs of tips. However, in the triple sum in (4) some triples appear more than once. For example, every triple that belongs in class B appears three times in (4), hence there is an extra factor three in front of $\text{TRS}(B)$ in (5).

To compute efficiently $E_{R \in \text{Sub}(S, r)}[\text{MPD}^3(\mathcal{T}, R)]$, it remains to compute efficiently each value $\text{TRS}(X)$ for every isomorphism class X that is presented in Figure 2. Next we show in detail how we can do that by expressing each quantity $\text{TRS}(X)$ as a function of the quantities that appear in Table 1.

For the triples that correspond to the isomorphism class A we have:

$$\text{TRS}(A) = \sum_{\{u, v\} \in \Delta(S)} \text{cost}^3(u, v) = \text{CB}(\mathcal{T}) .$$

For $\text{TRS}(B)$ we get:

$$\begin{aligned}
\text{TRS}(B) &= \sum_{\{u,v\} \in \Delta(S)} \text{cost}^2(u,v) \left(\sum_{x \in S \setminus \{u\}} \text{cost}(u,x) + \sum_{y \in S \setminus \{v\}} \text{cost}(v,y) - 2 \cdot \text{cost}(u,v) \right) \\
&= \sum_{\{u,v\} \in \Delta(S)} \text{cost}^2(u,v) (\text{TC}(u) + \text{TC}(v) - 2 \cdot \text{cost}(u,v)) \\
&= \sum_{u \in S} \text{SQ}(u) \cdot \text{TC}(u) - 2 \cdot \text{CB}(\mathcal{T}) .
\end{aligned}$$

The quantity $\text{TRS}(C)$ is equal to:

$$\begin{aligned}
&\frac{1}{6} \sum_{u \in S} \sum_{v \in S \setminus \{u\}} \text{cost}(u,v) \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \\
&= \frac{1}{6} \sum_{e \in E} w_e \sum_{u \in S(e)} \sum_{v \in S - S(e)} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) . \tag{6}
\end{aligned}$$

For any $e \in E$ we have that:

$$\begin{aligned}
&\sum_{u \in S(e)} \sum_{v \in S - S(e)} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \\
&= \sum_{u \in S(e)} \sum_{v \in S \setminus \{u\}} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \tag{7} \\
&- 2 \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) . \tag{7b}
\end{aligned}$$

The first of the two sums in (7) can be written as:

$$\begin{aligned}
&\sum_{u \in S(e)} \sum_{v \in S \setminus \{u\}} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \\
&= \sum_{u \in S(e)} \sum_{v \in S \setminus \{u\}} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,v) \cdot \text{cost}(v,x) \\
&= \sum_{u \in S(e)} \sum_{v \in S \setminus \{u\}} (\text{cost}(u,v) \cdot \text{TC}(v) - \text{cost}^2(u,v)) \\
&= \sum_{u \in S(e)} \text{SM}(u) - \text{SQ}(u) . \tag{8}
\end{aligned}$$

According to Lemma 2, we can compute $\text{SM}(u)$ and $\text{SQ}(u)$ for all tips $u \in S$ in linear time with respect to the size of \mathcal{T} . Given these values, we can compute $\sum_{u \in S(e)} \text{SM}(u) - \text{SQ}(u)$ for every edge $e \in E$ in \mathcal{T} with a single bottom-up scan of the tree. For any edge e in E , the second sum in (7b) is equal to:

$$\begin{aligned} & \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \\ &= \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S(e) \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \end{aligned} \tag{9}$$

$$+ \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}(u,x) \cdot \text{cost}(x,v) . \tag{9b}$$

We can express the first sum in (9) as:

$$\begin{aligned} & \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S(e) \setminus \{u,v\}} \text{cost}(u,x) \cdot \text{cost}(x,v) \\ &= \frac{1}{2} \sum_{u \in S(e)} \left(\sum_{v \in S(e) \setminus \{u\}} \text{cost}(u,v) \right)^2 - \frac{1}{2} \sum_{u \in S(e)} \sum_{v \in S(e) \setminus \{u\}} \text{cost}^2(u,v) \\ &= \frac{1}{2} \text{QD}(e) - \frac{1}{2} \sum_{u \in S(e)} \sum_{v \in S(e) \setminus \{u\}} \text{cost}^2(u,v) . \end{aligned} \tag{10}$$

The last sum in (10) is equal to:

$$\sum_{u \in S(e)} \sum_{v \in S(e) \setminus \{u\}} \text{cost}^2(u,v) = \sum_{u \in S(e)} \text{SQ}(u) - \text{SQ}(e) . . \tag{11}$$

The value of the sum $\sum_{u \in S(e)} \text{SQ}(u)$ can be computed for every edge e in $\Theta(n)$ time in total as follows; for every tip $u \in S$ we store $\text{SQ}(u)$ together with this tip, and then scan bottom-up the tree adding those values that are in the subtree of each edge. For the remaining part of (9b) we get:

$$\begin{aligned} & \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}(u,x) \cdot \text{cost}(x,v) \\ &= \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} (\text{cost}(u,e) + \text{cost}(x,e)) (\text{cost}(v,e) + \text{cost}(x,e)) \\ &= \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}(u,e) \cdot \text{cost}(v,e) \\ &+ \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}(x,e) \cdot (\text{cost}(u,e) + \text{cost}(v,e)) \\ &+ \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}^2(x,e) . \end{aligned} \tag{12}$$

The first sum in (12) is equal to:

$$\sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}(u, e) \cdot \text{cost}(v, e) = (s - s(e)) (\text{TC}_{\text{sub}}^2(e) - \text{SQ}_{\text{sub}}(e)) . \quad (13)$$

For the second sum in (12) we have:

$$\begin{aligned} & \sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}(x, e) \cdot (\text{cost}(u, e) + \text{cost}(v, e)) \\ &= (s(e) - 1) \sum_{x \in S \setminus S(e)} \text{cost}(x, e) \cdot \text{TC}_{\text{sub}}(e) = (s(e) - 1) \cdot \text{TC}_{\text{sub}}(e) \cdot (\text{PC}(e) - \text{TC}_{\text{sub}}(e)) . \end{aligned} \quad (14)$$

The last sum in (12) can be written as:

$$\sum_{\{u,v\} \in \Delta(S(e))} \sum_{x \in S \setminus S(e)} \text{cost}^2(x, e) = \frac{s(e)(s(e) - 1)}{2} (\text{PSQ}(e) - \text{SQ}_{\text{sub}}(e)) . \quad (15)$$

Combining the analyses that we did from (6) up to (15) we get:

$$\begin{aligned} \text{TRS}(C) &= \frac{1}{6} \sum_{e \in E} w_e \sum_{u \in S(e)} \left(\text{SM}(u) - \frac{3}{2} \text{SQ}(u) \right) + \frac{1}{2} \cdot \text{QD}(e) + \frac{1}{2} \cdot \text{SQ}(e) \\ &+ (s - 2s(e) + 1) \cdot \text{TC}_{\text{sub}}^2(e) - \frac{2s - 2 \cdot s(e) + s(e)(s(e) - 1)}{2} \cdot \text{SQ}_{\text{sub}}(e) \\ &+ (s(e) - 1) \cdot \text{TC}_{\text{sub}}(e) \cdot \text{PC}(e) + \frac{s(e)(s(e) - 1)}{2} \cdot \text{PSQ}(e) . \end{aligned}$$

The value of $\text{TRS}(D)$ can be expressed as:

$$\begin{aligned} & \sum_{u \in S} \sum_{\substack{v, x, y \in S \setminus \{u\} \\ v, x, y \text{ are distinct}}} \text{cost}(u, v) \cdot \text{cost}(u, x) \cdot \text{cost}(u, y) \\ &= \frac{1}{6} \left(\sum_{u \in S} \text{TC}^3(u) - 2 \cdot \text{TRS}(A) - 3 \cdot \text{TRS}(B) \right) \\ &= \frac{1}{6} \cdot \sum_{u \in S} \text{TC}^3(u) + \frac{2}{3} \cdot \text{CB}(\mathcal{T}) - \frac{1}{2} \cdot \text{SQ}(u) \cdot \text{TC}(u) . \end{aligned}$$

For $\text{TRS}(E)$ we get:

$$\begin{aligned}
 & \sum_{\{u,v\} \in \Delta(S)} \sum_{\{x,y\} \in \Delta(S \setminus \{u,v\})} cost^2(u,v) \cdot cost(x,y) \\
 &= \sum_{\{u,v\} \in \Delta(S)} cost^2(u,v) (\text{TC}(\mathcal{T}) - \text{TC}(u) - \text{TC}(v) + cost(u,v)) \\
 &= \text{TC}(\mathcal{T}) \sum_{e \in E} w_e \cdot \text{TC}(e) - \sum_{u \in S} (\text{SQ}(u) \cdot \text{TC}(u)) + \text{CB}(\mathcal{T}) .
 \end{aligned}$$

We can rewrite $\text{TRS}(F)$ as follows:

$$\begin{aligned}
 & \sum_{\{u,v\} \in \Delta(S)} cost(u,v) \left(\text{TC}(u) \cdot \text{TC}(v) - cost^2(u,v) - \sum_{x \in S \setminus \{u,v\}} cost(u,x) \cdot cost(x,v) \right) \\
 &= \sum_{\{u,v\} \in \Delta(S)} cost(u,v) \cdot \text{TC}(u) \cdot \text{TC}(v) - \text{CB}(\mathcal{T}) - 3 \cdot \text{TRS}(C) \\
 &= \sum_{e \in E} w_e \cdot \text{Mult}(e) - \text{CB}(\mathcal{T}) - 3 \cdot \text{TRS}(C) .
 \end{aligned}$$

For the value of $\text{TRS}(G)$ we have:

$$\begin{aligned}
 \text{TRS}(G) &= \frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} cost(u,v) \sum_{x \in S \setminus \{u,v\}} (cost(u,x) + cost(v,x)) \left(\text{TC}(\mathcal{T}) \right. \\
 & \quad \left. - \text{TC}(u) - \text{TC}(v) - \text{TC}(x) + cost(u,v) + cost(u,x) + cost(v,x) \right) .
 \end{aligned} \tag{16}$$

We now break the sum in (16) into five pieces and express each piece of this sum in terms of the quantities in Table 1. The first piece of the sum is equal to:

$$\begin{aligned}
 & \frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} cost(u,v) \sum_{x \in S \setminus \{u,v\}} (cost(u,x) + cost(v,x)) \cdot \text{TC}(\mathcal{T}) \\
 &= \frac{1}{2} \cdot \text{TC}(\mathcal{T}) \sum_{u \in S} \text{TC}^2(u) - \sum_{\{u,v\} \in \Delta(S)} cost^2(u,v) \\
 &= \frac{1}{2} \cdot \text{TC}(\mathcal{T}) \sum_{u \in S} \text{TC}^2(u) - \sum_{e \in E} w_e \cdot \text{TC}(e) .
 \end{aligned}$$

The second piece that we take from the sum in (16) can be expressed as:

$$\begin{aligned}
& -\frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) \sum_{x \in S \setminus \{u,v\}} (\text{cost}(u,x) + \text{cost}(v,x)) (\text{TC}(u) + \text{TC}(v)) \\
&= -\frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) (\text{TC}(u) + \text{TC}(v) - 2 \cdot \text{cost}(u,v)) (\text{TC}(u) + \text{TC}(v)) \\
&= -\frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) \left(\text{TC}^2(u) + \text{TC}^2(v) + 2 \cdot \text{TC}(u) \cdot \text{TC}(v) \right. \\
&\quad \left. - 2 \cdot \text{cost}(u,v) \cdot (\text{TC}(u) + \text{TC}(v)) \right) \\
&= -\frac{1}{2} \sum_{u \in S} \text{TC}^3(u) - \sum_{\{v,x\} \in \Delta(S)} \text{cost}(v,x) \cdot \text{TC}(v) \cdot \text{TC}(x) \\
&\quad + \sum_{\{y,z\} \in \Delta(S)} \text{cost}^2(y,z) (\text{TC}(y) + \text{TC}(z)) \\
&= -\frac{1}{2} \sum_{u \in S} \text{TC}^3(u) - \sum_{e \in E} w_e \cdot \text{Mult}(e) + \sum_{u \in S} \text{SQ}(u) \cdot \text{TC}(u) . \tag{17}
\end{aligned}$$

The next piece that we select from (16) is equal to:

$$\begin{aligned}
& -\frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) \sum_{x \in S \setminus \{u,v\}} (\text{cost}(u,x) + \text{cost}(v,x)) \cdot \text{TC}(x) \\
&= -\frac{1}{2} \sum_{u \in S} \sum_{v \in S \setminus \{u\}} \text{cost}(u,v) \sum_{x \in S \setminus \{u\}} \text{cost}(u,x) \cdot \text{TC}(x) \\
&\quad + \frac{1}{2} \sum_{u \in S} \sum_{v \in S \setminus \{u\}} \text{cost}^2(u,v) \cdot \text{TC}(v) \\
&= -\frac{1}{2} \sum_{u \in S} \text{TC}(u) \cdot \text{SM}(u) + \frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}^2(u,v) (\text{TC}(u) + \text{TC}(v)) \\
&= \frac{1}{2} \sum_{u \in S} \text{TC}(u) (\text{SQ}(u) - \text{SM}(u)) . \tag{18}
\end{aligned}$$

For the fourth piece of the sum in (16) we get:

$$\frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}^2(u,v) \sum_{x \in S \setminus \{u,v\}} \text{cost}(u,x) + \text{cost}(v,x) \tag{19}$$

$$= \frac{1}{2} \cdot \text{TRS}(B) = \frac{1}{2} \sum_{u \in S} \text{SQ}(u) \cdot \text{TC}(u) - \text{CB}(\mathcal{T}) . \tag{20}$$

The last piece of the sum in (16) can be expressed as:

$$\begin{aligned}
 & \frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) \sum_{x \in S \setminus \{u,v\}} (\text{cost}(u,x) + \text{cost}(v,x))^2 \\
 &= \frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) \sum_{x \in S \setminus \{u,v\}} (\text{cost}^2(u,x) + \text{cost}^2(v,x)) + 3 \cdot \text{TRS}(C) \\
 &= \frac{1}{2} \sum_{\{u,v\} \in \Delta(S)} \text{cost}(u,v) (\text{SQ}(u) + \text{SQ}(v) - 2 \cdot \text{cost}^2(u,v)) + 3 \cdot \text{TRS}(C) \\
 &= \frac{1}{2} \sum_{u \in S} \text{SQ}(u) \cdot \text{TC}(u) - \text{CB}(\mathcal{T}) + 3 \cdot \text{TRS}(C) . \tag{21}
 \end{aligned}$$

Combining our analyses from (16) up to (21) we get:

$$\begin{aligned}
 \text{TRS}(G) &= \frac{1}{2} \cdot \text{TC}(\mathcal{T}) \sum_{u \in S} \text{TC}^2(u) - \sum_{e \in E} w_e (\text{TC}(e) + \text{Mult}(e)) \\
 &+ \frac{1}{2} \sum_{u \in S} \text{TC}(u) \cdot (5 \cdot \text{SQ}(u) - \text{SM}(u) - \text{TC}^2(u)) - 2 \cdot \text{CB}(T) + 3 \cdot \text{TRS}(C) .
 \end{aligned}$$

We can express $\text{TRS}(H)$ using the values of the other isomorphism classes:

$$\begin{aligned}
 \text{TRS}(H) &= \frac{1}{6} \sum_{\{u,v\} \in \Delta(S)} \sum_{\{x,y\} \in \Delta(S)} \sum_{\{c,d\} \in \Delta(S)} \text{cost}(u,v) \cdot \text{cost}(x,y) \cdot \text{cost}(c,d) \\
 &- \text{TRS}(A) - 3 \cdot \text{TRS}(B) - 6 \cdot \text{TRS}(C) - 6 \cdot \text{TRS}(D) \\
 &- 3 \cdot \text{TRS}(E) - 6 \cdot \text{TRS}(F) - 6 \cdot \text{TRS}(G) \\
 &= \frac{1}{6} \cdot \text{TC}^3(\mathcal{T}) - \frac{1}{6} \cdot \text{TRS}(A) - \frac{1}{2} \cdot \text{TRS}(B) - \text{TRS}(C) - \text{TRS}(D) \\
 &- \frac{1}{2} \cdot \text{TRS}(E) - \text{TRS}(F) - \text{TRS}(G) .
 \end{aligned}$$

We get the value of $E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)]$ by plugging into (5) the values that we got for all eight isomorphism classes of triples. For any isomorphism class X we showed that the value $\text{TRS}(X)$ can be computed by using the quantities in Table 1. The lemma follows from the fact that each quantity that appears in this table is used a constant number of times for computing value $\text{TRS}(X)$ for any class X , and since we showed that we can precompute all these quantities in $\Theta(n)$ time in total. \square

Theorem 1. *Let \mathcal{T} be a phylogenetic tree that contains s tips, and let r be a natural number with $r \leq s$. The skewness of the mean pairwise distance on \mathcal{T} among all subsets of exactly r tips of \mathcal{T} can be computed in $\Theta(n)$ time.*

Proof. According to the definition of skewness, as it is also presented in (1), we need to prove that we can compute in $\Theta(n)$ time the expectation and the variance of the MPD, and the value of the expression $E_{R \in \text{Sub}(S,r)}[\text{MPD}^3(\mathcal{T}, R)]$. In a previous paper we showed that the expectation and the variance of the MPD can be computed in $\Theta(n)$ time. By combining this with Lemma 2 we get the proof of the theorem. \square

References

1. Cooper, N., Rodríguez, J., Purvis, A.: A Common Tendency for Phylogenetic Overdispersion in Mammalian Assemblages. In: Proceedings of the Royal Society B: Biological Sciences, vol. 275, pp. 2031–2037 (2008)
2. Harmon-Threatt, A.N., Ackerly, D.D.: Filtering Across Spatial Scales: Phylogeny, Biogeography and Community Structure in Bumble Bees. PLoS ONE 8, e60446 (2013)
3. Pontarp, M., Canbäck, B., Tunlid, A., Lundberg, P.: Phylogenetic Analysis Suggests that Habitat Filtering Is Structuring Marine Bacterial Communities Across the Globe. Microbial Ecology 64, 8–17 (2012)
4. Swenson, N.G.: Phylogenetic Beta Diversity Metrics. Trait Evolution and Inferring the Functional Beta Diversity of Communities. PLoS ONE 6(6), e21264 (2011)
5. Tsirogiannis, C., Sandel, B., Cheliotis, D.: Efficient Computation of Popular Phylogenetic Tree Measures. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS, vol. 7534, pp. 30–43. Springer, Heidelberg (2012)
6. Vamosi, J.C., Vamosi, S.M.: Body Size, Rarity, and Phylogenetic Community Structure: Insights from Diving Beetle Assemblages of Alberta. Diversity and Distributions 13, 1–10 (2007)
7. Webb, C.O., Ackerly, D.D., McPeck, M.A., Donoghue, M.J.: Phylogenies and Community Ecology. Annual Review of Ecology and Systematics 33, 475–505 (2002)

Characterizing Compatibility and Agreement of Unrooted Trees via Cuts in Graphs^{*}

Sudheer Vakati and David Fernández-Baca

Department of Computer Science, Iowa State University, Ames, IA 50011, USA
{svakati, fernande}@iastate.edu

Abstract. Deciding whether there is a single tree—a supertree—that summarizes the evolutionary information in a collection of unrooted trees is a fundamental problem in phylogenetics. We consider two versions of this question: agreement and compatibility. In the first, the supertree is required to reflect precisely the relationships among the species exhibited by the input trees. In the second, the supertree can be more refined than the input trees.

Tree compatibility can be characterized in terms of the existence of a specific kind of triangulation in a structure known as the display graph. Alternatively, it can be characterized as a chordal graph sandwich problem in a structure known as the edge label intersection graph. Here, we show that the latter characterization yields a natural characterization of compatibility in terms of minimal cuts in the display graph, which is closely related to compatibility of splits. We then derive a characterization for agreement.

1 Introduction

A *phylogenetic tree* T is an unrooted tree whose leaves are bijectively mapped to a label set $\mathcal{L}(T)$. Labels represent species and T represents the evolutionary history of these species. Let \mathcal{P} be a collection of phylogenetic trees. We call \mathcal{P} a *profile*, refer to the trees in \mathcal{P} as *input trees*, and denote the combined label set of the input trees, $\bigcup_{T \in \mathcal{P}} \mathcal{L}(T)$, by $\mathcal{L}(\mathcal{P})$. A *supertree* of \mathcal{P} is a phylogenetic tree whose label set is $\mathcal{L}(\mathcal{P})$. The goal of constructing a supertree for a profile is to synthesize the information in the input trees in a larger, more comprehensive, phylogeny [7]. Ideally, a supertree should faithfully reflect the relationships among the species implied by the input trees. In reality, it is rarely possible to achieve this, because of conflicts among the input trees due to errors in constructing them or to biological processes such as lateral gene transfer and gene duplication.

We consider two classic versions of the supertree problem, based on the closely related notions of compatibility and agreement. Let S and T be two phylogenetic trees where $\mathcal{L}(T) \subseteq \mathcal{L}(S)$ —for our purposes, T would be an input tree and S a supertree. Let S' be the tree obtained by suppressing any degree two vertices in the minimal subtree of S connecting the labels in $\mathcal{L}(T)$. We say that S *displays* T , or that T and S are *compatible*, if T can be derived from S' by contracting edges. We say that tree T is an *induced subtree* of S , or that T and S *agree*, if S' is isomorphic to T .

^{*} This work was supported in part by the National Science Foundation under grants CCF-1017189 and DEB-0829674.

Let \mathcal{P} be a profile. The *tree compatibility problem* asks if there exists a supertree for \mathcal{P} that displays all the trees in \mathcal{P} . If such a supertree S exists, we say that \mathcal{P} is *compatible* and S is a *compatible supertree* for \mathcal{P} . The *agreement supertree problem* asks if there exists a supertree for \mathcal{P} that agrees with all the trees in \mathcal{P} . If such a supertree S exists, we say that S is an *agreement supertree* (AST) for \mathcal{P} .

Compatibility and agreement embody different philosophies about conflict. An agreement supertree must reflect precisely the evolutionary relationships exhibited by the input trees. In contrast, a compatible supertree is allowed to exhibit more fine-grained relationships among certain labels than those exhibited by an input tree. Note that compatibility and agreement are equivalent when the input trees are binary.

If all the input trees share a common label (which can be viewed as a root node), both tree compatibility and agreement are solvable in polynomial time [1,11]. In general, however, the two problems are NP-complete, and remain so even when the trees are quartets; i.e., binary trees with exactly four leaves [14]. Nevertheless, Bryant and Lagergren showed that the tree compatibility problem is fixed parameter tractable when parametrized by number of trees [4]. It is unknown whether or not the agreement supertree problem has the same property.

To prove the fixed-parameter tractability of tree compatibility, Bryant and Lagergren first showed that a necessary (but not sufficient) condition for a profile to be compatible is that the tree-width of a certain graph—the *display graph* of the profile (see Section 3)—be bounded by the number of trees. They then showed how to express compatibility as a bounded-size monadic second-order formula on the display graph. By Courcelle’s Theorem [6,2], these two facts imply that compatibility can be decided in time linear in the size of the display graph. Unfortunately, Bryant and Lagergren’s argument amounts essentially to only an existential proof, as it is not clear how to obtain an explicit algorithm for unrooted compatibility from it.

A necessary step towards finding a practical algorithm for compatibility—and indeed for agreement—is to develop an explicit characterization of the problem. In earlier work [15], we made some progress in this direction, characterizing tree compatibility in terms of the existence of a legal triangulation of the display graph of the profile. Gysel et al. [9] provided an alternative characterization, based on a structure they call the edge label intersection graph (ELIG) (see Section 3). Their formulation is in some ways simpler than that of [15], allowing Gysel et al. to express tree compatibility as a chordal sandwich problem. Neither [15] nor [9] deal with agreement.

Here, we show that the connection between separators in the ELIG and cuts in the display graph (explored in Section 3) leads to a new, and natural, characterization of compatibility in terms of minimal cuts in the display graph (Section 4). We then show how such cuts are closely related to the splits of the compatible supertree (Section 5). Lastly, we give a characterization of the agreement in terms of minimal cuts of the display graph (Section 6). To our knowledge, there was no previous characterization of the agreement supertree problem for unrooted trees.

2 Preliminaries

Splits, Compatibility, and Agreement A *split* of a label set L is a bipartition of L consisting of non-empty sets. We denote a split $\{X, Y\}$ by $X|Y$. Let T be a phylogenetic

tree. Consider an internal edge e of T . Deletion of e disconnects T into two subtrees T_1 and T_2 . If L_1 and L_2 denote the set of all labels in T_1 and T_2 , respectively, then $L_1|L_2$ is a split of $\mathcal{L}(T)$. We denote by $\sigma_e(T)$ the split corresponding to edge e of T and by $\Sigma(T)$ the set of all splits corresponding to all internal edges of T .

We say that a tree T displays a split $X|Y$ if there exists an internal edge e of T where $\sigma_e(T) = X|Y$. A set of splits is compatible if there exists a tree that displays all the splits in the set. It is well-known that two splits $A_1|A_2$ and $B_1|B_2$ are compatible if and only if at least one of $A_1 \cap B_1$, $A_1 \cap B_2$, $A_2 \cap B_1$ and $A_2 \cap B_2$ is empty [13].

Theorem 1 (Splits-Equivalence Theorem [5,13]). *Let Σ be a collection of non-trivial splits of a label set X . Then, $\Sigma = \Sigma(T)$ for some phylogenetic tree T with label set X if and only if the splits in Σ are pairwise compatible. Tree T is unique up to isomorphism.*

Let S be a phylogenetic tree and let Y be a subset of $\mathcal{L}(S)$. Then, $S_{|Y}$ denotes the tree obtained by suppressing any degree-two vertices in the minimal subtree of S connecting the labels in Y . Now, let T be a phylogenetic tree such that $\mathcal{L}(T) \subseteq \mathcal{L}(S)$. Then, S displays T if and only if $\Sigma(T) \subseteq \Sigma(S_{|\mathcal{L}(T)})$; T and S agree if and only if $\Sigma(T) = \Sigma(S_{|\mathcal{L}(T)})$.

Cliques, Separators, Cuts, and Triangulations. Let G be a graph. We represent the vertices and edges of G by $V(G)$ and $E(G)$ respectively. A clique of G is a complete subgraph of G . A clique H of G is maximal if there is no other clique H' of G where $V(H) \subset V(H')$. For any $U \subseteq V(G)$, $G - U$ is the graph derived by removing vertices of U and their incident edges from G . For any $F \subseteq E(G)$, $G - F$ is the graph with vertex set $V(G)$ and edge set $E(G) \setminus F$.

For any two nonadjacent vertices a and b of G , an a - b separator of G is a set U of vertices where $U \subset V(G)$ and a and b are in different connected components of $G - U$. An a - b separator U is minimal if for every $U' \subset U$, U' is not an a - b separator. A set $U \subseteq V(G)$ is a minimal separator if U is a minimal a - b separator for some nonadjacent vertices a and b of G . We represent the set of all minimal separators of graph G by Δ_G . Two minimal separators U and U' are parallel if $G - U$ contains at most one component H where $V(H) \cap U' \neq \emptyset$.

A connected component H of $G - U$ is full if for every $u \in U$ there exists some vertex $v \in H$ where $\{u, v\} \in E(G)$.

Lemma 1 ([12]). *For a graph G and any $U \subset V(G)$, U is a minimal separator of G if and only if $G - U$ has at least two full components.*

A chord is an edge between two nonadjacent vertices of a cycle. A graph H is chordal if and only if every cycle of length four or greater in H has a chord. A chordal graph H is a triangulation of graph G if $V(G) = V(H)$ and $E(G) \subseteq E(H)$. The edges in $E(H) \setminus E(G)$ are called fill-in edges of G . A triangulation is minimal if removing any fill-in edge yields a non-chordal graph.

A clique tree of a chordal graph H is a pair (T, B) where (i) T is a tree, (ii) B is a bijective function from vertices of T to maximal cliques of H , and (iii) for every vertex $v \in H$, the set of all vertices x of T where $v \in B(x)$ induces a subtree in T . Property (iii) is called coherence.

Let \mathcal{F} be a collection of subsets of $V(G)$. We represent by $G_{\mathcal{F}}$ the graph derived from G by making the set of vertices of X a clique in G for every $X \in \mathcal{F}$. The next result summarizes basic facts about separators and triangulations (see [3,10,12]).

Theorem 2. *Let \mathcal{F} be a maximal set of pairwise parallel minimal separators of G and H be a minimal triangulation of G . Then, the following statements hold.*

- (i) $G_{\mathcal{F}}$ is a minimal triangulation of G .
- (ii) Let (T, B) be a clique tree of $G_{\mathcal{F}}$. There exists a minimal separator $F \in \mathcal{F}$ if and only if there exist two adjacent vertices x and y in T where $B(x) \cap B(y) = F$.
- (iii) Δ_H is a maximal set of pairwise parallel minimal separators of G and $G_{\Delta_H} = H$.

A *cut* in a connected graph G is a subset F of edges of G such that $G - F$ is disconnected. A cut F is *minimal* if there does not exist $F' \subset F$ where $G - F'$ is disconnected. Note that if F is minimal, $G - F$ has exactly two connected components. Two minimal cuts F and F' are *parallel* if $G - F$ has at most one connected component H where $E(H) \cap F' \neq \emptyset$.

3 Display Graphs and Edge Label Intersection Graphs

We now introduce the two main notions that we use to characterize compatibility and agreement: the display graph and edge label intersection graph. We then present some known results about these graphs, along with new results on the relationships between them. Here and in the rest of the paper, $[m]$ denotes the set $\{1, \dots, m\}$, where m is a non-negative integer. Since for any phylogenetic tree T there is a bijection between the leaves of T and $\mathcal{L}(T)$, we refer to the leaves of T by their labels.

Let $\mathcal{P} = \{T_1, T_2, \dots, T_k\}$ be a profile. We assume that for any $i, j \in [k]$ such that $i \neq j$, the sets of internal vertices of input trees T_i and T_j are disjoint. The *display graph* of \mathcal{P} , denoted by $G(\mathcal{P})$, is a graph whose vertex set is $\bigcup_{i \in [k]} V(T_i)$ and edge set is $\bigcup_{j \in [k]} E(T_j)$ (see Fig. 1). A vertex v of $G(\mathcal{P})$ is a *leaf* if $v \in \mathcal{L}(\mathcal{P})$. Every other vertex of $G(\mathcal{P})$ is an *internal*. An edge of $G(\mathcal{P})$ is *internal* if its endpoints are both internal. If H is a subgraph of $G(\mathcal{P})$, then $\mathcal{L}(H)$ represents the set of all leaves of H .

A triangulation G' of $G(\mathcal{P})$ is *legal* if it satisfies the following conditions.

- (LT1) For every clique C of G' , if C contains an internal edge, then it cannot contain any other edge of $G(\mathcal{P})$.
- (LT2) There is no fill-in edge in G' with a leaf as an endpoint.

Theorem 3 (Vakati, Fernández-Baca [15]). *A profile \mathcal{P} of unrooted phylogenetic trees is compatible if and only if $G(\mathcal{P})$ has a legal triangulation.*

In what follows, we assume that $G(\mathcal{P})$ is connected. If it is not, the connected components of $G(\mathcal{P})$ induce a partition of \mathcal{P} into sub-profiles such that for each sub-profile \mathcal{P}' , $G(\mathcal{P}')$ is a connected component of $G(\mathcal{P})$. It is easy to see that \mathcal{P} is compatible if and only if each sub-profile is compatible.

The *edge label intersection graph* of \mathcal{P} , denoted $\text{LG}(\mathcal{P})$, is the line graph of $G(\mathcal{P})$ [9].¹ That is, the vertex set of $\text{LG}(\mathcal{P})$ is $E(G(\mathcal{P}))$ and two vertices of $\text{LG}(\mathcal{P})$ are

¹ Note that Gysel et al. refer to $\text{LG}(\mathcal{P})$ as the modified edge label intersection graph [9].

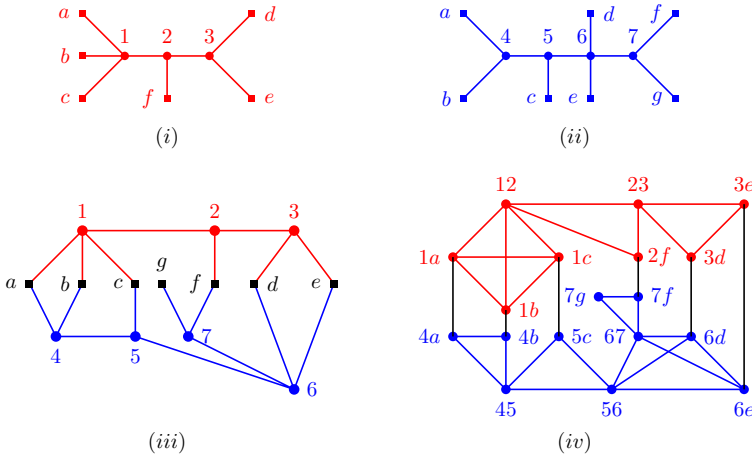


Fig. 1. (i) First input tree. (ii) A second input tree, compatible with the first. (iii) Display graph of the input trees. (iv) Edge label intersection graph of the input trees; for every vertex, uv represents edge $\{u, v\}$.

adjacent if the corresponding edges in $G(\mathcal{P})$ share an endpoint. For an unrooted tree T , $LG(T)$ denotes $LG(\{T\})$.

Observation 1. Let F be a set of edges of $G(\mathcal{P})$ and let $\{v_1, v_2, \dots, v_m\} \subseteq V(G(\mathcal{P}))$ where $m \geq 2$. Then, v_1, v_2, \dots, v_m is a path in $G(\mathcal{P}) - F$ if and only if $\{v_1, v_2\}, \dots, \{v_{m-1}, v_m\}$ is a path in $LG(\mathcal{P}) - F$.

Thus, if $G(\mathcal{P})$ is connected, so is $LG(\mathcal{P})$. Hence, in what follows, we assume that $LG(\mathcal{P})$ is connected.

A fill-in edge for $LG(\mathcal{P})$ is *valid* if for every $T \in \mathcal{P}$, at least one of the endpoints of the edge is not in $LG(T)$. A triangulation H of $LG(\mathcal{P})$ is *restricted* if every fill-in edge of H is valid.

Theorem 4 (Gysel, Stevens, and Gusfield [9]). A profile \mathcal{P} of unrooted phylogenetic trees is compatible if and only if $LG(\mathcal{P})$ has a restricted triangulation.

A minimal separator F of $LG(\mathcal{P})$ is *legal* if for every $T \in \mathcal{P}$, all the edges of T in F share a common endpoint; i.e., $F \cap E(T)$ is a clique in $LG(T)$. The following theorem was mentioned in [9].

Theorem 5. A profile \mathcal{P} is compatible if and only if there exists a maximal set \mathcal{F} of pairwise parallel minimal separators in $LG(\mathcal{P})$ where every separator in \mathcal{F} is legal.

Proof. Our approach is similar to the one used by Gusfield in [8]. Assume that \mathcal{P} is compatible. From Theorem 4, there exists a restricted triangulation H of $LG(\mathcal{P})$. We can assume that H is minimal (if it is not, simply delete fill-in edges repeatedly from H until it is minimal). Let $\mathcal{F} = \Delta_H$. From Theorem 2, \mathcal{F} is a maximal set of pairwise

parallel minimal separators of $\text{LG}(\mathcal{P})$ and $\text{LG}(\mathcal{P})_{\mathcal{F}} = H$. Suppose \mathcal{F} contains a separator F that is not legal. Let $\{e, e'\} \subseteq F$ where $\{e, e'\} \subseteq E(T)$ for some input tree T and $e \cap e' = \emptyset$. The vertices of F form a clique in H . Thus, H contains the edge $\{e, e'\}$. Since $\{e, e'\}$ is not a valid edge, H is not a restricted triangulation, a contradiction. Hence, every separator in \mathcal{F} is legal.

Let \mathcal{F} be a maximal set of pairwise parallel minimal separators of $\text{LG}(\mathcal{P})$ where every separator in \mathcal{F} is legal. From Theorem 2, $\text{LG}(\mathcal{P})_{\mathcal{F}}$ is a minimal triangulation of $\text{LG}(\mathcal{F})$. If $\{e, e'\} \in E(\text{LG}(\mathcal{P})_{\mathcal{F}})$ is a fill-in edge, then $e \cap e' = \emptyset$ and there exists a minimal separator $F \in \mathcal{F}$ where $\{e, e'\} \subseteq F$. Since F is legal, if $\{e, e'\} \subseteq E(T)$ for some input tree T then $e \cap e' \neq \emptyset$. Thus, e and e' are not both from $\text{LG}(T)$ for any input tree T . Hence, every fill-in edge in $\text{LG}(\mathcal{P})_{\mathcal{F}}$ is valid, and $\text{LG}(\mathcal{P})_{\mathcal{F}}$ is a restricted triangulation. \square

Let u be a vertex of some input tree, Then, $\text{Inc}(u)$ is the set of all edges of $G(\mathcal{P})$ incident on u . Equivalently, $\text{Inc}(u)$ is the set of all vertices e of $\text{LG}(\mathcal{P})$ such that $u \in e$.

Let F be a cut of the display graph $G(\mathcal{P})$. F is *legal* if for every tree $T \in \mathcal{P}$, the edges of T in F are incident on a common vertex; i.e., if $F \cap E(T) \subseteq \text{Inc}(u)$ for some $u \in V(T)$. F is *nice* if F is legal and each connected component of $G(\mathcal{P}) - F$ has at least one edge.

Lemma 2. *Let F be a subset of $E(G(\mathcal{P}))$. Then, F is a legal minimal separator of $\text{LG}(\mathcal{P})$ if and only if F is a nice minimal cut of $G(\mathcal{P})$.*

To prove the Lemma 2, we need two auxiliary lemmas and a corollary.

Lemma 3. *Let F be any minimal separator of $\text{LG}(\mathcal{P})$ and u be any vertex of any input tree. Then, $\text{Inc}(u) \not\subseteq F$.*

Proof. Suppose F is a minimal a - b separator of $\text{LG}(\mathcal{P})$ and u is a vertex of some input tree such that $\text{Inc}(u) \subseteq F$. Consider any vertex $e \in \text{Inc}(u)$. Then, there exists a path π from a to b in $\text{LG}(\mathcal{P})$ where e is the only vertex of F in π . If such a path π did not exist, then $F - e$ would still be a a - b separator, and F would not be minimal, a contradiction. Let e_1 and e_2 be the neighbors of e in π and let $e = \{u, v\}$. Since $\text{Inc}(u) \subseteq F$, π does not contain any other vertex e' where $u \in e'$. Thus, $e \cap e_1 = \{v\}$ and $e \cap e_2 = \{v\}$. Let $\pi = a, \dots, e_1, e, e_2, \dots, b$. Then $\pi' = a, \dots, e_1, e_2, \dots, b$ is also a path from a to b . But π' does not contain any vertex of F , contradicting the assumption that F is a separator of $\text{LG}(\mathcal{P})$. Hence, neither such a minimal separator F nor such a vertex u exist. \square

Lemma 4. *If F is a minimal separator of $\text{LG}(\mathcal{P})$, then $\text{LG}(\mathcal{P}) - F$ has exactly two connected components.*

Proof. Assume that $\text{LG}(\mathcal{P}) - F$ has more than two connected components. By Lemma 1, $\text{LG}(\mathcal{P}) - F$ has at least two full components. Let H_1 and H_2 be two full components of $\text{LG}(\mathcal{P}) - F$. Let H_3 be a connected component of $\text{LG}(\mathcal{P}) - F$ different from H_1 and H_2 . By assumption $\text{LG}(\mathcal{P})$ is connected. Thus, there exists an edge $\{e, e_3\}$ in $\text{LG}(\mathcal{P})$ where $e \in F$ and $e_3 \in H_3$. Since H_1 and H_2 are full components, there exist edges $\{e, e_1\}$ and $\{e, e_2\}$ in $\text{LG}(\mathcal{P})$ where $e_1 \in V(H_1)$ and $e_2 \in V(H_2)$.

Let $e = \{u, v\}$, and assume without loss of generality that $u \in e \cap e_3$. Then, there is no vertex $f \in V(H_1)$ where $u \in e \cap f$. Thus, $v \in e \cap e_1$. Similarly, there is no vertex $f \in V(H_2)$ such that $u \in f \cap e$ or $v \in f \cap e$. But then H_2 does not contain a vertex adjacent to e , so H_2 is not a full component, a contradiction. \square

Corollary 1. *If F is a minimal separator of $\text{LG}(\mathcal{P})$, then $\text{LG}(\mathcal{P}) - F'$ is connected for any $F' \subset F$.*

Proof of Lemma 2. We prove that if F is a legal minimal separator of $\text{LG}(\mathcal{P})$ then F is a nice minimal cut of $G(\mathcal{P})$. The proof for the other direction is similar and is omitted.

First, we show that F is a cut of $G(\mathcal{P})$. Assume the contrary. Let $\{u, v\}$ and $\{p, q\}$ be vertices in different components of $\text{LG}(\mathcal{P}) - F$. Since $G(\mathcal{P}) - F$ is connected, there exists a path between vertices u and q . Also, $\{u, v\} \notin F$ and $\{p, q\} \notin F$. Thus, by Observation 1 there also exists a path between vertices $\{u, v\}$ and $\{p, q\}$ of $\text{LG}(\mathcal{P}) - F$. This implies that $\{u, v\}, \{p, q\}$ are in the same connected component of $\text{LG}(\mathcal{P}) - F$, a contradiction. Thus F is a cut.

Next we show that F is a nice cut of $G(\mathcal{P})$. For every $T \in \mathcal{P}$ all the vertices of $\text{LG}(T)$ in F form a clique in $\text{LG}(T)$. Thus, all the edges of T in F are incident on a common vertex, so F is a legal cut. To complete the proof, assume that $G(\mathcal{P}) - F$ has a connected component with no edge and let u be the vertex in one such component. Then, $\text{Inc}(u) \subseteq F$. But F is a minimal separator of $\text{LG}(\mathcal{P})$, and by Lemma 3, $\text{Inc}(u) \not\subseteq F$, a contradiction. Thus, F is a nice cut.

Lastly, we show that F is a minimal cut of $G(\mathcal{P})$. Assume, on the contrary, that there exists $F' \subset F$ where $G(\mathcal{P}) - F'$ is disconnected. Since $F' \subset F$ and every connected component of $G(\mathcal{P}) - F$ has at least one edge, every connected component of $G(\mathcal{P}) - F'$ also has at least one edge. Let $\{u, v\}$ and $\{p, q\}$ be the edges in different components of $G(\mathcal{P}) - F'$. By Corollary 1, $\text{LG}(\mathcal{P}) - F'$ is connected and thus, there is a path between $\{u, v\}$ and $\{p, q\}$ in $\text{LG}(\mathcal{P}) - F'$. By Observation 1 there must also be a path between vertices u and p in $G(\mathcal{P}) - F'$. Hence, edges $\{u, v\}$ and $\{p, q\}$ are in the same connected component of $G - F'$, a contradiction. Thus, F is a minimal cut. \square

Lemma 5. *Two legal minimal separators F and F' of $\text{LG}(\mathcal{P})$ are parallel if and only if the nice minimal cuts F and F' are parallel in $G(\mathcal{P})$.*

Proof. Assume that legal minimal separators F and F' of $\text{LG}(\mathcal{P})$ are parallel, but nice minimal cuts F and F' of $G(\mathcal{P})$ are not. Then, there exists $\{\{u, v\}, \{p, q\}\} \subseteq F'$ where $\{u, v\}$ and $\{p, q\}$ are in different components of $G(\mathcal{P}) - F$. Since F and F' are parallel separators in $\text{LG}(\mathcal{P})$, and F does not contain $\{u, v\}$ and $\{p, q\}$, there exists a path between vertices $\{u, v\}$ and $\{p, q\}$ in $\text{LG}(\mathcal{P}) - F$. Then, by Observation 1 there also exists a path between vertices u and q in $G(\mathcal{P}) - F$. Thus, edges $\{u, v\}$ and $\{p, q\}$ are in the same connected component of $G(\mathcal{P}) - F$, a contradiction.

The other direction can be proved similarly, using Observation 1. \square

The next lemma, from [9], follows from the definition of restricted triangulation.

Lemma 6. *Let H be a restricted triangulation of $\text{LG}(\mathcal{P})$ and let (T, B) be a clique tree of H . Let $e = \{u, v\}$ be any vertex in $\text{LG}(\mathcal{P})$. Then, there does not exist a node $x \in V(T)$ where $B(x)$ contains vertices from both $\text{Inc}(u) \setminus e$ and $\text{Inc}(v) \setminus e$.*

Lemma 7. *Let T be a tree in \mathcal{P} and suppose F is a minimal cut of $G(\mathcal{P})$ that contains precisely one edge e of T . Then, the edges of the two subtrees of $T - e$ are in different connected components of $G(\mathcal{P}) - F$.*

Proof. Since F is a minimal cut of $G(\mathcal{P})$, the endpoints of e are in different connected components of $G(\mathcal{P}) - F$. Let $e = \{u, v\}$. For every $x \in e$, let T_x represent the subtree containing vertex x in $T - e$. Edge e is the only edge of T in F . Thus, for every $x \in e$ all the edges of T_x are in the same connected component of $G(\mathcal{P}) - F$ as vertex x . Since the endpoints of e are in different connected components of $G(\mathcal{P}) - F$, the edges of T_u and T_v are also in different connected components of $G(\mathcal{P}) - F$. \square

4 Characterizing Compatibility via Cuts

A set \mathcal{F} of cuts of $G(\mathcal{P})$ is *complete* if, for every input tree $T \in \mathcal{P}$ and every internal edge e of T , there exists a cut $F \in \mathcal{F}$ where e is the only edge of T in F .

Lemma 8. *$G(\mathcal{P})$ has a complete set of pairwise parallel nice minimal cuts if and only if it has a complete set of pairwise parallel legal minimal cuts.*

Proof. The “only if part” follows from the definition of a nice cut. Let \mathcal{F} be a complete set of pairwise parallel legal minimal cuts. Consider any minimal subset \mathcal{F}' of \mathcal{F} that is also complete. Let F be a legal minimal cut of \mathcal{F}' . Since \mathcal{F}' is minimal, there exists an edge $e \in F$ of some input tree T such that e is the only edge of T in F . Also, since e is an internal edge, both the subtrees of $T - e$ have at least one edge each. Thus by Lemma 7, both the connected components of $G(\mathcal{P}) - F$ have at least one edge each. Hence, F is a nice minimal cut of $G(\mathcal{P})$. It thus follows that \mathcal{F}' is a complete set of pairwise parallel nice minimal cuts of $G(\mathcal{P})$. \square

We now characterize the compatibility of a profile in terms of minimal cuts in the display graph of the profile.

Theorem 6. *A profile \mathcal{P} of unrooted phylogenetic trees is compatible if and only if there exists a complete set of pairwise parallel legal minimal cuts for $G(\mathcal{P})$.*

Example 1. For the display graph of Fig. 1, let $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$, where $F_1 = \{\{1, 2\}, \{5, 6\}\}$, $F_2 = \{\{2, 3\}, \{6, 7\}, \{5, 6\}\}$, $F_3 = \{\{4, 5\}, \{1, 2\}, \{1, c\}\}$ and $F_4 = \{\{6, 7\}, \{2, f\}\}$. Then, \mathcal{F} is a complete set of pairwise parallel nice minimal cuts.

Theorem 6 and Lemmas 2, 5, and 8 imply an analogous result for $\text{LG}(\mathcal{P})$. A set \mathcal{F} of legal minimal separators of $\text{LG}(\mathcal{P})$ is *complete*, if for every internal edge e of an input tree T , there exists a separator $F \in \mathcal{F}$ where e is the only vertex of $\text{LG}(T)$ in F .

Theorem 7. *A profile \mathcal{P} of unrooted phylogenetic trees is compatible if and only if there exists a complete set of pairwise parallel legal minimal separators for $\text{LG}(\mathcal{P})$.*

Theorem 6 follows from Theorem 5, Lemma 8, and the next result.

Lemma 9. *The following two statements are equivalent.*

- (i) *There exists a maximal set \mathcal{F} of pairwise parallel minimal separators of $\text{LG}(\mathcal{P})$ where every separator in \mathcal{F} is legal.*
- (ii) *There exists a complete set of pairwise parallel nice minimal cuts for $G(\mathcal{P})$.*

Proof. (i) \Rightarrow (ii): We show that for every internal edge $e = \{u, v\}$ of an input tree T there exists a minimal separator in \mathcal{F} that contains only vertex e from $\text{LG}(T)$. Then it follows from Lemmas 2 and 5 that \mathcal{F} is a complete set of pairwise parallel nice minimal cuts for display graph $G(\mathcal{P})$.

As shown in the proof of Theorem 5, $\text{LG}(\mathcal{P})_{\mathcal{F}}$ is a restricted minimal triangulation of $\text{LG}(\mathcal{P})$. Let (S, B) be a clique tree of $\text{LG}(\mathcal{P})_{\mathcal{F}}$. By definition, the vertices in each of the sets $\text{Inc}(u)$ and $\text{Inc}(v)$ form a clique in $\text{LG}(\mathcal{P})$. Consider any vertex p of S where $\text{Inc}(u) \subseteq B(p)$ and any vertex q of S where $\text{Inc}(v) \subseteq B(q)$. (Since (S, B) is a clique tree of $\text{LG}(\mathcal{P})_{\mathcal{F}}$, such vertices p and q must exist.) Also, by Lemma 6, $p \neq q$, $B(p) \cap (\text{Inc}(v) \setminus \{e\}) = \emptyset$ and $B(q) \cap (\text{Inc}(u) \setminus \{e\}) = \emptyset$.

Let $\pi = p, x_1, x_2, \dots, x_m, q$ be the path from p to q in S where $m \geq 0$. Let $x_0 = p$ and $x_{m+1} = q$. Let x_i be the vertex nearest to p in path π where $i \in [m + 1]$ and $B(x_i) \cap (\text{Inc}(u) \setminus \{e\}) = \emptyset$. Let $F = B(x_{i-1}) \cap B(x_i)$. Then by Theorem 2, $F \in \mathcal{F}$. Since $\text{Inc}(u) \cap \text{Inc}(v) = \{e\}$, by the coherence property, $e \in B(x_j)$ for every $j \in [m]$. Thus, $e \in F$. By Lemma 6, $B(x_{i-1}) \cap (\text{Inc}(v) \setminus \{e\}) = \emptyset$. Since $B(x_i) \cap (\text{Inc}(u) \setminus \{e\}) = \emptyset$, $F \cap \text{Inc}(u) = \{e\}$ and $F \cap \text{Inc}(v) = \{e\}$. Thus, for every vertex $e' \in \text{LG}(T)$ where $e \neq e'$ and $e \cap e' \neq \emptyset$, $e' \notin F$. Also, since every separator in \mathcal{F} is legal, we have $f \notin F$ for every vertex $f \in \text{LG}(T)$ where $f \cap e = \emptyset$. Thus, e is the only vertex of $\text{LG}(T)$ in F .

(i) \Leftarrow (ii): Consider any complete set of pairwise parallel nice minimal cuts \mathcal{F}' of $G(\mathcal{P})$. By Lemmas 2 and 5, \mathcal{F}' is a set of pairwise parallel legal minimal separators of $\text{LG}(\mathcal{P})$. There exists a maximal set \mathcal{F} of pairwise parallel minimal separators where $\mathcal{F}' \subseteq \mathcal{F}$.

Assume that $\mathcal{F} \setminus \mathcal{F}'$ contains a minimal separator F that is not legal. Then, there must exist a tree $T \in \mathcal{P}$ where at least two nonincident edges $e_1 = \{x, y\}$ and $e_2 = \{x', y'\}$ of T are in F . Consider any internal edge e_3 in T where e_1 and e_2 are in different components of $T - e_3$. Such an edge exists because e_1 and e_2 are nonincident. Since \mathcal{F}' is complete, there exists a cut $F' \in \mathcal{F}'$ where e_3 is the only edge of T in F' . Since F and F' are in \mathcal{F} , they are parallel to each other and vertices e_1 and e_2 are in the same connected component of $\text{LG}(\mathcal{P}) - F'$. Thus, by Observation 1, there exists a path between vertices x and x' in $G(\mathcal{P}) - F'$ and edges e_1 and e_2 are also in the same connected component of $G(\mathcal{P}) - F'$. But by Lemma 7 that is impossible.

Thus, every separator of $\mathcal{F} \setminus \mathcal{F}'$ is legal and \mathcal{F} is a maximal set of pairwise minimal separators of $\text{LG}(\mathcal{P})$ where every separator in \mathcal{F} is legal. □

5 Splits and Cuts

We first argue that for every nice minimal cut of $G(\mathcal{P})$ we can derive a split of $\mathcal{L}(\mathcal{P})$.

Lemma 10. *Let F be a nice minimal cut of $G(\mathcal{P})$ and let G_1 and G_2 be the two connected components of $G(\mathcal{P}) - F$. Then, $\mathcal{L}(G_1) \mid \mathcal{L}(G_2)$ is a split of $\mathcal{L}(\mathcal{P})$.*

Proof. Consider G_i for each $i \in \{1, 2\}$. We show that $\mathcal{L}(G_i)$ is non-empty. Since F is nice, G_i contains at least one edge e of $G(\mathcal{P})$. If e is a non-internal edge, then $\mathcal{L}(G_i)$ is non-empty. Assume that $e = \{u, v\}$ is an internal edge of some input tree T . If F does not contain an edge of T , then $\mathcal{L}(T) \subseteq \mathcal{L}(G_i)$ and thus $\mathcal{L}(G_i)$ is non-empty. Assume that F contains one or more edges of T . Let T_u, T_v be the two subtrees of $T - e$. Since F is a nice minimal cut, F contains edges from either T_u or T_v but not both. Without loss of generality assume that F does not contain edges from T_u . Then, every edge of T_u is in the same component as e . Since T_u contains at least one leaf, $\mathcal{L}(G_i)$ is non-empty. Thus, $\mathcal{L}(G_1)|\mathcal{L}(G_2)$ is a split of $\mathcal{L}(\mathcal{P})$. \square

Let $\sigma(F)$ denote the split of $\mathcal{L}(\mathcal{P})$ induced by a nice minimal cut F . If \mathcal{F} is a set of nice minimal cuts of $G(\mathcal{P})$, $\Sigma(\mathcal{F})$ denotes the set of all the non-trivial splits in $\bigcup_{F \in \mathcal{F}} \sigma(F)$. The following result expresses the relationship between complete sets of nice minimal cuts and the compatibility of splits.

Theorem 8. *If $G(\mathcal{P})$ has a complete set of pairwise parallel nice minimal cuts \mathcal{F} , then $\Sigma(\mathcal{F})$ is compatible and any compatible tree for $\Sigma(\mathcal{F})$ is also a compatible tree for \mathcal{P} .*

Example 2. For the cuts of the display graph in Fig. 1 given in Example 1, we have $\sigma(F_1) = abc|defg$, $\sigma(F_2) = abcf|gde$, $\sigma(F_3) = ab|cdefg$, and $\sigma(F_4) = abcde|fg$. Note that these splits are pairwise compatible.

The proof of Theorem 8 uses the following lemma.

Lemma 11. *Let F_1 and F_2 be two parallel nice minimal cuts of $G(\mathcal{P})$. Then, $\sigma(F_1)$ and $\sigma(F_2)$ are compatible.*

Proof. Let $\sigma(F_1) = U_1|U_2$ and $\sigma(F_2) = V_1|V_2$. Assume that $\sigma(F_1)$ and $\sigma(F_2)$ are incompatible. Thus, $U_i \cap V_j \neq \emptyset$ for every $i, j \in \{1, 2\}$. Let $a \in U_1 \cap V_1$, $b \in U_1 \cap V_2$, $c \in U_2 \cap V_1$ and $d \in U_2 \cap V_2$. Since $\{a, b\} \subseteq U_1$, there exists a path π_1 between leaves a and b in $G(\mathcal{P}) - F_1$. But a and b are in different components of $G(\mathcal{P}) - F_2$. Thus, an edge e_1 of path π_1 is in the cut F_2 . Similarly, $\{c, d\} \subseteq U_2$ and there exists a path π_2 between labels c and d in $G(\mathcal{P}) - F_1$. Since c and d are in different components of $G(\mathcal{P}) - F_2$, cut F_2 contains an edge e_2 of path π_2 . But π_1 and π_2 are in different components of $G(\mathcal{P}) - F_1$, so edges e_1 and e_2 are in different components of $G(\mathcal{P}) - F_1$. Since $\{e_1, e_2\} \subseteq F_2$, the cuts F_1 and F_2 are not parallel, a contradiction. \square

Proof of Theorem 8. The compatibility of $\Sigma(\mathcal{F})$ follows from Lemma 11 and Theorem 1. Let S be a compatible tree for $\Sigma(\mathcal{F})$, let T be an input tree of \mathcal{P} , let $S' = S|_{\mathcal{L}(T)}$, and let e be any internal edge of T . We now show that S' displays $\sigma(e)$.

Let $\sigma(e) = A|B$. There exists a cut $F \in \mathcal{F}$ where e is the only edge of T in F . By Lemma 7, since F is minimal, the leaves of sets A and B are in different components of $G(\mathcal{P}) - F$. Thus, if $\sigma(F) = A'|B'$ then up to renaming of sets we have $A \subseteq A'$ and $B \subseteq B'$. Because S displays $\sigma(F)$, S' also displays $\sigma(e)$. Since S' displays all the splits of T , T can be obtained from S' by contracting zero or more edges [13]. Thus, S displays T . Since S displays every tree in \mathcal{P} , S is a compatible tree for \mathcal{P} . \square

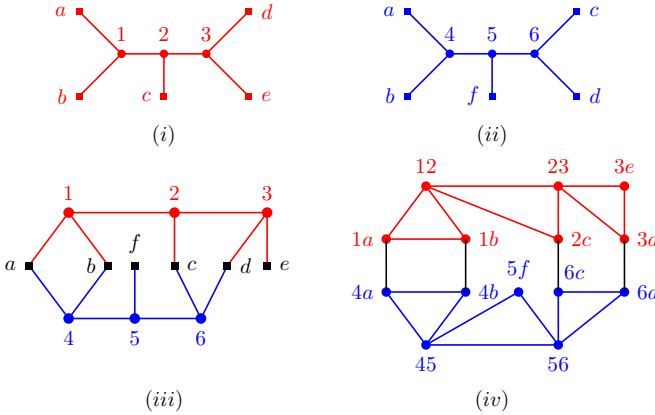


Fig. 2. (i) First input tree. (ii) Second input tree, which agrees with the first. (iii) Display graph of the input trees. (iv) Edge label intersection graph of the input trees, where label uv represents edge $\{u, v\}$ of the display graph.

6 Characterizing Agreement via Cuts

The following characterization of agreement is similar to the one for tree compatibility given by Theorem 6, except for an additional restriction on the minimal cuts.

Theorem 9. *A profile \mathcal{P} has an agreement supertree if and only if $G(\mathcal{P})$ has a complete set \mathcal{F} of pairwise parallel legal minimal cuts where, for every cut $F \in \mathcal{F}$ and for every $T \in \mathcal{P}$, there is at most one edge of T in F .*

Example 3. One can verify that the display graph of Fig. 1 does not meet the conditions of Theorem 9 and, thus, the associated profile does not have an AST. On the other hand, for the display graph of Fig. 2, let $\mathcal{F} = \{F_1, F_2, F_3\}$, where $F_1 = \{\{1, 2\}, \{4, 5\}\}$, $F_2 = \{\{1, 2\}, \{5, 6\}\}$ and $F_3 = \{\{2, 3\}, \{6, d\}\}$. For any given input tree T , every cut in \mathcal{F} has at most one edge of T . Also, \mathcal{F} is a complete set of pairwise parallel legal minimal cuts. Thus, by Theorem 9, the input trees of Fig. 2 have an AST.

The analogue of Theorem 9 for $LG(\mathcal{P})$ stated next follows from Theorem 9 and Lemmas 2, 5, and 8.

Theorem 10. *A profile \mathcal{P} has an agreement supertree if and only if $LG(\mathcal{P})$ has a complete set \mathcal{F} of pairwise parallel legal minimal separators where, for every $F \in \mathcal{F}$ and every $T \in \mathcal{P}$, there is at most one vertex of $LG(T)$ in F .*

Theorem 9 follows from Lemma 8 and the next result.

Lemma 12. *A profile \mathcal{P} has an agreement supertree if and only if $G(\mathcal{P})$ has a complete set \mathcal{F} of pairwise parallel nice minimal cuts where, for every cut $F \in \mathcal{F}$ and every $T \in \mathcal{P}$, there is at most one edge of T in F .*

The rest of the section is devoted to the proof of Lemma 12.

Let S be an AST of \mathcal{P} and let $e = \{u, v\}$ be an edge of S . Let S_u and S_v be the subtrees of $S - e$ containing u and v , respectively. Let $L_u = \mathcal{L}(S_u)$ and $L_v = \mathcal{L}(S_v)$. Thus, $\sigma_e(S) = L_u|L_v$. Assume that there exists an input tree T where $\mathcal{L}(T) \cap L_x \neq \emptyset$ for each $x \in \{u, v\}$. Then there exists an edge $f \in E(T)$ where, if $\sigma_f(T) = A_1|A_2$, then $A_1 \subseteq L_u$ and $A_2 \subseteq L_v$. (If there were no such edge, $S|_{\mathcal{L}(T)}$ would contain a split that is not in T and would thus not be isomorphic to T .) We call e an *agreement edge* of S corresponding to edge f of T . Note that there does not exist any other edge f' of T where e is also an agreement edge of S with respect to edge f' of T .

Given an AST S of \mathcal{P} , we define a function Ψ from $E(S)$ to subsets of edges of $G(\mathcal{P})$ as follows. For every $e \in E(S)$, an edge f of an input tree T is in $\Psi(e)$ if and only if e is an agreement edge of S corresponding to edge f of T . Observe that Ψ is uniquely defined. We call Ψ the *cut function* of S . Given an edge $e \in E(S)$, we define a set V_x for every $x \in e$ as follows. For every $T \in \mathcal{P}$, V_x contains all the vertices of the minimal subtree of T connecting the labels in $\mathcal{L}(T) \cap L_x$. Note that if $e = \{u, v\}$ then $\{V_u, V_v\}$ is a partition of $V(G(\mathcal{P}))$.

Lemma 13. *Let S be an AST of \mathcal{P} and let Ψ be the cut function of S . Then,*

- (i) *for every edge $e \in E(S)$, $\Psi(e)$ is a cut of $G(\mathcal{P})$ and*
- (ii) *for any edge $e \in E(S)$, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$ if and only if $G(\mathcal{P}) - \Psi(e)$ has exactly two connected components.*

Proof. (i) Let $e = \{u, v\}$. We show that $G(\mathcal{P}) - \Psi(e)$ does not contain an edge whose endpoints are in distinct sets of $\{V_u, V_v\}$. Assume the contrary. Let $f = \{x, y\}$ be an edge of $G(\mathcal{P}) - \Psi(e)$ where $x \in V_u$ and $y \in V_v$. Since $f \in G(\mathcal{P}) - \Psi(e)$, $f \notin \Psi(e)$. Suppose f is an edge of input tree T . There are two cases.

1. $\Psi(e)$ does not contain an edge of T . Then, there exists an endpoint p of e where $\mathcal{L}(T) \subseteq L_p$. Without loss of generality, let $u = p$. Then, $V(T) \subseteq V_u$ and thus $y \in V_u$, a contradiction.
2. $\Psi(e)$ contains an edge $f' \neq f$ of T . Let $f' = \{r, s\}$ and let $L_r \subseteq L_u$ and $L_s \subseteq L_v$. Let x, r be the vertices of f and f' where $L_x \subset L_r$. Since T is a phylogenetic tree, such vertices x and r exist. Since $L_r \subseteq L_u$, both the endpoints of f are in V_u , a contradiction.

Thus, $G(\mathcal{P}) - \Psi(e)$ does not contain an edge whose endpoints are in different sets of $\{V_u, V_v\}$. Since V_u and V_v are non-empty, it follows that $\Psi(e)$ is a cut of $G(\mathcal{P})$.

(ii) The “only if” part follows from the definition of a minimal cut. We now prove the “if” part. Let $e = \{u, v\}$. Assume that $G(\mathcal{P}) - \Psi(e)$ has exactly two connected components. From the proof of (i), V_u and V_v are the vertex sets of those two connected components. Consider any edge $f \in \Psi(e)$. The endpoints of f are in different sets of $\{V_u, V_v\}$ and thus are in different connected components of $G(\mathcal{P}) - \Psi(e)$. This implies that $G(\mathcal{P}) - (\Psi(e) \setminus \{f\})$ is connected. Thus, if $G(\mathcal{P}) - \Psi(e)$ has exactly two connected components, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$. □

Let S be an AST of \mathcal{P} and let e be an edge of S . Although the preceding result shows that $\Psi(e)$ is a cut of $G(\mathcal{P})$, $\Psi(e)$ may not be minimal. We now argue that we can always construct an agreement supertree whose cut function gives minimal cuts.

Suppose $e = (u, v)$ is an edge of S where $\Psi(e)$ is not minimal. Let $\{L_1, \dots, L_m\}$ be the partition of L_v where for every $i \in [m]$, $L_i = \mathcal{L}(C) \cap L_v$ for some connected component C in $G(\mathcal{P}) - \Psi(e)$. We assume without loss of generality that $m > 1$ (if not, we can just exchange the roles of u and v). Let R_v be the rooted tree derived from S_v by distinguishing vertex v as the root. Let $R_{v,i}$ be the (rooted) tree obtained from the minimal subtree of R_v connecting the labels in L_i by distinguishing the vertex closest to v as the root and suppressing every other vertex that has degree two. To split edge e at u is to construct a new tree S' from S in two steps: (i) delete the vertices of R_v from S and (ii) for every $i \in [m]$, add an edge from u to the root of $R_{v,i}$.

We can show the following by repeatedly splitting edges that do not correspond to minimal cuts. For brevity, we omit the proof.

Lemma 14. *If \mathcal{P} has an AST, then it has an AST S of \mathcal{P} whose cut function Ψ satisfies the following: For every edge $e \in S$, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$.*

Proof of Lemma 12. (\Leftarrow) Assume that \mathcal{P} has an AST. Then, by Lemma 14, \mathcal{P} has an AST S whose cut function Ψ has the property that, for every edge $e \in E(S)$, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$. Let \mathcal{F} be the set of all $\Psi(e)$ such that e is an internal edge of S . Then, \mathcal{F} is a set of minimal cuts of $G(\mathcal{P})$. Further, by definition of Ψ , for every $F \in \mathcal{F}$ and for every $T \in \mathcal{P}$, F contains at most one edge of T . Thus every cut in \mathcal{F} is legal. We now prove that \mathcal{F} is a complete set of pairwise parallel nice minimal cuts of $G(\mathcal{P})$.

We first prove that every cut in \mathcal{F} is nice. Consider any $F \in \mathcal{F}$. Let $e = \{u, v\}$ be the internal edge of S where $\Psi(e) = F$. Let T be an input tree that has an internal edge f in $\Psi(e)$. Since e is an internal edge at least one such input tree exists; otherwise $\Psi(e)$ is not a minimal cut. Now, by definition, f is the only edge of T in $\Psi(e)$, so, by Lemma 7, each of the two connected components of $G(\mathcal{P}) - \Psi(e)$ has at least one non-internal edge of T . Hence, F is a nice minimal cut of $G(\mathcal{P})$.

To prove that the cuts in \mathcal{F} are pairwise parallel, we argue that for any two distinct internal edges e_1 and e_2 of S , $\Psi(e_1)$ and $\Psi(e_2)$ are parallel. There exist vertices $x \in e_1$ and $y \in e_2$ where $L_x \subseteq L_y$. For every edge $f \in \Psi(e_1)$, we show that $f \in \Psi(e_2)$ or $f \subseteq V_y$. It then follows that $\Psi(e_1)$ and $\Psi(e_2)$ are parallel. Let f be an edge of input tree T . Then there exists $z \in f$ where $L_z \subseteq L_x$. Thus, $L_z \subseteq L_y$ and $z \in V_y$. By Lemma 13, all the vertices of V_y are in the same connected component of $G(\mathcal{P}) - \Psi(e_2)$. Thus, $f \in \Psi(e_2)$ or $f \subseteq V_y$.

Lastly, we show that \mathcal{F} is complete. Consider any internal edge $f = \{p, q\}$ of some input tree T . Since S is an AST of \mathcal{P} , there exists an edge $e = \{u, v\}$ where, up to relabeling of sets, $L_p \subseteq L_u$ and $L_q \subseteq L_v$. Thus, e is an agreement edge of S corresponding to f , so $f \in \Psi(e)$. Since f is an internal edge, e is also an internal edge of S and thus $\Psi(e) \in \mathcal{F}$. Hence, for every internal edge f of an input tree there is a cut $F \in \mathcal{F}$ where $f \in F$. Thus, S is complete.

(\Rightarrow) Assume that there exists a complete set \mathcal{F} of pairwise parallel nice minimal cuts of $G(\mathcal{P})$ where, for every $F \in \mathcal{F}$ and every $T \in \mathcal{P}$, F contains at most one edge of T . By Theorem 8, $\Sigma(\mathcal{F})$ is compatible and, by Theorem 1, there exists an unrooted tree S where $\Sigma(\mathcal{F}) = \Sigma(S)$. We prove that S is an AST of \mathcal{P} by showing that $\Sigma(S|_{\mathcal{L}(T)}) = \Sigma(T)$ for every input tree $T \in \mathcal{P}$.

Consider an input tree T of \mathcal{P} . Let $X_1|X_2$ be the non-trivial split of T corresponding to edge $f \in E(T)$. Since \mathcal{F} is complete, there exists a cut $F \in \mathcal{F}$ where $f \in F$. If $\sigma(F) = Y_1|Y_2$, by Lemma 7, up to relabeling of sets, $X_i \subseteq Y_i$ for every $i \in \{1, 2\}$. Since $\sigma(F)$ is a split of S , this implies that $\Sigma(T) \subseteq \Sigma(S|_{\mathcal{L}(T)})$.

Consider any non-trivial split $P_1|P_2$ of $\Sigma(S)$ where $P_i \cap \mathcal{L}(T) \neq \emptyset$ for each $i \in \{1, 2\}$. Let $Q_i = P_i \cap \mathcal{L}(T)$ for each $i \in \{1, 2\}$. Since $\Sigma(S) = \Sigma(\mathcal{F})$, there exists a cut $F \in \mathcal{F}$ where $\sigma(F) = P_1|P_2$. Since P_1 and P_2 are in different connected components of $G(\mathcal{P}) - F$, Q_1 and Q_2 are also in different connected components of $G(\mathcal{P}) - F$. Thus, there exists an edge f' of T in F . Since F does not contain any other edge of T , $\sigma(f') = Q_1|Q_2$. Thus, $\Sigma(S|_{\mathcal{L}(T)}) \subseteq \Sigma(T)$. \square

7 Conclusion

We have shown that the characterization of tree compatibility in terms of restricted triangulations of the edge label intersection graph transforms into a characterization in terms of minimal cuts in the display graph. These two characterizations are closely related to the legal triangulation characterization of [15]. We also derived characterizations of the agreement supertree problem in terms of minimal cuts and minimal separators of the display and edge label intersection graphs respectively.

It is not known if the agreement supertree problem is fixed parameter tractable when parametrized by the number of input trees. It remains to be seen whether any of these characterizations can lead to explicit fixed parameter algorithms for the tree compatibility and agreement supertree problems when parametrized by the number of trees.

Acknowledgment. We thank Sylvain Guillemot for his valuable comments.

References

1. Aho, A., Sagiv, Y., Szymanski, T., Ullman, J.: Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput.* 10(3), 405–421 (1981)
2. Amborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *J. Algorithms* 12(2), 308–340 (1991)
3. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.* 31(1), 212–232 (2001)
4. Bryant, D., Lagergren, J.: Compatibility of unrooted phylogenetic trees is FPT. *Theor. Comput. Sci.* 351, 296–302 (2006)
5. Buneman, P.: The recovery of trees from measures of dissimilarity. In: *Mathematics in the Archaeological and Historical Sciences*, pp. 387–395. Edinburgh University Press, Edinburgh (1971)
6. Courcelle, B.: The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Inf. Comput.* 85(1), 12–75 (1990)
7. Gordon, A.D.: Consensus supertrees: The synthesis of rooted trees containing overlapping sets of labelled leaves. *Journal of Classification* 9, 335–348 (1986)
8. Gusfield, D.: The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. *J. Comput. Biol.* 17(3), 383–399 (2010)

9. Gysel, R., Stevens, K., Gusfield, D.: Reducing problems in unrooted tree compatibility to restricted triangulations of intersection graphs. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS, vol. 7534, pp. 93–105. Springer, Heidelberg (2012)
10. Heggernes, P.: Minimal triangulations of graphs: A survey. *Discrete Math.* 306(3), 297 (2006)
11. Ng, M.P., Wormald, N.C.: Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics* 69(1-2), 19–31 (1996)
12. Parra, A., Scheffler, P.: Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Appl. Math.* 79(1-3), 171–188 (1997)
13. Semple, C., Steel, M.: *Phylogenetics*. Oxford Lecture Series in Mathematics. Oxford University Press, Oxford (2003)
14. Steel, M.A.: The complexity of reconstructing trees from qualitative characters and subtrees. *J. Classif.* 9, 91–116 (1992)
15. Vakati, S., Fernández-Baca, D.: Graph triangulations and the compatibility of unrooted phylogenetic trees. *Appl. Math. Lett.* 24(5), 719–723 (2011)

Unifying Parsimonious Tree Reconciliation

Nicolas Wieseke, Matthias Bernt, and Martin Middendorf

University of Leipzig, Faculty of Mathematics and Computer Science,
Augustusplatz 10, 04109 Leipzig, Germany
{wieseke,bernt,middendorf}@informatik.uni-leipzig.de

Abstract. Evolution is a process that is influenced by various environmental factors, e.g. the interactions between different species, genes, and biogeographical properties. Hence, it is interesting to study the combined evolutionary history of multiple species, their genes, and the environment they live in. A common approach to address this research problem is to describe each individual evolution as a phylogenetic tree and construct a tree reconciliation which is parsimonious with respect to a given event model. Unfortunately, most of the previous approaches are designed only either for host-parasite systems, for gene tree/species tree reconciliation, or biogeography. Hence, a method is desirable, which addresses the general problem of mapping phylogenetic trees and covering all varieties of coevolving systems, including e.g., predator-prey and symbiotic relationships. To overcome this gap, we introduce a generalized cophylogenetic event model considering the combinatorial complete set of local coevolutionary events. We give a dynamic programming based heuristic for solving the maximum parsimony reconciliation problem in time $O(n^2)$, for two phylogenies each with at most n leaves. Furthermore, we present an exact branch-and-bound algorithm which uses the results from the dynamic programming heuristic for discarding partial reconciliations. The approach has been implemented as a Java application which is freely available from <http://pacosy.informatik.uni-leipzig.de/coresym>.

Keywords: cophylogeny, coevolution, coevolutionary event model, reconciliation, host-parasite, gene tree/species tree, biogeography, symbiosis.

1 Introduction

Tree reconciliation analysis is a powerful tool in phylogenetics and has a wide variety of applications. It is used in cladistic biogeography as well as for studying host-parasite coevolution and gene/species tree inference [21]. A common principle for creating tree reconciliations is to use event-based maximum parsimony [24]. Therefore, coevolutionary events are defined together with a cost model for the events and a reconciliation of the trees is sought that minimizes the overall costs. Table 1 shows the common events used within the different types of applications. Two closely related types of this problem can be distinguished: tree inference and tree embedding/tree mapping. In the first case an

overarching tree is sought that embeds a given set of, possibly incongruent, trees. In the second case two (or more) trees are given and a mapping of those trees onto each other is sought. Although tree reconciliation refers to both types of problems, the scope of this paper lies on the latter case.

Starting in 1979 Goodman et al. [8] introduced the problem of embedding gene trees into species trees. They presented a method to construct most parsimonious reconciliations, based on the evolutionary events *gene duplication* and *gene loss*. Since then, several algorithms for gene tree/species tree reconciliation have been developed, e.g., GeneTree [20], SDI [29], Softparsmap [1], Notung [5,27], and Mowgli [7]. These tools either consider additional events like *lateral gene transfer* or *incomplete lineage sorting*, or they are extended to suit unresolved phylogenetic trees. Recently, [26] examines lateral gene transfer to and from species that are not represented in the phylogenetic tree, i.e. extincted or unsampled species. The respective event was called *lateral transfer from the dead*.

A similar problem arose in the field of biogeography, where species trees and area cladograms have to be reconciled. Nelson and Platnick [15] were the first who proposed general assumptions for inferring area cladograms from species trees that have been implemented in the software COMPONENT 1.5 [18]. There are several other approaches for inferring area cladograms from species trees, based on association matrices, e.g., component compatibility analysis [28], Brooks parsimony analysis [2], and three area statement analysis [14]. But all of them are pattern-based approaches and do not produce reconciled trees. The dispersal-variance analysis [23] implemented in the software DIVA is an event-based method considering *vicariance*, *duplication*, *dispersal*, and *extinction* events. It supports tree inference as well as tree embedding.

In 1988 Hafner and Nadler published a cophylogenetic analysis on pocket gophers and their chewing lice parasites [9]. Methods designed for biogeographical [15] as well as gene tree/species tree problems [11] to examine *cospeciation* events in host-parasite systems have been proposed in [10]. In the same year Ronquist and Nylin [24] suggested the usage of *colonisation*, *exclusion*, and *successive specialization* events with given weights relative to the probability of each event. Based on these events they developed a method for reconstructing the evolutionary history for two given phylogenetic trees and an association matrix describing the associations between extant species. This type of data set can be represented graphically by a so called tanglegram [3]. In the same paper Charleston developed a data structure, called jungles, and a method to construct all optimal solutions for the reconciliation problem with events *cospeciation*, *duplication*, *lineage sorting*, and *host switch* under a general weighting scheme. The approach was implemented in the software TreeMap 2.0 and extended in Tarzan [12] by the use of additional timing information. Jane [6] and CoRe-PA [13] use dynamic programming to efficiently compute reconciliations based on the same event model. The latter tool also considered unresolved phylogenies and presented a method for approximating event costs automatically. Additionally a first attempt was given to handle parasites infesting multiple hosts. The current version 4 of Jane

supports multi-host parasites and *failure to diverge* events as well. Furthermore, it is able to automatically resolve polytomies.

In [19] Page already pointed out the similarity between the three different types of reconciliation problems and gave a unified definition of reconciled trees. He presented the software COMPONENT 2.0, which he applied to data sets from all three types of problems.













Gene tree/species tree, biogeography, and host-parasite reconciliation problems have in common, that there exists an overarching tree into which the other trees have to be embedded, i.e., the gene trees have to be embedded into a species tree, the species tree into an area cladogram, and the parasite tree into a host tree. Until now there is no event model and respective algorithms which considers the general problem of two trees which also can be equitably mapped onto each other, as it is the case for, e.g., symbiotic systems or gene-gene interactions. To fill this gap and to straighten up the event proliferation we introduce a combinatorial complete event model of local association patterns. However, a certain type of application may consider a subset of all possible event types only. Therefore, the event model can be utilized by defining a cost model with infinity costs for neglected events. Furthermore, we present a dynamic programming heuristic as well as an exact branch-and-bound algorithm to construct tree reconciliations under the new event model. In [16] it was shown that even the special case of host-parasite tree reconciliation considering the events *cospeciation*, *sorting*, *duplication*, and *host switching* is NP-complete. Therefore, heuristic or approximation algorithms might be the only chance to obtain solutions in reasonable time for large data sets. However, for smaller data sets an computing an optimal solution might be feasible.

2 Basic Notations and Preliminaries

For the following formal description we select the cophylogenetic reconciliation of two dependent sets of species as reference problem. Other kinds of tree reconciliation problems, e.g., of species, areas, or genes, are covered analogously.

The evolution of a set of species is usually depicted as a *phylogenetic tree*, which is a tree $T = (V_T, E_T)$ with node set V_T , edge set E_T , and leaf set $L_T \subset V_T$. In the context of phylogenetic trees an internal node $u \in V_T \setminus L_T$ refers to a speciation of an ancestral species u into subspecies. An edge $(u', u) \in E_T$ represents the time span of the existence of a species u from its emergence after the speciation of u' until its own speciation or the present day. We refer to an edge (u', u) as e_u or simply u if it is clear from the context. If not stated otherwise we assume a phylogenetic tree being binary and rooted, i.e., each node has an outdegree of either two (internal node) or zero (leaf node) and there is exactly one node, the root ρ_T , with indegree zero whereas all other nodes $u \in V_T \setminus \rho_T$ have indegree one. For each internal node $u \in V_T \setminus L_T$ we denote the children of u as u_i with $i \in \{1, 2\}$. For technical reasons we introduce an artificial root ρ' and an edge (ρ', ρ) . In that way it is possible to refer to the time span of the existence of root species ρ by e_ρ , or simply edge ρ . We define a partial order \preceq_T on V_T

Table 1. Coevolutionary events with the various names used within the different types of applications - biogeography (BG), gene/species tree (GST) and host-parasite co-evolution (HPC). Solid lines represent the overarching tree (i.e., the area cladogram, species tree, and host tree, respectively), dotted lines represent the embedded tree (i.e., species tree, parasite tree, and gene tree, respectively).

	biogeography	gene tree/species tree	host-parasite
	vicariance [25], allopatric speciation [23]	speciation (null event)	cospeciation [17], codivergence [4], successive specialisation [24]
	duplication [25], sympatric speciation [23]	gene duplication [8]	duplication [17], independent speciation [17]
	partial extinction [25]	-	sorting [17], partial extinction [17], missing the boat [17]
	-	speciation and loss [7]	-
	complete extinction [25]	gene loss [8]	extinction [17]
	(partial) dispersal [25]	horizontal/lateral gene transfer [5,26], direct transfer [26]	host switch [17], partial switch [17]
	complete dispersal [25]	-	complete switch [17]
	-	transfer and loss [7]	-
	-	lateral transfer from the dead [26], indirect transfer [26]	-
	-	-	takeoff [13], exclusion [24]
	-	-	landing [13], colonisation [24]
	-	-	failure to speciate/diverge [17,4]

such that $u' \preceq_T u$, if and only if u' lies on the path from ρ to u . In addition, $u' \prec_T u$ if and only if $u' \preceq_T u$ and $u' \neq u$. Node u' is called an ancestor of u and u a descendant of u' , respectively. Furthermore, we define the *timing* τ of a tree as $\tau : V_T \rightarrow \mathbb{R}$ such that $\forall u', u \in V_T$ it holds that $u' \preceq_T u \Rightarrow \tau(u') \leq \tau(u)$ and $u' \prec_T u \Rightarrow \tau(u') < \tau(u)$, respectively. In the evolutionary context $\tau(u)$ represents the point in time at which the speciation of u took place.

Let (S, T, ϕ) be a pair of rooted binary trees $S = (V_S, E_S)$ and $T = (V_T, E_T)$ together with a mapping $\phi(s, t) : V_S \times V_T \rightarrow [0, 1]$ representing inter-species association strengths measured by a value between zero and one. The strength $\phi(s, t)$ can be interpreted as an a priori probability of two species s and t being associated. The given definition of ϕ is a generalization of the leaf-to-leaf associations φ defined in [3], extended by association strengths for extant and ancestral species. According to the notion of tanglegrams we refer to such a tuple (S, T, ϕ) as an *X-tanglegram*.

A *cophylogenetic reconciliation* for a given X-tanglegram (S, T, ϕ) can be described as a set of associations $\mathcal{R} \subseteq E_S \times E_T$ between edges, with \mathcal{R} being the reconciled interactions between extant as well as ancestral species. Depending on the type of application additional constraints on the set \mathcal{R} are required for a reconciliation to be phylogenetically meaningful, e.g., timing constraints. These will be discussed later on. A *sub-reconciliation* $\mathcal{R}_{|s,t}$ is a subset of \mathcal{R} such that $(e_u, e_v) \in \mathcal{R}_{|s,t} \Leftrightarrow (e_u, e_v) \in \mathcal{R}$ and $s \preceq_S u$ and $t \preceq_T v$, i.e., $\mathcal{R}_{|s,t}$ is a reconciliation of the subtrees of S and T rooted at nodes s and t , respectively.

Note that in contrast to previous approaches we describe a reconciliation as a mapping between edges. In [12] a reconciliation was assumed to be a bijective mapping $m : \{V_T\} \rightarrow \{V_S \cup E_S\}$ from the nodes of the parasite tree to the nodes and edges of the host tree. However, from a given mapping m the respective edge-to-edge mapping \mathcal{R} can be derived as follows. Let $(t, t_i) \in E_T$ be an edge from T and $s \in V_S$ be the node for which $m(t) = s$ or $m(t) = e_s$. Furthermore, let $s'' \in V_S$ be a node for which $m(t_i) = s''$ or $m(t_i) = e_{s''}$. Then it holds that $(e_s, e_t), (e_{s''}, e_{t_i}) \in \mathcal{R}$. If s is an ancestor of s'' then for all nodes $s', s \prec_S s' \prec_S s''$ it holds that $\{(e_{s'}, e_{t_i})\} \in \mathcal{R}$. Additionally, if $m(t) = e_s$ is an edge mapping then also $(e_s, e_{t_i}) \in \mathcal{R}$.

3 Methods

3.1 Generalized Coevolutionary Event Model

Previous reconciliation approaches always consider a certain set of cophylogenetic events. These events are designed to suit to a certain type of application and some of them are combinations of other events, e.g., *speciation and loss* or *transfer and loss*. In this section a generalized event model is presented that covers all possible local association patterns, i.e., all possible associations between coincident edges of two nodes. Hence, this event model can be applied to most of the applications. In the following the term *species* will be interchangeably used for *edge*. This is because associations between species have a one-to-one correspondence to pairs of associated edges in the reconciliation \mathcal{R} .

In this approach a cophylogenetic event is defined as a relation between the sets of coincident edges of two nodes $s \in V_S$ and $t \in V_T$. Formally, an event is described as a subset of $\{e_s, e_{s_1}, e_{s_2}\} \times \{e_t, e_{t_1}, e_{t_2}\}$. For a pair of species (s, t) we define a binary variable $b_{(s,t)} \in \{0, 1\}$, such that $b_{(s,t)} = 1$ if species s and t are assumed to be associated, i.e., $(e_s, e_t) \in \mathcal{R}$, otherwise $b_{(s,t)} = 0$. Hence, an event can be described as an association tuple $\vec{b}_{s,t} = (b_{s,t}, b_{s_1,t}, b_{s_2,t}, b_{s,t_1}, b_{s,t_2}, b_{s_1,t_1}, b_{s_1,t_2}, b_{s_2,t_1}, b_{s_2,t_2})$ of length nine.

A cost model $\gamma : \{0, 1\}^9 \rightarrow \mathbb{R}$ is defined specifying a cost value for each of the 2^9 cophylogenetic events. Among the events there are some events which are isomorphic, i.e., they are identical when changing the child order of s_1 and s_2 , respectively t_1 and t_2 . For instance, the event $\{(e_s, e_t), (e_{s_1}, e_{t_1})\}$ is isomorphic to the event $\{(e_s, e_t), (e_{s_2}, e_{t_2})\}$. Although it is not required, these isomorphic events should have the same cost value and events with no associations at all should be regarded as *null event* and therefore get a value of zero.

Not all events which can be modeled this way are phylogenetically meaningful. If, for example, species s is associated with a child t_j of species t and t with a child species s_i of s , this would immediately result in an inconsistency as the speciation of s had to occur before the speciation of t and vice versa. Therefore, we distinguish between three types of events: i) events with $\tau(s) < \tau(t)$ (denoted as “<” events), ii) events with $\tau(s) > \tau(t)$ (“>” events) and iii) events with $\tau(s) = \tau(t)$ (“=” events). In the first case there must not be an association between s and the descendant species t_i . The second case is equivalently but with no associations between t and s_j . In the last case none of the species s , respectively t , is associated with the descendant species t_j , respectively s_i . The events for case i) and case ii) are depicted in Figure 1. Case ii) is symmetric to the first case with exchanged roles of s and t . Case iii) is shown in Figure 2.

In the algorithmic sections the restricted set of phylogenetically meaningful events is considered only.

3.2 Dynamic Programming

For a given X-tanglegram (S, T, ϕ) and cost model γ a reconciliation \mathcal{R} is sought, such that the sum of all event and leaf-to-leaf association costs is minimal.

In the first step dynamic programming is used to determine all optimal sub-reconciliations with respect to a given cost model. Therefore, two dynamic programming matrices C^0 and C^1 are computed. $C^0(s, t)$ gives the costs for the optimal sub-reconciliation $\mathcal{R}_{|s,t}$ of the two subtrees rooted at s and t with s and t being not associated. Accordingly, $C^1(s, t)$ computes the optimal sub-reconciliation costs with s and t being associated. Starting from a pair of extant species (s, t) the cost of a (non-)association $C^b(s, t)$ is evaluated based on the a priori association strength $\phi(s, t)$. For pairs of ancestral species (s, t) the cost of each possible cophylogenetic event is evaluated, weighted, and accumulated with the costs of the respective sub-reconciliations. From all those costs the minimum is chosen for $C^b(s, t)$. With $b = b_{s,t} \in \{0, 1\}$ the dynamic programming formulation is as follows.

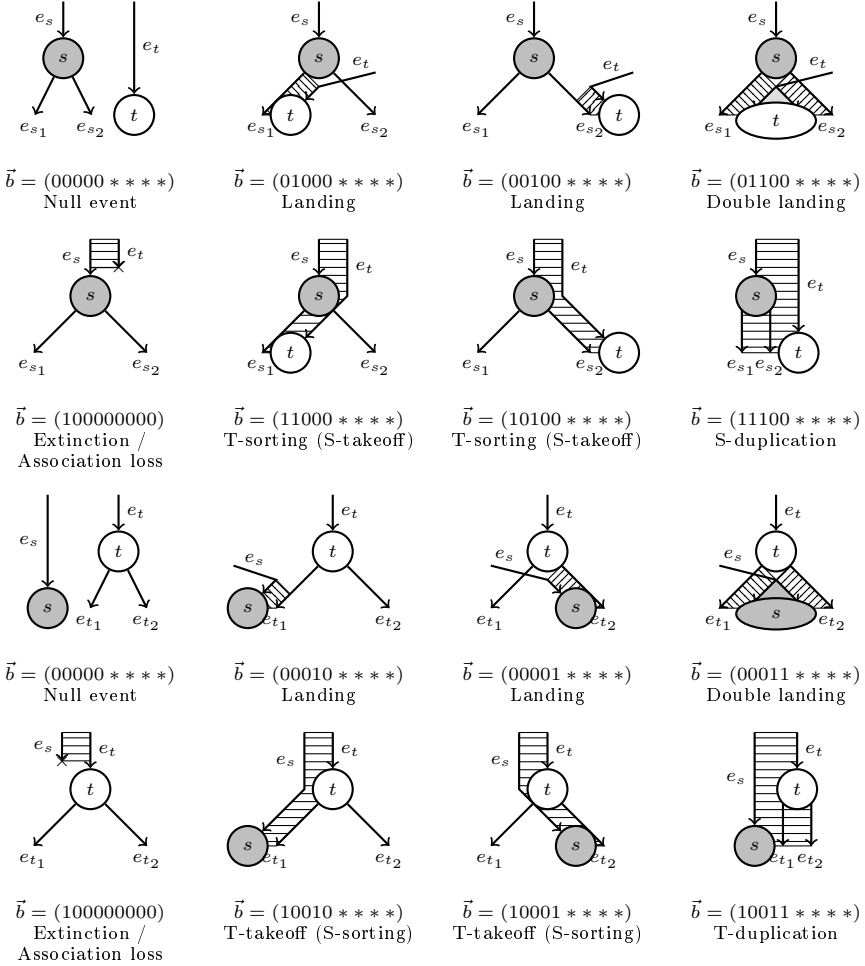


Fig. 1. Coevolutionary events for case i), i.e., speciation of s occurs before the speciation of t , are shown in the upper two rows. Events for case ii), i.e., speciation of s occurs after the speciation of t , are shown in the lower two rows. A '*' in the association tuples represents an arbitrary binary value.

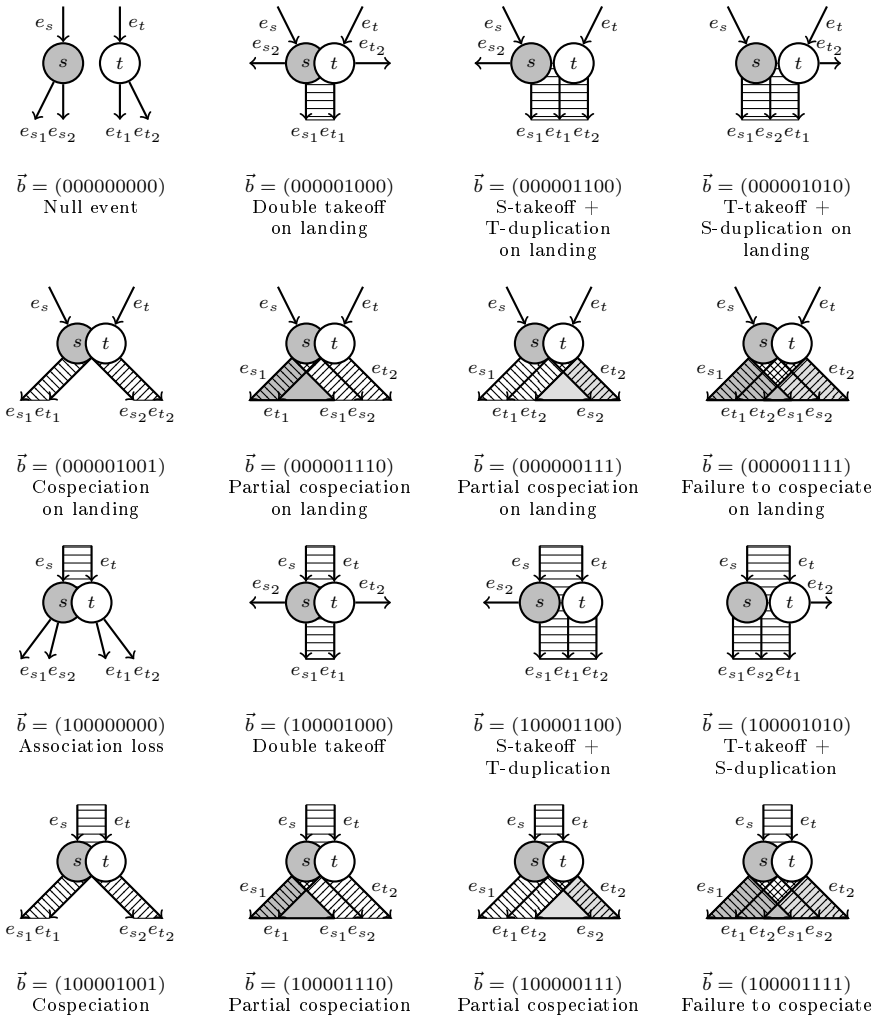


Fig. 2. Coevolutionary events for case iii), i.e., speciation of s and t occurring simultaneously

$$C^b(s, t) = \begin{cases} \alpha(1/\phi(s, t) - 1) & \text{if } s \in L_S, t \in L_T \text{ and } b = 1 \\ \alpha(1/(1 - \phi(s, t)) - 1) & \text{if } s \in L_S, t \in L_T \text{ and } b = 0 \\ C_{<}^b(s, t) & \text{if } s \notin L_S \text{ and } t \in L_T \\ C_{>}^b(s, t) & \text{if } s \in L_S \text{ and } t \notin L_T \\ \min(C_{<}^b(s, t), C_{>}^b(s, t), C_{=}^b(s, t)) & \text{otherwise} \end{cases} \quad (1)$$

with the user defined parameter $\alpha \geq 1$ resulting in a cost value between 0 and ∞ for the leaf-to-leaf associations (division by 0 is evaluated as ∞). $C_{\odot}^b(s, t)$, $\odot \in \{<, >, =\}$ is the cost of the minimal sub-reconciliation with an event of type \odot occurring during the speciation of s and/or t . Precisely:

$$\begin{aligned} C_{<}^b(s, t) &= \min_{b_{s_1, t}, b_{s_2, t} \in \{0, 1\}} \left(\left(\sum_{i \in \{1, 2\}} C^{b_{s_i, t}}(s_i, t) \right) + g^b(\phi(s, t)) \cdot \gamma(\vec{b}_{s, t}) \right) \\ C_{>}^b(s, t) &= \min_{b_{s, t_1}, b_{s, t_2} \in \{0, 1\}} \left(\left(\sum_{j \in \{1, 2\}} C^{b_{s, t_j}}(s, t_j) \right) + g^b(\phi(s, t)) \cdot \gamma(\vec{b}_{s, t}) \right) \\ C_{=}^b(s, t) &= \min_{\substack{b_{s_x, t_y} \in \{0, 1\}, \\ x, y \in \{1, 2\}}} \left(\left(\sum_{i, j \in \{1, 2\}} C^{b_{s_i, t_j}}(s_i, t_j) \right) + g^b(\phi(s, t)) \cdot \gamma(\vec{b}_{s, t}) \right) \end{aligned} \quad (2)$$

$$\text{with } g^0(x) = \begin{cases} \infty & \text{if } x = 1 \\ \beta^{(2x-1)} & \text{otherwise} \end{cases} \quad \text{and } g^1(x) = \begin{cases} \infty & \text{if } x = 0 \\ \beta^{-(2x-1)} & \text{otherwise} \end{cases}.$$

The functions $g^b(x)$, $b \in \{0, 1\}$ give the weighting factors for the event costs $\gamma(\vec{b}_{s, t})$. The user defined parameter $\beta \geq 1$ results in factors ranging from $1/\beta$ to β . Associations with a strength of $x = 0$, respectively non-associations with a strength of $x = 1$, result in infinite costs. The term $(2x - 1)$ is used to normalize $g^b(x)$ to result in a factor of 1 if the association strength is 0.5.

After computing the dynamic programming matrices the cost for an optimal reconciliation is given by $\min(C_{\rho_S, \rho_T}^0, C_{\rho_S, \rho_T}^1)$ with ρ_S and ρ_T being the roots of trees S and T . The reconciliation can be retrieved by backtracking.

3.3 Time Consistency

Although the dynamic programming solutions are optimal with respect to the given cost model γ , the reconciliations might be phylogenetically invalid due to chronological inconsistencies [12]. With the dynamic programming formulation it is assured that for an association of species s and t no descendant of s is associated with an ancestor of t , and vice versa. However, additional timing constraints are introduced by each pairwise association in \mathcal{R} . Assume two species s and t being associated and therefore $(e_s, e_t) \in \mathcal{R}$. As species s and t interacted, both existed at the same time. Hence, species s has to be emerged before the speciation of t and vice versa and it follows that $\forall s' <_T s : \tau(s') < \tau(t)$, respectively $\forall t' <_T t : \tau(t') < \tau(s)$, see Figure 3 (a). This is not assured by the dynamic programming formulation. Contradicting associations of species from disjoint subtrees of both phylogenies can occur. For an example consider the species s_1 , s_2 , t_1 , and t_2 from disjoint subtrees of S , respectively T , with

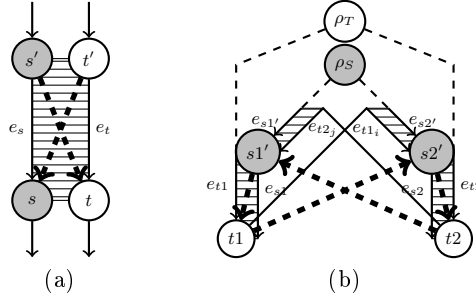


Fig. 3. (a) The association between the two edges e_s and e_t results in the two timing constraints $\tau(s') < \tau(t)$ and $\tau(t') < \tau(s')$ (dashed arrows). (b) The four associations (e_{s1}, e_{t1}) , (e_{s2}, e_{t2}) , $(e_{s1'}, e_{t2_j})$ and $(e_{s2'}, e_{t1_i})$ result in circular timing constraints (cycle of dashed arrows).

$\{(e_{s1}, e_{t1}), (e_{s2}, e_{t2})\} \subseteq \mathcal{R}$. Furthermore, let the species $s1'$ and $s2'$ be the parent species of $s1$, respectively $s2$. Now assume that $s1'$ is associated with the child species $t2_j$ of $t2$ and $s2'$ is associated with the child species $t1_i$ of $t1$, i.e., $\{(e_{s1'}, e_{t2_j}), (e_{s2'}, e_{t1_i})\} \subseteq \mathcal{R}$. This scenario creates (among others) the timing constraints $\tau(s1') < \tau(t1) < \tau(s2') < \tau(t2) < \tau(s1')$ which gives a timing inconsistency. Figure 3 (b) shows this example. For a reconciliation to be phylogenetically meaningful such cases must not occur. To determine phylogenetically valid reconciliations the following two definitions are needed.

Definition 1 (Timing Graph). For a given X -tanglegram (S, T, ϕ) and a reconciliation \mathcal{R} the timing graph TG is a directed graph (V_{TG}, E_{TG}) with node set $V_{TG} = V_S \cup V_T$ and edge set $E_{TG} = E_S \cup E_T \cup \{(s', t), (t', s) : (e_s, e_t) \in \mathcal{R} \text{ and } (s', s) \in E_S \text{ and } (t', t) \in E_T\}$.

Definition 2 (Time Consistency). A reconciliation \mathcal{R} for an X -tanglegram (S, T, ϕ) is said to be time consistent if its timing graph is acyclic.

Observe that there are event models where the dynamic programming will always create time consistent reconciliations which are parsimonious with respect to the total event costs. Chronological inconsistencies are created by contradicting timing constraints. But the set of timing constraints coming from the tree topologies of both trees without any associations are always compatible. Therefore, for an inconsistency to occur there must be paths in the timing graph connecting nodes of S , respectively T , which have no ancestor/descendant relation, i.e., nodes from distinct subtrees of the same phylogenetic tree. Such constraints are created by either S-duplications (only between nodes from S) and T-duplications (only between nodes from T) or by any of the landing events (constraints between nodes from both trees). A chronological inconsistency can be created only if combinations of such constraints between both types of trees exist. Therefore, any event model which forbids landing events and allows only one type of duplication (S or T) will lead to time consistent reconciliations.

For instance the gene duplication/gene loss event model commonly used for gene tree/species tree reconciliation can be solved without chronological inconsistencies using dynamic programming.

But even if events are considered which possibly lead to chronological inconsistencies, dynamic programming can be used to construct valid solutions as well. This is done by running dynamic programming and checking the timing graph for cycles. In each cycle there exist edges which were introduced due to associations between species. In an iterative way one can restrict the input data by forbidding some of the associations and redo the dynamic programming until the timing graph is acyclic. However, the produced reconciliation may not be parsimonious anymore, see also [22].

3.4 Branch-and-Bound Algorithm

Although the (unrestricted) dynamic programming does not necessarily result in a time consistent reconciliation, it can be used to determine a lower bound for the costs of a partially computed reconciliation. This lower bound is used by a branch-and-bound algorithm to cut the computation whenever a partial solution indicates that the cost will be higher than a certain value, e.g., the cost of a previously found reconciliation or a maximum threshold.

The algorithm starts with an empty set of associations and a lower bound of $\min(C_{\rho_S, \rho_T}^0, C_{\rho_S, \rho_T}^1)$ for the reconciliation costs. In the branching procedure of the algorithm a decision tree is traversed by considering in each step a pair of species, i.e., edges, as either associated or not associated. In this way a partial reconciliation is constructed adding at most one association per branching step. The pairs are selected in a top-down manner, starting from (ρ_S, ρ_T) , such that it is assured that when selecting a pair (s, t) all pairs of ancestral species (s', t') , with $s' \prec_S s$ and $t' \prec_T t$, were already processed beforehand. For each selection the timing graph is updated and checked for cycles. If the graph contains any cycles the computation is cut at this point and the next pair of species is selected.

As a cophylogenetic event consists of multiple associations, all the respective pairs have to be processed before an event, and therefore its cost, can be determined. If this is the case then the cost of the respective event is added to the cost of the already computed partial reconciliation. The lower bound for the total costs is then given by the costs of the partial reconciliation plus the minimum costs needed for the sub-reconciliations of the unprocessed pairs of subtrees. These minimum costs are taken from the two dynamic programming matrices C^0 and C^1 .

To avoid unnecessary branching after choosing two species s and t as being associated, all pairs (s', t'') and (s'', t') with $s' \prec_S s \prec_S s''$ and $t' \prec_T t \prec_T t''$ are assumed to be not associated, as this would result in timing inconsistencies. The pseudocode is given in Algorithm 1.

Beside timing constraints, some coevolutionary systems require that additional properties have to be satisfied for a reconciliation to be valid. There may be systems which require, e.g., that each species of one type is associated with at least, respectively at most, one species of the other type at a time. For instance

in gene tree/species tree reconciliations each (ancestral) gene is associated with exactly one species. As these constraints restrict the set of valid solutions they can be easily integrated into the branch-and-bound algorithm. In each branching step it is checked if the current association will lead to a violation and, if applicable, the computation can be cut.

Algorithm 1. Pseudocode for computing the minimal sum of event costs for the cophylogenetic reconciliation of S and T

Input: $pairs$: the queue of unprocessed pairs - $[(\rho_S, \rho_T)]$ at first call;
 $A[\square]$: the association matrix; $cost_{partial}$: the costs of the partial reconciliation;
 $cost_{bounds}$: the lower bound for the costs of unprocessed sub-reconciliations;
 $cost_{best}$: cost of current best solution or infinity/max threshold;
 TG: the timing graph; the trees S and T and DP matrices C^0 and C^1 ;
if $pairs$ is not empty **then**

- get next unprocessed pair (s, t) from $pairs$;
- add all pairs $(s, t_j), (s_i, t), (s_i, t_j)$ with $i, j \in \{1, 2\}$ to $pairs$;
- foreach** b in $\{\text{"not-associated"}, \text{"associated"}\}$ **do**
 - if** $A[s][t]$ is undefined or equals b **then**
 - set $A[s][t]$ to b ;
 - if** b equals "associated" **then**
 - foreach** s', s'', t', t'' with $s' <_T s <_T s''$ and $t' <_T t <_T t''$ **do**
 - $A[s'][t''] \leftarrow \text{"not-associated"}$; $A[s''][t'] \leftarrow \text{"not-associated"}$;
 - update timing graph TG;
 - if** timing is consistent **then**
 - if** a new event can be determined from $A[\square]$ **then**
 - update $cost_{partial}$; update $cost_{bounds}$;
 - if** $cost_{partial} + cost_{bounds} \leq cost_{best}$ **then**
 - $cost_{total} \leftarrow$ result of recursive call to Alg. 1;
 - $cost_{best} \leftarrow \min(cost_{best}, cost_{total})$;
 - undo changes to TG, $A[\square]$, $cost_{partial}$, and $cost_{bounds}$;

return $cost_{best}$

4 Discussion

A certain type of application usually requires only a subset of the events modeled by this approach. For each event from this subset a cost value has to be specified. All neglected events get infinite costs and will therefore not occur in reconciliations with finite overall costs. Obviously there is a 1-to-1 correspondence between most of the cophylogenetic events depicted in Table 1 and the events defined by local association patterns shown in Figures 1 and 2. Only the two types of *switches* (partial and complete) and the *lateral transfer from the dead* can not be modeled directly by this approach. Instead, a *partial switch*

is seen as a combination of the events *takeoff* and *landing*, while a *complete switch* and a *lateral transfer from the dead* are modeled by an *extinction* and a *landing*. When giving the *takeoff* and *landing* events, respectively *extinction* and *landing* events, the same accumulated costs, then a certain reconciliation results in the same overall costs in both event models. This opens up new possibilities for solving reconciliation problems for a variety of applications. Biogeography, gene tree/species tree, and host-parasite systems can be reconciled with the same algorithms while only the cost model γ differs. Beside that, further cases of application exists, e.g., general symbiotic systems or interactions of genes or gene products, where both association partners are equitable and a reconciliation can not be produced by simply embedding one tree into the other.

In this approach we decided a reconciliation to be a set of associations between edges. This has been done to retain a similar graphical representation for the cophylogenetic events as it has been used in previous publications. However, as more complex events can be modeled by this approach, e.g., the *partial cospeciation*, visualizing a reconciliation is a complex task and the interpretation of the results might be hard. An alternative is to redefine a reconciliation to be a set of associations between nodes on the directed line graphs (line digraphs) of the phylogenetic trees with artificial root ρ' . Both definitions are interchangeable as in the line graph only the roles of nodes and edges are changed but the tree structure is retained. Then a reconciliation can be depicted as a graph with directed *tree edges* representing the tree topology of both trees and undirected *association edges*.

The dynamic programming algorithm has a time complexity of $O(n^2)$, where n is the number of leaves in the larger phylogenetic tree. For each of the $O(n^2)$ many values from the two matrices $C_{<}^b(s, t)$, $b \in \{0, 1\}$, the cost of a constant number of possible association patterns is determined. For each pattern this can be done in constant time.

5 Conclusion

in this paper we introduced a cophylogenetic event model based on local association patterns between coincident edges. due to the large variety of possible events and the possibility to neglect events via a corresponding cost model the approach suits to various reconciliation problems. in addition we provided the possibility to use a priori knowledge about association strengths to improve the reconciliations. we presented a $o(n^2)$ time heuristic based on dynamic programming as well as an exact branch-and-bound algorithm to solve the reconciliation problem for a given x-tanglegram (s, t, ϕ) and cost model γ .

Until now, only binary phylogenetic trees are considered. But extending the approach to support polytomies can be done by extending the event model to non-binary events. However, this requires a fixed maximal outdegree for the internal nodes and therefore can not be used for multifurcations of arbitrary size. Alternatively, the polytomies can be resolved within the reconciliation process using heuristic approaches.

In recent years there is a trend towards using also phylogenetic networks. These networks can display hybridization events, as nodes with indegree greater than one. While extending our event model to consider cophylogenetic hybridization is straightforward, further research is needed to adopt the reconciliation algorithms.

Acknowledgements. This work was funded by the German Research Foundation (DFG) through the project MI439/14-1.

References

1. Berglund-Sonnhammer, A.C., Steffansson, P., Betts, M., Liberles, D.: Optimal gene trees from sequences and species trees using a soft interpretation of parsimony. *J. Mol. Evol.* 63(2), 240–250 (2006)
2. Brooks, D.: Parsimony analysis in historical biogeography and coevolution: methodological and theoretical update. *Syst. Biol.* 39(1), 14–30 (1990)
3. Charleston, M.: Jungles: a new solution to the host/parasite phylogeny reconciliation problem. *Math. Biosci.* 149(2), 191–223 (1998)
4. Charleston, M., Perkins, S.: Traversing the tangle: algorithms and applications for cophylogenetic studies. *J. Bio. Med. Inform.* 39(1), 62–71 (2006)
5. Chen, K., Durand, D., Farach-Colton, M.: NOTUNG: a program for dating gene duplications and optimizing gene family trees. *J. Comput. Biol.* 7(3–4), 429–447 (2000)
6. Conow, C., Fielder, D., Ovadia, Y., Libeskind-Hadas, R.: Jane: a new tool for the cophylogeny reconstruction problem. *Algorithms Mol. Biol.* 5, 16 (2010)
7. Doyon, J.P., Scornavacca, C., Gorbunov, K.Y., Szölloši, G., Ranwez, V., Berry, V.: An efficient algorithm for gene/species trees parsimonious reconciliation with losses, duplications and transfers (2010)
8. Goodman, M., Czelusniak, J., Moore, G., Romero-Herrera, A., Matsuda, G.: Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. *Syst. Biol.* 28(2), 132–163 (1979)
9. Hafner, M.S., Nadler, S.A.: Phylogenetic trees support the coevolution of parasites and their hosts. *Nature* 332(6161), 258–259 (1988)
10. Hafner, M., Nadler, S.: Cospeciation in host-parasite assemblages: comparative analysis of rates of evolution and timing of cospeciation events. *Syst. Biol.* 39(3), 192–204 (1990)
11. Hendy, M., Little, C., Penny, D.: Comparing trees with pendant vertices labelled. *Siam J. Appl. Math.* 44(5), 1054–1065 (1984)
12. Merkle, D., Middendorf, M.: Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information. *Theory Biosci.* 123(4), 277–299 (2005)
13. Merkle, D., Middendorf, M., Wieseke, N.: A parameter-adaptive dynamic programming approach for inferring cophylogenies. *BMC Bioinformatics* 11(S-1), 60 (2010)
14. Nelson, G., Ladiges, P.Y.: Three area statements: standard assumptions for biogeographic analysis. *Syst. Zool.* 40, 470–485 (1991)
15. Nelson, G., Platnick, N.: *Systematics and biogeography: cladistics and vicariance*. Columbia University Press (1981)

16. Ovadia, Y., Fielder, D., Conow, C., Libeskind-Hadas, R.: The co-phylogeny reconstruction problem is NP-complete. *J. Comput. Biol.* 18(1), 59–65 (2011)
17. Page, R.D.M.: *Tangled Trees. Phylogeny, Cospeciation and Coevolution.* The University of Chicago Press (2003)
18. Page, R.: Quantitative cladistic biogeography: constructing and comparing area cladograms. *Soc. Syst. Zool.* (1988)
19. Page, R.: Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Syst. Biol.* 43(1), 58–77 (1994)
20. Page, R.D.M.: GeneTree: comparing gene and species phylogenies using reconciled trees. *Bioinformatics* 14(9), 819–820 (1998)
21. Page, R., Charleston, M.: Trees within trees: phylogeny and historical associations. *Trends Ecol. Evol.* 13(9), 356–359 (1998)
22. Patro, R., Sefer, E., Malin, J., Marçais, G., Navlakha, S., Kingsford, C.: Parsimonious reconstruction of network evolution. *Algorithms Mol. Biol.* 7(1), 25 (2012)
23. Ronquist, F.: Dispersal-vicariance analysis: a new approach to the quantification of historical biogeography. *Syst. Biol.* 46(1), 195–203 (1997)
24. Ronquist, F., Nylin, S.: Process and pattern in the evolution of species associations. *Syst. Biol.* 39(4), 323–344 (1990)
25. Ronquist, F., Sanmartín, I.: Phylogenetic methods in biogeography. *Annu. Rev. Ecol. Evol. Syst.* 42, 441–464 (2011)
26. Szollosi, G.J., Tannier, E., Lartillot, N., Daubin, V.: Lateral gene transfer from the dead. *Syst. Biol.* 62(3), 386–397 (2013)
27. Vernot, B., Stolzer, M., Goldman, A., Durand, D.: Reconciliation with non-binary species trees. *J. Comput. Biol.* 15(8), 981–1006 (2008)
28. Zandee, M., Roos, M.C.: Component-compatibility in historical biogeography. *Cladistics* 3, 305–332 (1987)
29. Zmasek, C., Eddy, S.: A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics* 17(9), 821–828 (2001)

Sibelia: A Scalable and Comprehensive Synteny Block Generation Tool for Closely Related Microbial Genomes

Ilya Minkin, Anand Patel, Mikhail Kolmogorov, Nikolay Vyahhi,
and Son Pham

Department of Computer Science and Engineering, UCSD, La Jolla, CA, USA
St. Petersburg Academic University, St. Petersburg, Russia

Abstract. Comparing strains within the same microbial species has proven effective in the identification of genes and genomic regions responsible for virulence, as well as in the diagnosis and treatment of infectious diseases. In this paper, we present Sibelia, a tool for finding synteny blocks in multiple closely related microbial genomes using *iterative de Bruijn graphs*. Unlike most other tools, Sibelia can find synteny blocks that are repeated within genomes as well as blocks shared by multiple genomes. It represents synteny blocks in a hierarchy structure with multiple layers, each of which representing a different granularity level. Sibelia has been designed to work efficiently with a large number of microbial genomes; it finds synteny blocks in 31 *S. aureus* genomes within 31 minutes and in 59 *E.coli* genomes within 107 minutes on a standard desktop. Sibelia software is distributed under the GNU GPL v2 license and is available at: <https://github.com/bioinf/Sibelia>. Sibelia's web-server is available at: <http://etool.me/software/sibelia>.

1 Introduction

Early in the genomic era, sequencing a single representative isolate was thought to be sufficient to describe the genetics of a microbial species, and due to computational and technological limitations, *comparative genomics* was restricted to comparing closely related microbial species. However, outbreaks of virulent forms of common microbes (e.g. *Escherichia coli O157:H7*) and multidrug-resistant bacterial strains (e.g. TB, MRSA) have intensified efforts to understand genetic diversity among microbial isolates belonging to the same species.

The task of decomposing genomes into non-overlapping highly conserved segments called *synteny blocks* has proven to be important in genome comparison. It has been applied to finding structural variations between genomes [10,6] and is also a prerequisite in most genome rearrangement software. Additionally, finding highly conserved regions shared among many strains within the same microbial species helps to infer the minimum genomic material (or core genome) required for bacterial life and thus can be useful for the Minimal Genome Project [9].

Finding synteny blocks in multiple microbial genomes presents the following four challenges. (1) Genomes of strains belonging to the same species differ by

point mutations, small-/large indels, small-/large-scale rearrangements and duplications. (2) With the current deluge of microbial genomes, genome comparison tools face the problem of comparing hundreds or even thousands of genomes simultaneously. A number of synteny block generation tools exist [2,3,8,18], but most of them require the calculation of local alignments between all pairs of genomes. As the number of genomes increases, the number of required pairwise comparisons quickly becomes a bottleneck in terms of total computational time. (3) The task of synteny block reconstruction has been heavily dependent on parameters, which determine the size and granularity (coarse-grained or fine-grained) of the resulting synteny blocks [19]. Different applications favor different scales of synteny block reconstruction. For instance, while most current ancestral genome reconstruction software [1] favors large-scale synteny blocks, the analysis of virulence factors in pathogen genomes considers both small- (transposons, insert elements) and large-scale synteny blocks to be important [4]. Thus, general synteny block generation software should be able to find and represent synteny blocks in multiple resolutions. (4) Synteny block software designed for a large number of bacterial genomes should be able to work directly with a high volume of unannotated genome sequences (represented in the alphabet of nucleotides) rather than with annotated genomes (represented in the alphabet of genes). This is because the accumulation of errors in gene annotation¹ can grow substantially as the number of genomes increases.

By concatenating multiple sequences into a highly repetitive “virtual genome”, Peng et al. [16] noticed that the problem of constructing synteny blocks from multiple genomes is equivalent to the problem of *de novo* repeat classification in the “virtual genome”, and they utilized A-Bruijn graphs [17] for synteny block reconstruction.

Since repeats are inexact, A-Bruijn graph frameworks require an initial step of graph simplification to determine the consensus of repeats, and later, *threading* the genome through the simplified graph to determine the positions of repeats. The threading procedure is usually problematic, and Peng et al. [16] concluded that threading is a major bottleneck in synteny block reconstruction. Pham and Pevzner [18] introduced the first A-Bruijn graph approach (DRIMM-Synteny) that does not require the threading step by using a *sequence modification algorithm*. Rather than simplifying the graph, this approach modifies the sequence so that its corresponding graph is simplified, and thus completely bypasses the threading procedure. As a result, the sequence modification procedure returns a sequence that is modified so that its A-Bruijn graph reveals synteny blocks as non-branching paths.

DRIMM-Synteny was designed to work with mammalian genomes, but it faces the roadblock of constructing synteny blocks for large numbers of bacterial genomes because it takes as input a set of genomes represented in the alphabet of genes. Adapting DRIMM-Synteny to work with raw DNA sequences faces many computational challenges described above.

¹ Errors in gene annotation can also be caused by using different software to annotate different genomes.

Even when computational resources do not pose a problem, DRIMM-Synteny and most other current synteny block generation tools do not present synteny blocks in order to satisfy many different applications, namely, presenting synteny blocks in multiple granularity levels (i.e., different resolutions).

Indeed, repeats in the “virtual genome”, which are obtained by the concatenation of dozens to hundreds of simple bacterial genomes, are both multi-scale (i.e., multiple size) and multi-granular, since repeats in the virtual genome are accounted for not only by repeated blocks within each bacterial genome, but also by blocks shared among multiple bacterial genomes. Whereas the repeat size within each bacterial genome can range from dozens to several thousands of bp, regions conserved among different genomes can have an even wider range of sizes; some of these regions may even reach several Mbp in size, and they usually contain sub-repeats. For that matter, repeats are not exact, and the longer the repeat, the more likely it is disrupted by other smaller insertions/deletions. Thus, repeats of different sizes usually have different granularities.

While the de Bruijn graph has offered the best model for representing perfect repeats in a simple genome [17], we argue that a single de Bruijn graph is not sufficient to capture the complicated repeat structure of virtual genomes (obtained by concatenating dozens to hundreds of simple genomes), which are both multi-scale and multi-granular. In this work, we propose an iterative de Bruijn graphs algorithm, which uses multiple de Bruijn graphs constructed from different values of k to capture the complicated repeat structure of virtual genomes. Our iterative de Bruijn graphs algorithm allows us to construct synteny blocks and represent them in a hierarchy structure. Large-scale (coarse-grained) synteny blocks can be further decomposed into multiple layers, where each layer represents a different granularity level.

Our algorithm has been developed into Sibelia software, which offers a tool for decomposing multiple closely related microbial genomes into synteny blocks. Sibelia has three special properties: (1) Sibelia is able to reveal synteny blocks repeated within genomes as well as blocks shared simultaneously by many genomes (repeats within genomes are usually problematic for most current tools). (2) Sibelia represents synteny blocks in a hierarchical structure. (3) Sibelia is fast: it analyzes 31 *S. aureus* genomes within 31 minutes and 59 *E. coli* genomes within 107 minutes on a standard desktop.

2 Methods

For simplicity of exposition, we assume that the given set of genomic sequences is concatenated (using delimiters) into a single “virtual” genome and consider the problem of finding syntenic (repeated) blocks within this *highly duplicated genome*. From now on, we will use the terms *repeated block* and *synteny block* interchangeably.

2.1 De Bruijn Graph and Cycles

Let a genome of length n be represented as a circular string $S = s_1 \dots s_n$ over the nucleotide alphabet $\{A, T, C, G\}$. A k -mer is a string of length k . The de Bruijn graph $DB(S, k)$ represents every k -mer in S as a vertex and connects two vertices by a directed edge if they correspond to a pair of consecutive k -mers in the genome (these two k -mers overlap in a shared $(k - 1)$ -mer). The de Bruijn graph can be viewed as both a multigraph (i.e., adjacent vertices can be connected by multiple edges) and a weighted graph with the multiplicity of an edge (a, b) defined as the number of times that the k -mers a and b appear consecutively in S .

Alternatively, the de Bruijn graph of a string S can be defined by a *gluing operation* (see [18,13] for a formal definition of this operation): represent S as a sequence of vertices $1, \dots, n$ with $n - 1$ edges $i \rightarrow (i + 1), 1 \leq i \leq n - 1$; label each vertex i by the k -mer starting at position i in S ; *glue* two vertices together if they have the same label (See Fig. 1 for de Bruijn graphs constructed from DNA strings).

Given a value of k and a sequence S , perfectly repeated regions of size larger than k in S are *glued* into paths in its de Bruijn graph $DB(S, k)$. Perfectly repeated regions that do not share any k -mer with other regions correspond to non-branching paths, which are maximal paths in the graph satisfying the condition that all their internal vertices have only two neighboring vertices. The multiplicity of a path is equal to the number of times that the corresponding region appears in S .

One issue with using de Bruijn graphs for repeat analysis is that de Bruijn graphs constructed from real genomes have many short cycles and “hide” the genome’s repeat structure. Cycles in de Bruijn graphs are commonly classified into two types: bulges (Fig. 1d) and loops (Fig. 1f). Intuitively, bulges are caused by mismatches/indels between two homologous sequences, and loops are caused by closely located k -mer repeats. To reveal repeats in de Bruijn graphs, these small cycles should be removed. To avoid the threading procedure, which is usually problematic, we adopt a sequence modification approach to remove bulges in the de Bruijn graphs. As will be made obvious in the next subsection, Sibelia does not need to explicitly remove loops, but focuses only on bulges. The reason behind this is that merely increasing the value of k can help to eliminate small loops in the de Bruijn graph. Fig. 2c shows a loop, which is caused by a closely located 3-mer repeat (*ATC*). The de Bruijn graph with larger vertex size $k = 4$ does not have a loop. Below, we formulate **SMP-B**: the sequence modification problem for removing bulges in de Bruijn graphs.

We say that a string P *covers* an edge $a \rightarrow b$ in $DB(S, k)$ if a and b are consecutive k -mers in P . A cycle C in $DB(S, k)$ is classified as a bulge if all its edges can be covered by two non-overlapping substrings P_1, P_2 of S , and P_1, P_2 do not cover any edges in $DB(S, k)$ except for those in C .

SMP-B: *Given a string S and parameters \mathbf{C}, k . Find a string S' with minimum edit distance $d(S, S')$ such that $DB(S', k)$ has no two-way cycle shorter than \mathbf{C} .*

Since the complexity of this problem remains unknown, we apply the *sequence modification algorithm* [18], a heuristic algorithm for removing bulges.

Sequence Modification Algorithm. Let C be a bulge with total number of edges smaller than \mathbf{C} , formed by substrings P_1 and P_2 of S . To remove the bulge, the algorithm modifies S by substituting all occurrences of P_1 in S by P_2 . Fig. 2a shows the de Bruijn graph for $S = ATCGGTTAACT\dots ATCGATCAACT$, with two inexact repeats. Minor differences between these two repeat instances form a bulge having two branches (colored red and blue in the figure). By changing S into $S' = ATCGGTTAACT\dots ATCGGTTAACT$ (i.e., substituting the blue branch with the red branch), the bulge is also simplified. Note that S' now contains an exact repeat of multiplicity 2 (Fig. 2b).

2.2 Effects of k -Mer Size and Bulge Removal Procedure in Repeat Decomposition

In this subsection, we give a relationship between repeats (revealed by non-branching paths) in de Bruijn graphs constructed with different values of k as well as repeats revealed by non-branching paths before and after the bulge simplification procedure.

Effects of k -mer Size

Observation 1. Let k_0, k_1 be two positive integers such that $k_0 < k_1$, and let S be a cyclic genome. Furthermore, let G_0 and G_1 be the de Bruijn graphs constructed from S with $k = k_0$ and $k = k_1$, respectively. Any repeat R revealed by a non-branching path in G_1 can be decomposed into a sequence of sub-repeats, each corresponding to a non-branching path in G_0 .

See Appendix for a formal description of this observation. Intuitively, when reducing the value of k -mer from k_1 to k_0 , some different non-branching paths in G_1 having shared k_0 -mers will interfere with each other (these common k_0 -mers play as new “glues” in the graph), thus, fragment large non-branching paths into shorter ones. The above observation allows us to decompose any non-branching path of G_1 into a sequence of non-branching paths in G_0 . In other words, repeats revealed by any non-branching path from one de Bruijn graph can be decomposed into a sequence of repeats (smaller sub-blocks) revealed by another de Bruijn graph constructed from a smaller value of k .

Effects of Bulge Simplification

Observation 2. Bulge simplification can be viewed as the process of “merging” consecutive non-branching paths in the graph. Fig. 2a shows a bulge, which breaks the red segment into 3 segments, each corresponds to a non-branching path. Fig. 2b shows the same graph after collapsing the bulge. Thus, the red large repeat can be decomposed into 3 sub-segments; each of the sub-segments corresponding to a non-branching path in the graph before simplification ($ATCGGTTAACT$ is decomposed into $(ATCG)$, (GTT) , $(AACT)$ according to Fig. 2).

These two observations are important for representing synteny blocks with different scales, which will be described in later sections.

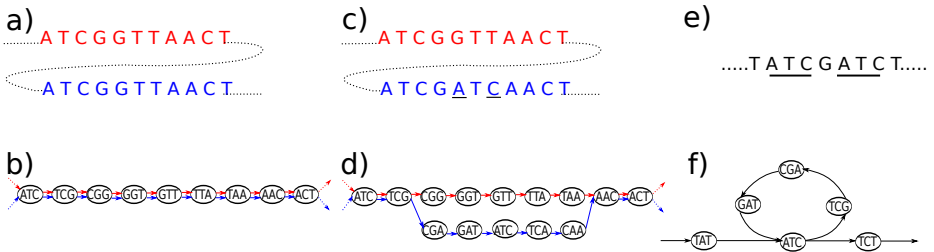


Fig. 1. De Bruijn graphs and cycles in de Bruijn graphs. a) A sequence $S = \text{ATCGGTTAACT...ATCGGTTAACT}$ with the segment ATCGGTTAACT repeated twice. b) The de Bruijn graph of the sequence in a). c) Sequence with inexact repeats: the green segment represents an inexact repeat of the red segment. Positions with different nucleotides are underlined. d) De Bruijn graph of the sequence in c). Minor differences between these two segments (colored red and blue) are reflected by a bulge with two branches (the red and the blue branches of the bulge). e) A sequence with a closely located repeated k -mer ATC . f) The de Bruijn graph of the sequence in e). The closely located repeated k -mer (ATC) forms a loop in the graph.

Challenges in Using de Bruijn Graphs in Repeats and Synteny Analysis

Finding repeated blocks in a genome using de Bruijn graphs faces two problems. (1) Even in the case that synteny blocks are perfectly repeated, they can have very different lengths. Representing both small (e.g., insertional elements, transposons) and large blocks in the de Bruijn graph is difficult because if k is set equal to a large value, then repeats of size smaller than k can not be revealed, while using a small value of k may introduce additional gluings on large repeats. While such additional gluings may help to reveal mosaic structures (subrepeats within a larger repeat) of repeats, they also hide repeats when gluing is excessive. (2) Synteny blocks often contain many mismatches and gaps, which restrict the use of large values of k when constructing the de Bruijn graph.

The first challenge motivates us to use different values of k for representing repeats with different sizes. The second challenge can be resolved by using the *sequence modification algorithm* [18] to remove bulges, since sequence modification will eliminate mutations, gaps, and indels between homologous blocks and thus can help to increase the value of k . We propose the *iterative de Bruijn graph algorithm* as follows.

Initially, the algorithm constructs the de Bruijn graph from a relatively small value of $k = k_0$ and performs graph simplification with a small cycle length threshold (C_0). The algorithm operates on the de Bruijn graph $G_0(S_0, k_0)$ and simplifies all bulges using the sequence modification approach described above. As a result, we obtain a simplified de Bruijn graph G_1 and the corresponding modified genome S_1 . S_1 is a distorted version of S such that its de Bruijn graph $DB(S_1, k_0)$ does not contain short bulges of length smaller than C_0 . We note that

graph simplification should be applied using the *sequence modification algorithm*; otherwise, S_1 , the distorted version of S_0 , is not available for the construction of the graph using a larger value of $k = k_1$. The goal of the first iteration is to collapse bulges caused by single point mutations or very short indels. Thus, we can increase the value of k and construct a new de Bruijn graph $G_1 = DB(S_1, k_1)$, where $k_1 > k_0$. The process continues until we reach a value of k that is large enough to reveal large-scale synteny blocks (pseudo code of Sibelia is described in Algorithm 1). Generally speaking, the iterative process should continue until the genome is presented as a single synteny block. This argument may appear unreasonable at first sight, as our goal was to decompose genomes into synteny blocks. However, one should notice that the two observations in the above subsection allow us to retrace our steps to find previous synteny blocks.

2.3 Hierarchical Representation of Synteny Blocks

Sibelia works iteratively as follows: from a sequence S_{i-1} , it constructs a de Bruijn graph $G_i = DB(S_{i-1}, k_i)$ and removes bulges to obtain a simplified graph $G_i^+ = DB(S_i, k_i)$. It then reconstructs the de Bruijn graph for larger $k = k_{i+1}$: $G_{i+1} = DB(S_i, k_{i+1})$. Using Observation 1, each non-branching path in G_{i+1} can be decomposed into a sequence of non-branching paths in G_i^+ because $k_{i+1} > k_i$. Since we only perform bulge simplification at each stage, each non-branching path in G_i^+ can in turn be decomposed into a sequence of non-branching paths in G_i . Therefore, each non-branching path in the last stage can be represented as the root of a tree, where its children are the decompositions of the previous stage. The leaves of the tree represent non-branching paths in the de Bruijn graph constructed from the smallest value k_0 (See Fig. 4 for the hierarchy representation of synteny blocks in two strains of *H. pylori*). Under this decomposition, each chromosome can be considered as a single synteny block, which can be further decomposed into multiple large-scale “synteny blocks”. Each large-scale synteny block can be further decomposed into smaller-scale synteny blocks. The process of decomposition continues until we reach the synteny blocks revealed by the graph $DB(S, k_0)$.

Parameter Choices. It appears that the iterative de Bruijn graph algorithm depends on many parameters: (1) the number of iterations; (2) for each iteration i , k_i (k -mer size) and C_i (cycle length for *bulge simplification*). However, we notice that the most important parameters determining the outcome² of synteny block reconstruction are the values of parameters in the first iteration: k_0 and C_0 . The reason for this is that in microbial genomes, a point mutation event is the most common, and large indels occur at a much lower rate [12]. In latter stages I_i , ($i > 0$), k_i and C_i reflect the size of repeated blocks and the granularity of synteny blocks in that stage. While the choices of k_0 and C_0 require a careful analysis of the evolutionary distance between genomes (See Appendix, Section

² If the total coverage of synteny blocks for any set of genomes within the same species is smaller than 40%, then we classify the synteny block construction as unsuccessful according to the analysis of core genomes in [7].

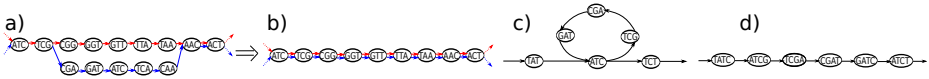


Fig. 2. (a) De Bruijn graph of a sequence with two inexact repeats $S = ATCGGTTAACT\dots ATCGATCAACT$. The minor differences in the inexact repeats form a bulge with two branches: The red branch $(TCG) \rightarrow (CGG) \rightarrow (GGT) \rightarrow (GTT) \rightarrow (TTA) \rightarrow (TAA) \rightarrow (AAC)$ and the blue branch $(TCG) \rightarrow (CGA) \rightarrow (GAT) \rightarrow (ATC) \rightarrow (TCA) \rightarrow (CAA) \rightarrow (AAC)$. (b) We simplify the graph by changing the sequence from ATCGATCAACT to ATCGGTTAACT, thus forming an exact repeat. The modified sequence corresponds to a non-branching path on the de Bruijn graph. (c) A closely located repeated k -mer (ATC) forms a loop in the graph. (d) Increasing the value of k can help to resolve the loop in c).

1), latter iterations can be seen as according users the flexibility to choose granularity as well as the size of blocks in both the final and intermediate stages (See Fig. 4 for a hierarchical representation of synteny blocks in two strains of *H. pylori*).

Similar to the analysis in [5], we derive the values for (k_0, C_0) to be $(30, 150)$ for any set of microbial strains within the same species (see Appendix, Section 1 for a more detailed analysis). Sibelia’s default mode has 4 stages (iterations) with the following parameters: $((30, 150), (100, 1000), (1000, 5000), \text{ and } (5000, 15000))$, where each pair of values corresponds to (k_i, C_i) in the corresponding stage. While the pair of parameters for the first stage was derived using the sequence similarity of genomes within the same microbial species (see Appendix, Section 1), the final 3 stages ($k = 100, 1000, 5000$) were designed to capture small repeats of length equal to several hundred bp ($k = 100$), transposons, insertion elements with average length about 1 Kbp [14] ($k = 1000$), and large-scale blocks (usually comprising several genes) that are among multiple genomes ($k = 5000$). Users can add more iterations between any consecutive stages to obtain a “smoother” decomposition between stages.

Algorithm 1. Iterative de Bruijn Graph

```

1: procedure ITERATIVE DE BRUIJN( $G, ((k_0, C_0), \dots, (k_t, C_t))$ )
2:    $S_0 \leftarrow \text{Concatenate}(G)$ 
3:    $\text{Assert}(k_0 < k_1 < \dots < k_t)$ 
4:    $\text{Assert}(C_0 < C_1 < \dots < C_t)$ 
5:    $i \leftarrow 0$ 
6:   while  $i < t$  do
7:      $\text{Graph}_i \leftarrow \text{ConstructDeBruijnGraph}(k_i, S_i)$ 
8:      $S_i \leftarrow \text{SimplifyBulgesSmallerThanC}(\text{Graph}_i, C_i)$ 
9:      $i \leftarrow i + 1$ 
10:  return  $S_t, \text{Graph}_t$ 
11: end procedure

```

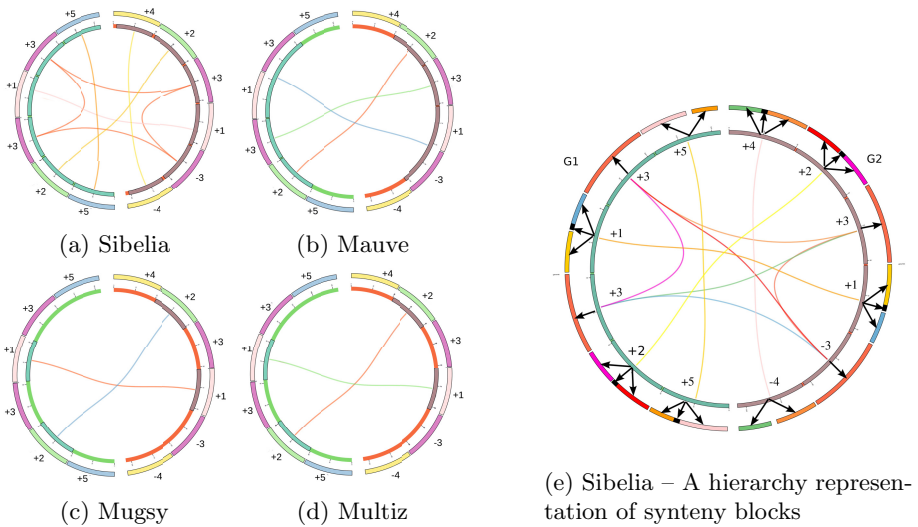


Fig. 3. Performance of different tools on synthetic examples. The outer circles in a), b), c), and d) indicate the gold-standard synteny blocks, where different instances of the same synteny block have the same color and are denoted by the same number. Inner circles indicate blocks found by different tools. All tools were run with their default parameters. a) The last stage from Sibelia; b) Mauve ; c) Mugsy; d) Multiz. e) Sibelia represents synteny blocks in a hierarchy structure. The inner circle indicates large-scale synteny blocks, and the outer circle shows synteny blocks at a finer scale (black blocks represent insertion elements). Each large-scale synteny block (on the inner circle) corresponds to the root of a tree, e.g., block +1 in G_1 (inner circle) can be decomposed into 3 blocks: yellow, black (insertion sequence), and blue (outer circle).

3 Results

Since no gold standard exists for the evaluation of synteny blocks, we first benchmark Sibelia and other tools (Mugsy, Multiz, Mauve) on a simulated dataset. The test case consists of two small hypothetical closely-related genomes, each 120 kbp long. These genomes could be represented as permutations of synteny blocks as follows:

Genome 1: +4 + 2 + 3 + 1 + 3 - 4
 Genome 2: +5 + 2 - 3 + 1 + 3 + 5

In the notation above, numbers depict synteny blocks, and signs designate their orientations. All blocks are 20 kbp long. These blocks indicate various types of repeats: blocks 2 and 1 correspond to common genetic cores, block 3 indicates a repeat common to both genomes, and blocks 4 and 5 are duplicated blocks within each genome. Different instances of a synteny block also contain point mutations, with a 3% probability for each position to change its nucleotide.

Fig. 3 shows the results of different tools on the test case. Sibelia correctly identifies all synteny blocks. No tool except Sibelia is able to locate blocks 4 and 5, and only Mauve detects repeats with multiplicity greater than 2 shared by both genomes. Mugsy and Multiz rely on the Nucmer pairwise aligner package, thus limiting their ability to locate duplications within genomes.

We further demonstrate the ability of Sibelia in detecting and representing synteny blocks on multiple scales by making an additional complication to our simulated genomes. We generate a random DNA sequence of length 1,500 bp (a typical size of insertion sequences, which are common in microbial genomes), and insert this sequence into some previous synteny blocks of these two genomes. We also add 3% mutations to each instance of the inserted sequence.

As different applications favor different synteny block scales (e.g, MGRA [1] may favor the original decomposition, ignoring these insertion elements), other applications may find the translocation of these insert elements biologically significant and thus partition synteny blocks on a finer scale. Fig. 3e shows the hierarchy presentation of Sibelia on the simulated example. In this figure, large-scale synteny blocks are presented in the inner circle, while a finer representation of synteny blocks (with insertion elements denoted as black blocks) is shown in the outer circle. Each synteny block in the inner circle can be decomposed into a sequence of smaller synteny blocks in the outer circle.

3.1 Comparing Sibelia with Existing Tools

We benchmarked Sibelia against Mugsy [2], Multiz [3], and Mauve [8] on 3 datasets: *E.coli-3* — 3 *E.coli* genomes (15 MB), *S.aureus-31* — 31 *S.aureus* genomes (90 MB), and *E.coli-59* — 59 *E.coli* genomes (344 MB). The first dataset *E.coli-3* is used to demonstrate the quality of synteny block generation, while the other larger datasets show the memory consumption and running time performance of these different genome decomposition tools.

On the *E.coli-3* dataset, synteny blocks³ generated from different tools are compared by *genome coverage* and *F-score*. We define the *F-score* of synteny blocks generated from tools *T1* and *T2* as $F = 2(PR)/(P + R)$, where *P* is the fraction of nucleotides in the blocks reported by *T1* that overlap with blocks reported by *T2*, and *R* is the fraction of nucleotides in the blocks from *T2* that overlap with blocks from *T1* (see Table 1). The genome decompositions⁴ of these tools are illustrated in Fig. 5. While the genome decompositions from Sibelia, Mauve, and Mugsy (shown by the three innermost circles in Fig. 5), are similar, Multiz’s blocks are more fragmented. We do not criticize Multiz because different applications favor a different size and scale of repeated blocks. Since Sibelia can present synteny blocks on multiple scales, we show its genome decomposition from the finest scale (first stage) in the outermost circle (Fig. 5), which turns out to be similar to Multiz’s decomposition.

³ Mauve and Mugsy use the term “locally collinear block” instead of “repeated blocks”.

⁴ The ends of repeated blocks define breakpoints on the genome and thus decompose the genome into segments of non-overlapping blocks

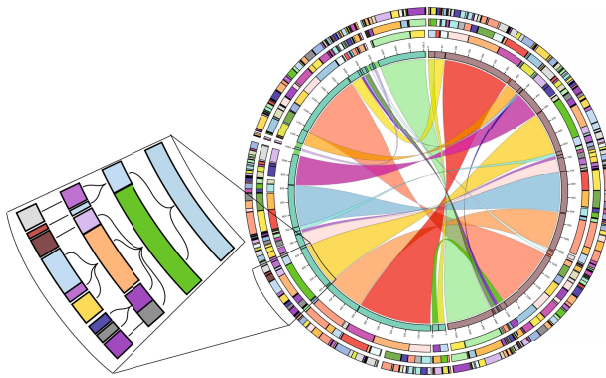


Fig. 4. The figure illustrates iterative construction of synteny blocks between two strains of *Helicobacter pylori*: F32 and Gambia94/24. Each circle represents synteny blocks obtained at a particular stage. The outermost circle represents the first stage of computation, the next inner circle represents the second stage, and so on. Synteny blocks are depicted by colored bands. Multiple instances of the same synteny block within each stage (circle) have the same color. One can notice that from stage to stage, blocks are merged together to form longer blocks. The panel on the left zooms in on a synteny block in the final stage. This panel depicts a tree that represents the decomposition of a synteny block into multiple layers.

Table 1. Synteny block (LCB) Comparison
Genome Coverage F-score

Sibelia	91	100
Mugsy	82	95
Mauve	90	95
Multiz	70	85

Sibelia, Mugsy, Mauve, Multiz were run with their default parameters.

As the number of compared genomes increases, Sibelia shows its advantage in running time performance. When running on 59 *E.coli* and 31 *S.aureus* datasets, Sibelia proves to be 7 times faster than Mugsy and Multiz on *E.coli-59*, (see Table 2). The memory usage of Sibelia is similar to Mugsy and Mauve but is worse than Multiz (Table 2). The synteny blocks that are shared among all genomes (59 *E.coli* and 31 *S.aureus*) cover 66.95% and 54.25% of the average of the genomes size. Using these synteny blocks, one can identify the core genome of each bacterial species. These numbers are consistent with the size of core genomes in *S.aureus* and *E.coli* previously reported [7].

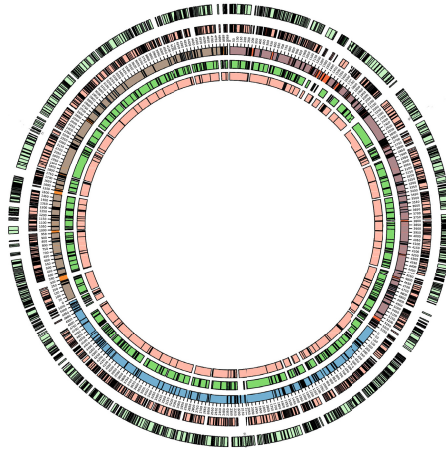


Fig. 5. Circos diagram of syntenic blocks on 3 *E. coli* genomes. From inside to outside: Mauve, Mugsy, Sibelia (the last stage), Multiz, and the first stage of Sibelia ($k = 30, C = 150$). All tools were run with their default parameters.

Table 2. Comparison of running-time/memory usage

	Sibelia	Mugsy	Multiz	Mauve
31 Aureus (min/GB)	28/2.95	362/3.47	129/0.175	814/2.36
59 E.coli (min/GB)	107/8.75	749/9.23	815/0.6	DNF/DNF

The runtime and memory usage for all tools. All tools were run with default parameters. Tests were run on a single CPU Intel Xeon X5675 3GHz processor with 25GB RAM. DNF: allocation error after 12 hours running.

4 Discussion

We have introduced Sibelia, a scalable and comprehensive new syntenic block generation tool for analyzing large numbers of microbial genomes belonging to the same species. By using the *iterative de Bruijn graph*, Sibelia represents syntenic blocks in a hierarchical structure that allows users to explore the composition of syntenic blocks. We are aware that Cactus graphs [15] also decompose genome alignments into substructures based on the topology of nested elements. Our algorithm of decomposing syntenic blocks is different from the nested structure in the Cactus graph, and we plan to further study the relation between these approaches. With the availability of Sibelia, studying genome rearrangement and genome evolution using multiple levels of syntenic blocks promises to be an interesting future research topic.

Acknowledgments. We would like to thank Pavel Pevzner, Hamilton Smith, Steve O’Brien, Alla Lapidus, Matt Schultz, Dinh Diep and Shay Zakov for many insightful discussions. We are indebted to Phillip Compeau, Nitin Udpa and Han Do for revising the manuscript and for many helpful suggestions that significantly improved the paper. We would like to thank Hoa Pham for deploying Sibelia to the webserver. This work was supported by the Government of the Russian Federation (grant 11.G34.31.0018) and the National Institutes of Health (NIH grant 3P41RR024851-02S1).

Appendix

Microbial Species and the Choice of k

While there is no uniquely accepted concept of species in bacteria, the pragmatic species definition is based on DNA-DNA hybridization (DDH) [20]. According to this definition, two isolates belong to the same species if they have $DDH > 70\%$, which in turn corresponds to approximately 95% average nucleotide identity [11]. In other words, within a conserved segment, each position has a 5% chance of mutating.

These mutated points partition any homologous region into a sequence of exact match segments with different lengths. Segments that are longer than k (the size of a vertex in the de Bruijn graph) are *glued* together in the de Bruijn graph; we call these segments *gluing segments*. Two consecutive gluing segments correspond to a bulge in the de Bruijn graph, and any non-gluing segments between the two consecutive gluing segments correspond to branches of the bulge. The distance between two consecutive gluing segments characterizes the size of the bulge.

The probability of encountering an exact matching region of size k is $P\{l = k\} = (1 - \rho)^k \rho$, and the probability of encountering an exact matching region of size at least k is $P\{l \geq k\} = (1 - \rho)^k$, where l is the length of the exact matching region. Given the value $\epsilon = 0.05$, the analysis in [5] allows us to characterize the function $d(k)$, which represents the distance from a given position such that one can encounter at least one gluing segment (exact match segment with length at least k) with probability $1 - \epsilon$. According to [5], $d(k) = \frac{\log(\epsilon)}{\log(1 - (1 - \rho)^k)} \left(\frac{1}{\rho} - \frac{k(1 - \rho)^k}{1 - (1 - \rho)^k} \right)$. The function $d(k)$ characterizes the choice of bulge length threshold for simplification for each given value of k .

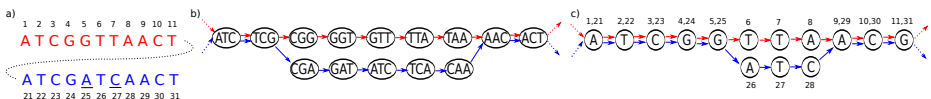


Fig. 6. C-Graph. a) A genome sequence. b) De Bruijn graph for $k = 3$. c) C-Graph for $k = 3$

Hierarchy Representation of Synteny Blocks

Parameter k in Repeats Reconstruction

In this subsection, we give a relationship of repeats that is revealed by non-branching paths in the de Bruijn graph constructed with different values of k . For the simplicity of proving the theorem, we introduce a different type of A-Bruijn graphs, called C-Graphs (Character Graphs), with a slightly different gluing rule from de Bruijn graphs. Given a value of k and a string S of length n formed over the alphabet $\{A, T, C, G\}$, the C-Graph $CG(S, k)$ is defined as follows:

- Represent S as a graph with n vertices labeled $1, \dots, n$ and $n - 1$ edges $(i) \rightarrow (i + 1)$.
- Glue vertex i and j if there exists $t \in [0, k]$ such that $S[i - t : i - t + k - 1] = S[j - t : j - t + k - 1]$

Note that the de Bruijn graph can be obtained by changing the gluing rule above so that we glue i and j if $S[i : i + k - 1] = S[j : j + k - 1]$.

Each vertex v corresponds to a set of integers $A(v)$, representing the positions that are glued to this component. A position i belongs to a vertex if it is contained in $A(v)$. The C-Graph (See Fig. 6) differs from the de Bruijn graph at the boundaries of repeats (branching vertices). The C-graph allows us to avoid overlapping synteny blocks at their shared branching vertices, since each vertex is labeled by a single character that corresponds to the character of S at that particular position⁵. The following theorem shows the relationship between synteny blocks revealed by non-branching paths in C-graphs constructed from different values of k .

Theorem 1. *Given two integers $k_0 < k_1$ and a cyclic genome S , let G_0 and G_1 be the de Bruijn graphs constructed from S with $k = k_0$ and $k = k_1$, respectively. If $S[i : j]$ corresponds to a non-branching path in G_1 (i.e., vertices i, j belong to branching vertices and there does not exist any $t \in (i, j)$ such that t belongs to a branching vertex), then in G_0 , $S[i : j]$ corresponds to a (not necessarily nonbranching) path connecting two branching vertices containing i and j .*

Proof

Since S corresponds to an edge-covering tour in a character graph, it's sufficient to prove that i and j belong to branching vertices in G_0 . Since i belongs to a branching vertex in G_1 , let $I = \{i_1, \dots, i_r\} (i \in I)$ be a set of positions in S that are glued to this vertex. It is evident that these positions will also be glued together in G_0 because $k_0 < k_1$. Since i belongs to a branching vertex in G_1 , there must exist $i_{t1}, i_{t2} \in I$ such that either $S[i_{t1} - 1] = S[i_{t2} - 1]$ or $S[i_{t1} + 1] \neq S[i_{t2} + 1]$. This also implies that i belongs to a branching vertex in G_0 . Similarly, we can prove that j belongs to a branching vertex in G_0 .

⁵ If multiple positions are glued into the same vertex, we can also use the character to label any of these gluing positions, as they are identical

References

1. Alekseyev, M.A., Pevzner, P.A.: Breakpoint graphs and ancestral genome reconstructions. *G.R.* 19(5), 943–957 (2009)
2. Angiuoli, S.V., Salzberg, S.L.: Mugsy: fast multiple alignment of closely related whole genomes. *Bioinformatics* 27(3), 334–342 (2011)
3. Blanchette, M., Kent, W., Riemer, C., Elnitski, L., Smit, A., Roskin, K., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E., et al.: Aligning multiple genomic sequences with the threaded blockset aligner. *G.R.* 14(4), 708–715 (2004)
4. Brüßow, H., Canchaya, C., Hardt, W.-D.: Phages and the evolution of bacterial pathogens: from genomic rearrangements to lysogenic conversion. *Microbiology and Molecular Biology Reviews* 68(3), 560–602 (2004)
5. Chaisson, M., Tesler, G.: Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC Bioinformatics* 13, 238 (2012)
6. Chambers, H.F., et al.: Community-associated mrsa-resistance and virulence converge. *N. Engl. J. Med.* 352(14), 1485–1487 (2005)
7. Chattopadhyay, S., Weissman, S.J., Minin, V.N., Russo, T.A., Dykhuizen, D.E., Sokurenko, E.V.: High frequency of hotspot mutations in core genes of *escherichia coli* due to short-term positive selection. *PNAS* 106(30), 12412–12417 (2009)
8. Darling, A., Mau, B., Blattner, F., Perna, N.: Mauve: multiple alignment of conserved genomic sequence with rearrangements. *G.R.* 14(7), 1394–1403 (2004)
9. Gibson, D.G., Benders, G.A., Andrews-Pfannkoch, C., Denisova, E.A., Baden-Tillson, H., Zaveri, J., Stockwell, T.B., Brownley, A., Thomas, D.W., Algire, M.A., et al.: Complete chemical synthesis, assembly, and cloning of a *mycoplasma genitalium* genome. *Science Signalling* 319(5867), 1215 (2008)
10. Kaper, J.B., Nataro, J.P., Mobley, H.L.T.: Pathogenic *escherichia coli*. *Nature Reviews Microbiology* 2(2), 123–140 (2004)
11. Konstantinidis, K., Ramette, A., Tiedje, J.: The bacterial species definition in the genomic era. *Philosophical Transactions of the Royal Society B: Biological Sciences* 361(1475), 1929–1940 (2006)
12. Lunter, G., Rocco, A., Mimouni, N., Heger, A., Caldeira, A., Hein, J.: Uncertainty in homology inferences: assessing and improving genomic sequence alignment. *G.R.* 18(2), 298–309 (2008)
13. Medvedev, P., Pham, S., Chaisson, M., Tesler, G., Pevzner, P.: Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *JCB* 18(11), 1625–1634 (2011)
14. Ohtsubo, E., Sekine, Y.: Bacterial insertion sequences. In: *Transposable Elements*, pp. 1–26. Springer (1996)
15. Paten, B., Earl, D., Nguyen, N., Diekhans, M., Zerbino, D., Haussler, D.: Cactus: Algorithms for genome multiple sequence alignment. *G.R.* 21(9), 1512–1528 (2011)
16. Peng, Q., Alekseyev, M., Tesler, G., Pevzner, P.: Decoding synteny blocks and large-scale duplications in mammalian and plant genomes. *Algorithms in Bioinformatics*, 220–232 (2009)
17. Pevzner, P.A., Tang, H., Tesler, G.: De novo repeat classification and fragment assembly. *G.R.* 14(9), 1786–1796 (2004)
18. Pham, S.K., Pevzner, P.A.: Drimm-synteny: decomposing genomes into evolutionary conserved segments. *Bioinformatics* 26(20), 2509–2516 (2010)
19. Sinha, A.U., Meller, J.: Cinteny: flexible analysis and visualization of synteny and genome rearrangements in multiple organisms. *BMC Bioinformatics* 8(1), 82 (2007)
20. Wayne, L., Brenner, D., et al.: Report of the ad hoc committee on reconciliation of approaches to bacterial systematics. *International Journal of Systematic Bacteriology* 37(4), 463–464 (1987)

On the Matrix Median Problem

João Paulo Pereira Zanetti¹, Priscila Biller¹, and João Meidanis^{1,2}

¹ Institute of Computing, University of Campinas, SP, Brazil

² Scylla Bioinformatics, Campinas, Brazil

Abstract. The Genome Median Problem is an important problem in phylogenetic reconstruction under rearrangement models. It can be stated as follows: given three genomes, find a fourth that minimizes the sum of the pairwise rearrangement distances between it and the three input genomes. Recently, Feijão and Meidanis extended the algebraic theory for genome rearrangement to allow for linear chromosomes, thus yielding a new rearrangement model (the algebraic model), very close to the celebrated DCJ model. In this paper, we study the genome median problem under the algebraic model, whose complexity is currently open, proposing a more general form of the problem, the matrix median problem. It is known that, for any metric distance, at least one of the corners is a $\frac{4}{3}$ -approximation of the median. Our results allow us to compute up to three additional matrix median candidates, all of them with approximation ratios at least as good as the best corner, when the input matrices come from genomes. From the application point of view, it is usually more interesting to locate medians farther from the corners. We also show a fourth median candidate that gives better results in cases we tried. However, we do not have proven bounds for this fourth candidate yet.

1 Introduction

Genome rearrangements are evolutionary events where large, continuous pieces of the genome shuffle around, changing the order of genes in the genome of a species. Gene order data can be very useful in estimating the evolutionary distance between genomes, and also in reconstructing the gene order of ancestral genomes. The simplest form of inference of evolutionary scenarios based on gene order is the pairwise genome rearrangement problem: given two genomes, find a parsimonious rearrangement scenario between them, that is, the smallest sequence of rearrangement events that transforms one genome into the other.

For most rearrangement events proposed, this problem has already been solved, usually with linear or subquadratic algorithms. However, when more than two genomes are considered, inferring evolutionary scenarios becomes much more difficult. This problem is known as the *multiple genome rearrangement problem* (MGRP): given a set of genomes, find a tree with the given genomes as leaves and an assignment of genomes to the internal nodes such that the sum of all edge lengths (the pairwise distance between adjacent genomes) is minimal.

The MGRP may still be hard even when only three genomes are considered, the so called *genome median problem* (GMP): given three genomes, find a fourth

genome that minimizes the sum of its pairwise distances to the other three. The GMP is NP-complete in most rearrangement models, with notable exceptions being the multichromosomal breakpoint distance [10] and the Single-Cut-or-Join (SCJ) model [3]. The GMP is a particularly interesting problem, because several algorithms for the MGRP are based on repeatedly solving GMP instances, until convergence is reached (for instance, the pioneering BPAAnalysis [8], the more recent GRAPPA [7], and MGR [1]).

In this paper we will focus on the algebraic rearrangement model proposed by Meidanis and Dias [6], recently extended to allow linear chromosomes in a very natural way by Feijão and Meidanis [4]. This extended algebraic rearrangement model is similar to the well-known Double-Cut-and-Join (DCJ) model [11], with a slight difference in the weight of single cut/join operations, where the weight for this operations is 1 in the DCJ model, but 1/2 in the algebraic model. The algebraic pairwise distance problem can be solved in linear time, but the median problem remains open.

The main goal of this paper is to investigate the problem of computing the algebraic median of three genomes. The median problem can be stated as follows: given three genomes π_1 , π_2 , and π_3 , and a distance metric d , find a genome μ that minimizes $d(\mu; \pi_1, \pi_2, \pi_3)$, defined as the *total score*

$$d(\mu; \pi_1, \pi_2, \pi_3) = d(\mu, \pi_1) + d(\mu, \pi_2) + d(\mu, \pi_3).$$

We do not know yet the status of the median problem for the algebraic distance, but suspect it may be NP-hard as well. However, in this paper we show that viewing genomes (or even general permutations) as matrices, the median can be approximated quickly, although the matrix solution may not be always translated back into permutations or genomes. Nevertheless, this positive result can help shed more light into the problem, by leading to approximation solutions, or to special cases that can be solved polynomially in the genome setting.

It is known that, for any distance satisfying the axioms of a metric, at least one of the corners is a $\frac{4}{3}$ -approximation of the median [9]. Our results allow us to compute up to three additional matrix median candidates, all of them with approximation ratios at least as good as the best corner, when the input matrices come from genomes. Also, application-wise, it is usually more interesting to locate medians farther from the corners [5].

The rest of this paper is organized as follows. In Section 2, we have basic definitions regarding the algebraic adjacency theory needed in this work. In Section 3, we show how genomes can also be seen as matrices, define the matrix median problem, show our results, and propose an algorithm. Finally, in Section 4, we present our conclusions.

2 Algebraic Rearrangement Theory

We will start this section showing some basic definitions of the algebraic theory of Feijão and Meidanis [4].

2.1 Permutations

Given a set E , a *permutation* $\alpha : E \rightarrow E$ is a bijective map from E onto itself. Permutations are represented as parenthesized lists, with each element followed by its image. For instance, on $E = \{a, b, c\}$, $\alpha = (a b c)$ is the permutation that maps a to b , b to c , and maps c back to a . This representation is not unique; $(b c a)$ and $(c a b)$ are equivalent. Permutations are composed of one or more *cycles*. For instance, the permutation $\alpha = (a b c)(d e)(f)$ has three cycles. A cycle with k elements is called a *k-cycle*. An 1-cycle represents a fixed element in the permutation and is usually omitted.

The *product* or *composition* of two permutations α, β is denoted by $\alpha\beta$. The product $\alpha\beta$ is defined as $\alpha\beta(x) = \alpha(\beta(x))$ for $x \in E$. For instance, with $E = \{a, b, c, d, e, f\}$, $\alpha = (b d e)$ and $\beta = (c a e b f d)$, we have $\alpha\beta = (c a b f e d)$.

The *identity permutation*, which maps every element into itself, will be denoted by $\mathbf{1}$. Every permutation α has an *inverse* α^{-1} such that $\alpha\alpha^{-1} = \alpha^{-1}\alpha = \mathbf{1}$. For a cycle, the inverse is obtained by reverting the order of its elements: $(c b a)$ is the inverse of $(a b c)$.

A *2-cycle decomposition* of a permutation α is a representation of α as a product of 2-cycles, not necessarily disjoint. All permutations have a 2-cycle decomposition. The *norm* of a permutation α , denoted by $\|\alpha\|$, is the minimum number of cycles in a 2-cycle decomposition of α . For example, the permutation $\alpha = (a b c d)$ can be decomposed as $(a b)(b c)(c d)$, and $\|\alpha\| = 3$.

2.2 Modeling Genomes as Permutations

To model genomes with the Algebraic Theory, the formulation is similar to the set representation of a genome, used in several related works [10,3]. In this representation, each gene a has two *extremities*, called *tail* and *head*, respectively denoted by a_t and a_h , or alternatively using signs, where $-a = a_h$ and $+a = a_t$. An *adjacency* is an unordered pair of extremities indicating a linkage between two consecutive genes in a chromosome. An extremity not adjacent to any other extremity in a genome is called a *telomere*. A genome is represented as a set of adjacencies and telomeres (the telomeres may be omitted, when the gene set is given) where each extremity appears at most once.

According to algebraic rearrangement theory [4], a genome can be seen as a permutation $\pi : E \mapsto E$, where E is the set of gene extremities, with the added property that $\pi^2 = \mathbf{1}$, the identity permutation. In this paper, the *distance* between two genomes or two permutations π and σ will be defined as $d(\pi, \sigma) = \|\sigma\pi^{-1}\|$. It is important to note that, in the original paper, the algebraic distance is defined as $\frac{\|\sigma\pi^{-1}\|}{2}$. However, to avoid dealing with fractional numbers and to simplify the calculations, we will multiply the distances by 2 in this paper.

In the algebraic theory, genomes are represented by permutations, with a *genome* being a product of 2-cycles, where each 2-cycle corresponds to an adjacency. Figure 1 shows an example of a genome and its representation as a permutation.

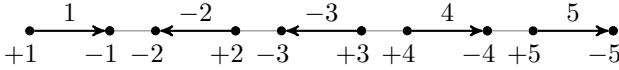


Fig. 1. A genome with one linear chromosome, represented by the permutation $\pi = (-1 -2)(+2 -3)(+3 +4)(-4 +5)$

With these definitions, a distance between two genomes σ and π can be defined as $d(\pi, \sigma) = \|\sigma\pi^{-1}\|$, as mentioned in the introduction. The resulting distance agrees with DCJ in circular genomes, and in general these two distances are very close.

3 Results

Permutations can be seen as matrices, and, in this section, we define a matrix distance that corresponds to the algebraic distance and prove that it is indeed a valid metric in general, that is, it applies to all square matrices, not only those associated to permutations. Such a metric can be useful in the computation of genome medians, and we show here how to compute an approximate solution to the matrix median problem, by solving a system of linear equations.

3.1 Matrix Distance

Given two $n \times n$ matrices A and B , we define the *distance* between them as:

$$d(A, B) = r(B - A),$$

where $r(X)$ denotes the **rank** of matrix X . It is well-known that

$$r(X) = \dim \text{im}(X), \tag{1}$$

where \dim denotes the dimension of a vector space and $\text{im}(X)$ is the **image** of X , namely, the space of all vectors that can be written as Xv for some $v \in \mathbb{R}^n$. Here we treat vectors as column matrices, that is, $n \times 1$ matrices. Therefore, the distance satisfies

$$d(A, B) = \dim \text{im}(B - A).$$

for every pair of matrices A and B . It is also well-known that

$$r(X) = n - \dim \ker(X), \tag{2}$$

for every $x \times n$ matrix X , where $\ker(X)$ is the **kernel** of X , defined as the space of all vectors $v \in \mathbb{R}^n$ such that $Xv = 0$, so the distance satisfies yet another formula:

$$d(A, B) = n - \dim \ker(B - A).$$

This distance can be shown to satisfy the conditions of a metric, that is, it is symmetric, obeys the triangle inequality, and $d(A, B) = 0$ if and only if $A = B$ [2].

Given this metric, the first interesting observation is that permutations (including genomes) can be mapped to matrices in a distance-preserving way. Given a permutation $\alpha : E \mapsto E$, with $|E| = n$, we first identify each element v of E with a unit vector of \mathbb{R}^n , and then define A , the matrix counterpart of α , so that

$$Av = \alpha v. \tag{3}$$

In Equation 3, we use v both as a unit vector of \mathbb{R}^n in the left side, and as an element of E in the right side.

An example will help. Let α be the permutation $(a\ b)(c\ d)$. Identify $a = [1\ 0\ 0\ 0]^t$, $b = [0\ 1\ 0\ 0]^t$, $c = [0\ 0\ 1\ 0]^t$, and $d = [0\ 0\ 0\ 1]^t$, where v^t is the transpose of vector v . We then have

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

It is well known that this mapping produces matrices A that are invertible, and that satisfy $A^{-1} = A^t$, where A^t denotes the transpose of matrix A . Also, the identity permutation corresponds to the identity matrix I , and the product $\alpha\beta$ corresponds to matrix AB , where A is the matrix corresponding to α , and B is the matrix corresponding to β . If α happens to be a genome, that is, if $\alpha^2 = \mathbf{1}$, then A is a symmetric matrix and vice-versa.

We now show that this mapping is distance-preserving.

Lemma 1. *For any permutation σ and π , and their respective associated matrices S and P , we have:*

$$d(\sigma, \pi) = d(S, P).$$

Proof. First, notice that it suffices to show that

$$\|\alpha\| = r(A - I), \tag{4}$$

for any permutation α and associated matrix A . Indeed, if P is invertible then

$$r(S - P) = r(SP^{-1} - I),$$

and then Equation (4) will relate $d(\sigma, \pi)$ to the distances of the corresponding matrices S and P .

Then proceed to show that, for a k -cycle, Equation (4) is true since both sides are equal to $k - 1$. Finally, for a general permutation, decompose it in disjoint cycles, and use the fact that, for *disjoint* permutations α and β , with associated matrices A and B , respectively, we have

$$\ker(A - I) \cap \ker(B - I) = \ker(AB - I),$$

and

$$\ker(A - I) + \ker(B - I) = \mathbb{R}^n,$$

which guarantee that

$$n - \dim \ker(AB - I) = n - \dim \ker(A - I) + n - \dim \ker(B - I).$$

or

$$r(AB - I) = r(A - I) + r(B - I),$$

because of Equation 2.

Therefore, if Equation (4) is valid for α and β , and if α and β are disjoint, then Equation (4) is valid for the product $\alpha\beta$. Since any permutation can be written as a product of disjoint cycles, Equation (4) is valid in general. \square

3.2 Matrix Median

Because the correspondence between permutations and matrices preserves distances, it makes sense to study the matrix median problem as a way of shedding light into the permutation median problem, which in turn is related to the genome median problem.

Let A , B , and C be three $n \times n$ matrices. Suppose we want to find a matrix M such that

$$d(M; A, B, C) = d(M, A) + d(M, B) + d(M, C)$$

is minimized. In order to have small $d(M, A)$, M must be equal to A in a large subspace, so that $\ker(A - M)$ is large. Similarly with B and C .

This suggests the following strategy. Decompose \mathbb{R}^n as a direct sum of five subspaces, where the following relations are true: (1) $A = B = C$, (2) $A = B \neq C$, (3) $A \neq B = C$, (4) $A = C \neq B$, and (5) $A \neq B \neq C \neq A$.

In the first subspace, since A , B , and C all have the same behaviour, M should also do the same thing. In the second subspace, since $A = B$ but C is different, it is better for M to go with A and B . Likewise, in the third subspace M should concur with B and C , and with A and C in the fourth. Finally, in the final subspace it seems hard to gain points in two different distances, so the best course for M would be to mimic one of A , B , or C .

Therefore, making M equal to A , except in the third subspace, where it should be equal to B (and C) should yield a good approximation of a median, if not a median. The rest of this section will be devoted to showing the details on this construction.

3.3 Partitioning \mathbb{R}^n

We begin by introducing notation aimed at formalizing subspaces such as $A = B \neq C$. Given $n \times n$ matrices A , B , and C , we will use a dotted notation to indicate a partition, e.g., $.AB.C$ means a partition where A and B are in one

class, and C is in another class by itself. To each such partition, we associate a vector subspace of \mathbb{R}^n formed by those vectors having the same image in each class:

$$V(.AB.C.) = \{v \in \mathbb{R}^n \mid Av = Bv\}.$$

Notice that singleton classes do not impose additional restrictions. With this notation, the subspace we used to call $A = B = C$ can be written as $V(.ABC.)$. Notice also that $V(.A.B.C.) = \mathbb{R}^n$.

We need also a notation for strict subspaces, such as $A = B \neq C$, where distinct classes actually disagree, that is, subspaces where vectors have different images under each class. There can be more than one subspace satisfying this property, but we can use orthogonality to define a unique subspace. For a partition p , we define $V_*(p)$ as the orthogonal complement of the sum of the partitions strictly refined by p with respect to $V(p)$:

$$V_*(p) = V(p) \cap \left(\sum_{p < q} V(q)\right)^\perp$$

where $p < q$ means that partition p strictly refines partition q . In other words, we want to capture the part of the subspace $V(p)$ that is orthogonal to the sum of the subspaces corresponding to coarser partitions. Notice that $p < q$ implies $V(q) \subseteq V(p)$.

It is easy to see that the V_* subspaces are pairwise disjoint, but this is not enough to prove that their direct sum is \mathbb{R}^n . So, the proof will start from the basic sum $V_*(.ABC.) \oplus V_*(.AB.C.)$, where we already know that $V_*(.ABC.) \cap V_*(.AB.C.) = \{0\}$, and it will add one subspace in the sum at a time, ensuring that the intersection between the new subspace and the sum of the subspaces previously included contains the zero vector only.

Lemma 2. *If A , B , and C are permutation matrices, then*

$$(V_*(.ABC.) + V_*(.AB.C.)) \cap V_*(.BC.A.) = \{0\}.$$

Proof. Let v be a vector such that

$$v \in (V_*(.ABC.) + V_*(.AB.C.)) \cap V_*(.BC.A.).$$

We have that $Av = Bv$ and also $Av = Cv$. Therefore $Av = Bv = Cv$ and, consequently, $v = 0$, since $v \in V_*(.ABC.)$. □

Before we add more subspaces to the sum, we will need the result of the following lemma.

Lemma 3. *If A , B , and C are permutation matrices, and if*

$$2Bv = Av + Cv$$

for a given vector v , then $Av = Cv$.

Proof. Denote by $|x|$ the norm of a vector x . Note that A , B , and C preserve norms, thus

$$|Av + Cv| = |2Bv| = 2|v| = |v| + |v| = |Av| + |Cv|.$$

But, if the norm of the sum is equal to the sum of the norms, then the two vectors are parallel, and have the same orientation. In other words, there is a positive scalar c such that $cAv = Cv$. But we already have that $|Av| = |v| = |Cv|$. Therefore, $c = 1$. □

Lemma 4. *If A , B , and C are permutation matrices, then*

$$(V_*(.ABC.) + V_*(.AB.C.) + V_*(.BC.A.)) \cap V_*(.AC.B.) = \{0\}.$$

Proof. Suppose that $u + v + w \in V_*(.AC.B.)$, where $u \in V_*(.ABC.)$, $v \in V_*(.AB.C.)$, and $w \in V_*(.BC.A.)$. We have that $A(u + v + w) = C(u + v + w)$, which implies $A(v + w) = C(v + w)$, since $Au = Cu$. Thus, we have

$$\begin{aligned} Av + Aw &= Cv + Cw \\ Bv + Aw &= Cv + Bw \\ B(v - w) &= Cv - Aw. \end{aligned}$$

Now we will apply A and C to $v - w$, and sum the results, obtaining:

$$\begin{aligned} A(v - w) &= Bv - Aw \\ C(v - w) &= Cv - Bw \\ A(v - w) + C(v - w) &= B(v - w) + Cv - Aw = 2B(v - w) \end{aligned}$$

By Lemma 3, we conclude that $A(v - w) = C(v - w)$. But we also have that $A(v + w) = C(v + w)$, which implies $Av = Cv$ and $Aw = Cw$. In other words, $u + v + w \in V_*(.ABC.)$. But $u + v + w$ also belongs to $V_*(.AC.B.)$. It follows that $u + v + w = 0$. □

Lemma 5. *If A , B , and C are permutation matrices, then*

$$(V_*(.ABC.) + V_*(.AB.C.) + V_*(.BC.A.) + V_*(.AC.B.)) \cap V_*(.A.B.C.) = \{0\}.$$

Proof. By definition, the two subspaces whose intersection is indicated in the left-hand side are orthogonal to one another. Therefore, they have a zero intersection. □

With Lemmas 2 through 5 we get to the following theorem:

Theorem 1. *If A , B , and C are permutation matrices, then*

$$\mathbb{R}^n = V_*(.ABC.) \oplus V_*(.AB.C.) \oplus V_*(.BC.A.) \oplus V_*(.AC.B.) \oplus V_*(.A.B.C.).$$

It is important to observe that Theorem 1 does not apply to general matrices, for instance:

$$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

With these three matrices, we have

$$\begin{aligned} V_*(.ABC.) &= \{0\}, \\ V_*(.AB.C.) &= \langle [1 \ 0]^t \rangle, \\ V_*(.BC.A.) &= \langle [1 \ 0]^t, [0 \ 1]^t \rangle, \\ V_*(.AC.B.) &= \langle [0 \ 1]^t \rangle, \end{aligned}$$

where $\langle X \rangle$ denotes the space spanned by the set X .

We can see that, in this case,

$$(V_*(.ABC.) + V_*(.AB.C.) + V_*(.BC.A.)) \cap V_*(.AC.B.) \neq \{0\}.$$

3.4 Computing Median Candidates

We now get into further detail in the procedure to compute these matrices. We saw that, when A , B , and C are permutation matrices, \mathbb{R}^n can be decomposed into a direct sum of V_* subspaces. We will now implement the procedure outlined in Section 3.2, summarized in Table 1, to obtain a median candidate M_A .

Table 1. Distance contribution — Given three permutation matrices A , B and C , this table shows the distance contribution of each of the five subspaces partitioning \mathbb{R}^n to the distances $d(M_A, A)$, $d(M_A, B)$, and $d(M_A, C)$, for a candidate median matrix M_A .

Subspace	$M_A = \dots$	Contributes to ...		
		$d(M, A)$	$d(M, B)$	$d(M, C)$
$V_*(.A.B.C.)$	A	no	yes	yes
$V_*(.AB.C.)$	A	no	no	yes
$V_*(.BC.A.)$	B	yes	no	no
$V_*(.AC.B.)$	A	no	yes	no
$V_*(.ABC.)$	A	no	no	no

One way to implement this strategy is to compute orthogonal projection matrices P_1, P_2, P_3, P_4 , and P_5 for each of the subspaces and then compute M_A as follows:

$$\begin{aligned} M_A &= AP_1 + AP_2 + BP_3 + AP_4 + AP_5 \\ &= A + (B - A)P_3, \end{aligned}$$

since $P_1 + P_2 + P_3 + P_4 + P_5 = I$. To obtain each matrix P_i all we need is a $n \times k_i$ matrix B_i whose columns form an orthonormal basis of the corresponding subspace, because P_i can then be written as $B_i B_i^t$.

To build the B_i s, one possibility is to use Function Add below, a basic routine to expand an orthonormal basis so that it can generate a given extra vector v . It projects the new vector in the orthogonal complement of the subspace generated by the original basis and then adds the normalized projection to form the new basis. Function Add's complexity is $O(kn)$ arithmetic operations (additions, subtractions, multiplications, and divisions).

Function Add(B, v) Augments an orthonormal basis B so that it can generate the vector v .

Data: An orthonormal basis $B = \{v_1, \dots, v_k\}$ and a vector v .

Result: An augmented basis.

```

1  $w \leftarrow v - \sum_{i=1}^k v_i v_i^t v$ 
2 if  $w = 0$  then
3   | Return  $B$ 
4 else
5   | Normalize  $w$ 
6   | Return  $B \cup \{w\}$ 

```

Algorithm 1 uses Function Add and a linear system solving method Solve, that returns a linearly independent set of vectors spanning the solution space, to determine orthonormal bases for each of the five V_* subspaces. Assuming that we can solve a system of n linear equations in time $O(n^3)$, the total complexity for Algorithm 1 is $O(n^3)$ as well.

Once we have determined orthonormal bases for the V_* subspaces, it is easy to find projection matrices for them. For each B_i , its projection matrix is $P_i = B_i B_i^t$, as previously said.

Knowing the projection matrices, we can then finally compute the median candidates, in Algorithm 2. Previously, we saw how to compute M_A . It is also possible to define M_B and M_C in an analogous way. The matrix M_B follows B in $V_*(.A.B.C.)$ instead of A , and M_C follows C . The entire computation takes $O(n^3)$ arithmetic operations.

We can also define a median candidate M_I , that follows the identity in the subspace $V_*(.A.B.C.)$. That is,

$$M_I = A(P_1 + P_2 + P_4) + B P_3 + P_5 = A + (B - A)P_3 + (I - A)P_5.$$

We still have no proven results on its total median score, but we conjecture that, for genomic matrices, M_I is better than M_A , M_B and M_C , or it might even be a median. This is due to the symmetric nature of the genomic matrices.

Algorithm 1. Computation of orthonormal bases for each of the V_* subspaces

Data: Three $n \times n$ permutation matrices A , B and C .

Result: The bases for the V_* subspaces.

```

1  $B \leftarrow \emptyset$ 
2  $L \leftarrow \text{Solve } \{(A - B)v = 0, (A - C)v = 0\}$ 
3 foreach  $v \in L$  do
4    $B \leftarrow \text{Add}(B, v)$ 
5  $B_1 \leftarrow B$  // Basis for  $V_*(.ABC.)$ 
6  $L \leftarrow \text{Solve } \{(A - B)v = 0\}$ 
7 foreach  $v \in L$  do
8    $B \leftarrow \text{Add}(B, v)$ 
9  $B_2 \leftarrow B - B_1$  // Basis for  $V_*(.AB.C.)$ 
10  $L \leftarrow \text{Solve } \{(B - C)v = 0\}$ 
11 foreach  $v \in L$  do
12    $B \leftarrow \text{Add}(B, v)$ 
13  $B_3 \leftarrow B - B_1 - B_2$  // Basis for  $V_*(.BC.A.)$ 
14  $L \leftarrow \text{Solve } \{(A - C)v = 0\}$ 
15 foreach  $v \in L$  do
16    $B \leftarrow \text{Add}(B, v)$ 
17  $B_4 \leftarrow B - B_1 - B_2 - B_3$  // Basis for  $V_*(.AC.B.)$ 
18  $L \leftarrow$  canonical basis for  $\mathbb{R}^n$ 
19 foreach  $v \in L$  do
20    $B \leftarrow \text{Add}(B, v)$ 
21  $B_5 \leftarrow B - B_1 - B_2 - B_3 - B_4$  // Basis for  $V_*(.A.B.C.)$ 

```

Algorithm 2. Computation of median candidates

Data: Three $n \times n$ permutation matrices A , B and C .

Result: Three median candidates M_A , M_B , and M_C .

```

1 Compute the bases  $B_1, B_2, B_3, B_4$ , and  $B_5$  with Algorithm 1
2 foreach  $i \in \{1, 2, 3, 4, 5\}$  do
3    $P_i \leftarrow B_i B_i^t$ 
4  $M_A \leftarrow A + (B - A)P_3$ 
5  $M_B \leftarrow B + (A - B)P_4$ 
6  $M_C \leftarrow C + (B - C)P_2$ 

```

3.5 Approximation Factor

We already know that the direct sum of the V_* subspaces is \mathbb{R}^n , when A , B , and C are permutation matrices. Now we can use this property to express the distance in terms of these subspaces dimensions. The matrix M_A will have a total distance to A , B , and C equal to:

$$\begin{aligned}
 d(M_A; A, B, C) &= d(M, A) + d(M, B) + d(M, C) \\
 &= \dim V_*(.BC.A.) + \\
 &\quad \dim V_*(.A.B.C.) + \dim V_*(.AC.B.) + \\
 &\quad \dim V_*(.A.B.C.) + \dim V_*(.AB.C.) \\
 &= 2 \dim V_*(.A.B.C.) + \\
 &\quad \dim V_*(.AB.C.) + \dim V_*(.AC.B.) + \dim V_*(.BC.A.). \quad (5)
 \end{aligned}$$

There are cases where M_A is not a median, even if the input matrices are genomic. Take, for example:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \text{ and } C = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

By Equation (5), the matrix M_A has the following total score:

$$d(M_A; A, B, C) = 2 \times 2 + 0 + 0 + 0 = 4.$$

However, the identity matrix I has a better total score than M_A , and is actually a median in this case:

$$d(I; A, B, C) = 1 + 1 + 1 = 3.$$

Thus, given that the procedure described in the Section 3.4 does not guarantee a matrix median, it is interesting to know whether it is an approximation algorithm, namely, whether there is a constant ρ such that the candidate's total score is at most ρ times the score of a median.

In general, consider permutation matrices A , B , and C and a matrix M such that $d(M; A, B, C)$ is minimum, that is, M is a median. There is a trivial lower bound for the median score of a matrix, easily obtained with the help of the triangle inequality, namely

$$d(M; A, B, C) \geq \frac{1}{2}(d(A, B) + d(B, C) + d(C, A)).$$

According to Equation (5), the median score of the approximate solution M_A constructed in Section 3.4 is given by:

$$\begin{aligned}
 d(M_A; A, B, C) &= 2 \dim V_*(.A.B.C.) + \\
 &\quad \dim V_*(.AB.C.) + \dim V_*(.AC.B.) + \dim V_*(.A.BC.),
 \end{aligned}$$

For comparison, we can write the trivial lower bound in terms of subspace dimensions. It suffices to write each distance as a dimension sum of the subspaces where they differ. The result is:

$$\frac{1}{2}(d(A, B) + d(B, C) + d(C, A)) = \frac{3}{2} \dim V_*(.A.B.C.) + \dim V_*(.AB.C.) + \dim V_*(.AC.B.) + \dim V_*(.A.BC.).$$

Then, to prove that the matrix M_A is indeed an approximate solution, it suffices to show that there is a constant ρ such that

$$d(M_A; A, B, C) \leq \rho d(M; A, B, C),$$

for any given matrices A , B , and C .

It is possible to demonstrate that $\frac{4}{3}$ is an approximate factor for our solution, as follows:

$$\begin{aligned} d(M_A; A, B, C) &= 2 \dim V_*(.A.B.C.) + \dim V_*(.AB.C.) + \dim V_*(.AC.B.) + \\ &\quad + \dim V_*(.A.BC.) \\ &\leq \frac{4}{3} \left[\frac{3}{2} \dim V_*(.A.B.C.) + \dim V_*(.AB.C.) + \dim V_*(.AC.B.) + \right. \\ &\quad \left. + \dim V_*(.A.BC.) \right] \\ &\leq \frac{4}{3} \left[\frac{1}{2} (d(A, B) + d(B, C) + d(C, A)) \right] \\ &\leq \frac{4}{3} d(M; A, B, C). \end{aligned}$$

Thus, we proved that $d(M_A; A, B, C)$ is at most $\frac{4}{3}$ times $d(M; A, B, C)$. The same result holds for M_B and M_C .

4 Conclusions

We showed in this paper that it is possible to define a distance on matrices in a way that yields exactly the algebraic distance when restricted to permutation matrices. For the case where the input matrices represent genomes, we have shown how to compute matrices that approximate the median with factors at least as good as the best corner, that is, they are approximations to the median within a factor of $\frac{4}{3}$. In addition, we showed a construction, in the form of matrix M_I , that in several examples we worked out is even better than the other median candidates. We conjecture that this matrix might be a median if A , B , and C come from genomes. The implications to computing algebraic genome medians can be significant.

Acknowledgements. We thank Luiz Antonio Barrera San Martin, who suggested linear representations in a discussion on permutations, the Research Funding Agency of the State of Sao Paulo (FAPESP), for grants 2012/13865-7 and 2012/14104-0, and the anonymous referees for helpful comments that improved the paper significantly.

References

1. Bourque, G., Pevzner, P.A.: Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Research* 12(1), 26–36 (2002)
2. Delsarte, P.: Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A* 25(3), 226–241 (1978)
3. Feijao, P., Meidanis, J.: SCJ: a breakpoint-like distance that simplifies several rearrangement problems. *IEEE IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8, 1318–1329 (2011)
4. Feijao, P., Meidanis, J.: Extending the Algebraic Formalism for Genome Rearrangements to Include Linear Chromosomes. In: de Souto, M.C.P., Kann, M.G. (eds.) *BSB 2012. LNCS (LNBI)*, vol. 7409, pp. 13–24. Springer, Heidelberg (2012)
5. Haghighi, M., Sankoff, D.: Medians seek the corners, and other conjectures. *BMC Bioinformatics* 13(suppl. 19), S5 (2012)
6. Meidanis, J., Dias, Z.: An alternative algebraic formalism for genome rearrangements. In: Sankoff, D., Nadeau, J. (eds.) *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and Evolution of Gene Families*, pp. 213–223. Kluwer Academic Publishers (2000)
7. Moret, B.M., Wang, L.S., Warnow, T., Wyman, S.K.: New approaches for reconstructing phylogenies from gene order data. *Bioinformatics* 17(suppl. 1), S165–S173 (2001)
8. Sankoff, D., Blanchette, M.: Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology* 5(3), 555–570 (1998)
9. Sankoff, D., Sundaram, G., Kececioğlu, J.: Steiner points in the space of genome rearrangements. *International Journal of Foundations of Computer Science* 7(01), 1–9 (1996)
10. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics* 10, 120 (2009)
11. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics* 21(16), 3340–3346 (2005)

A Fixed-Parameter Algorithm for Minimum Common String Partition with Few Duplications

Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz*,
and Irena Rusu

Université de Nantes, LINA - UMR CNRS 6241, France.

{Laurent.Bulteau,Guillaume.Fertin,Christian.Komusiewicz,
Irena.Rusu}@univ-nantes.fr

Abstract. Motivated by the study of genome rearrangements, the NP-hard MINIMUM COMMON STRING PARTITION problem asks, given two strings, to split both strings into an identical set of blocks. We consider an extension of this problem to unbalanced strings, so that some elements may not be covered by any block. We present an efficient fixed-parameter algorithm for the parameters number k of blocks and maximum occurrence d of a letter in either string. We then evaluate this algorithm on bacteria genomes and synthetic data.

1 Introduction

Comparative genomics has various applications, one of which is understanding the evolution of genomes under the assumption that gene content and gene order conservation are closely related to gene function [11]. To this end, a fundamental task is to define and compute the true evolutionary distance between two given genomes [15]. This is done by the correct identification of orthologs and paralogs and by the correct identification of the evolutionary events resulting into changes in gene content and gene order. The first of these objectives is handled by several homology-based approaches [12, 16]; more evolved programs handle both objectives [1, 4, 13]. The second objective gave birth to a large number of important distances between genomes represented either as strings or as permutations. Such distances either exploit the similarity between genomes in terms of gene content and order, or count specific genome rearrangements needed to transform one genome into another (see [3] for an extensive survey).

In this work, both objectives above are followed *via* a distance between genomes represented as strings, which was defined independently by Chen et al. [1] (for ortholog/paralog identification) and Swenson et al. [15] (for evolutionary events defining an evolutionary distance). Informally, given two strings S_1 and S_2 representing two genomes, the operation to realize is cutting S_1 into non-overlapping substrings and reordering a subset of these substrings such that the concatenation of the re-ordered substrings is as close as possible to S_2 . The ortholog/paralog identification

* Post-doc funded by a Région Pays de la Loire grant.

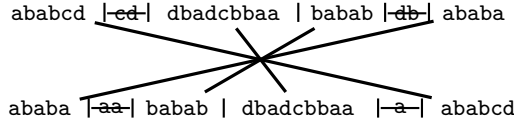


Fig. 1. A common string partition of size 4. Copies of a, b, c and d that could not be matched are deleted.

between S_1 and S_2 is then directly given by the substrings of S_1 used to approximately recompose S_2 , whereas the evolutionary distance is given by the minimum number of substrings needed to obtain such a reconstruction.

The above transformation between the two genomes is formalized by the notion of common string partition (CSP). Let S_1 and S_2 be two strings on an alphabet Σ . A partition P of S_1 and S_2 into blocks $x_1x_2 \cdots x_p$ and $y_1y_2 \cdots y_q$ is a *common string partition* if there is a bijective function M from $D(M) \subseteq \{x_i \mid 1 \leq i \leq p\}$ to $I(M) \subseteq \{y_j \mid 1 \leq j \leq q\}$ such that (1) for each $x_i \in D(M)$, x_i is the same string as $M(x_i)$, and (2) there is no letter $a \in \Sigma$ that is simultaneously present in some block $x_j \notin D(M)$ and in some block $y_l \notin I(M)$ (see Fig. 1 for an example). The *size* of the common string partition P is the cardinality k of $D(M)$. We study the problem of finding a minimum-size CSP:

MINIMUM COMMON STRING PARTITION (MCSP)

Input: Two strings S_1 and S_2 on an alphabet Σ , and an integer k .

Question: Is there a common string partition (CSP) of S_1 and S_2 of size at most k ?

The definition of a CSP given above is actually a generalization to arbitrary (or *unbalanced*) strings of the definition given in [1] for *balanced* strings, that is, when each letter appears the same number of times in S_1 and S_2 . Note also that in this paper, the strings we consider are unsigned. Although this model is less realistic from a genomic viewpoint, our study is a first step towards improved algorithms for the MCSP problem in the most general case, that is, for signed and unbalanced strings.

Related Work. MCSP was introduced by Chen et al. [1], but close variants also exist with different names, such as *block edit distance* [10] or *sequence cover* [15]. Most of the literature on MCSP actually considers the restricted case where the input strings S_1 and S_2 are balanced. In that case, necessarily $D(M)$ (resp. $I(M)$) contains every block from S_1 (resp. S_2). Let Bal-MCSP denote this restricted class of problems. Bal-MCSP has been shown to be NP-hard and APX-hard even if $d = 2$, where d is the maximum number of occurrences of any letter in either input string [5]. Several approximation algorithms exist with ratios 1.1037 when $d = 2$ [5], 4 when $d = 3$ [5], and $4d$ in general [9]. Concerning fixed-parameter tractability issues, Damaschke [2] initiated the study of Bal-MCSP in the context of parameterized algorithmics by showing that it is fixed-parameter tractable with respect to the combined parameter “partition size k and repetition

number r ". More recently, Jiang et al. [6] showed that Bal-MCSP can be solved in $O((d!)^k \cdot \text{poly}(n))$ time.

Our Results. Our main result in this paper is an improvement on the latter result, showing that MCSP (and thus, Bal-MCSP) can be solved in $O(d^{2k} \cdot kn)$ time, thus considerably improving the running time from Jiang et al. [6]. Our result is also more general since it is one of the rare known fixed-parameter algorithms that deals with unbalanced strings. Moreover, a(n approximate) solution to MCSP is computed within the pipeline of MSOAR, MSOAR2.0 and MultiM-SOAR software [4, 13, 14] (all used to determine orthology relations between genes), hence these programs could benefit from any algorithmic improvement concerning MCSP [7], such as the one presented here. Indeed, our algorithm actually runs in $d^{2k'} \cdot kn$, where k' is the number of blocks of $D(M)$ that contain *no letter appearing only once* in S_1 and S_2 . Moreover, we present reduction rules that yield further speed-up, and finally test our algorithm on genomic and synthetic data.

Basic Notation. A *marker* is an occurrence of a letter at a specific position in a string. Formally, the marker at position i in a string S corresponds to the pair (S, i) , which we denote by $S[i]$. Given a marker u we denote by $S(u)$ the string that contains u . For all i , $1 \leq i < n$, the markers $S[i]$ and $S[i + 1]$ are called *consecutive*. Let $r(S[i]) := S[i + 1]$, $1 \leq i < n$, denote the right neighbor of marker $S[i]$, and let $l(S[i]) := S[i - 1]$, $1 < i \leq n$ denote the left neighbor of marker $S[i]$. An *adjacency* is a pair of consecutive markers. For two markers u and v we write $u \equiv v$ if their letters are the same and $u = v$ if the markers are identical, that is, they are at the same position in the same string. An *interval* is a set of consecutive markers, that is, an interval is a set $\{S[i], S[i + 1], \dots, S[j]\}$ for some $i \leq j$. We write $[u, v]$ to denote the interval whose first marker is u and whose last marker is v . For two intervals s and t , we write $s \equiv t$ if they represent the same string of letters (if they have the same contents) and $s = t$ if they are the same interval, that is, they start and end at the same position in the same string. Given two strings S_1, S_2 , a letter is *abundant* in a string S_i if it appears with strictly more occurrences in S_i than in the other string. Otherwise, it is *rare* in S_i . A marker u is *abundant* if it corresponds to an abundant letter in $S(u)$, and *rare* otherwise.

Fundamental CSP-Related Definitions. We assume that $S_1 \neq S_2$, otherwise MCSP is trivially solved by reporting a CSP of size one. A *candidate match* is an unordered pair of markers $\{u, v\}$ such that $u \equiv v$ and $S(u) \neq S(v)$, that is, the markers have the same letters and are from different input strings. Two candidate matches $\{x, y\}$ and $\{x', y'\}$ where $S(x) = S(x')$ and x is to the left of x' are called *parallel* if $[x, x'] \equiv [y, y']$. Note that this implies that for the i -th marker u in $[x, x']$ and the i -th marker v in $[y, y']$ the pair $\{u, v\}$ is also a candidate match and it is parallel to $\{x, y\}$ and to $\{x', y'\}$. Informally, being parallel means that two candidate matches could potentially be in the same block of a CSP.

A CSP P is a set of pairwise disjoint candidate matches containing all rare markers. If a marker does not appear in any candidate match of P then it is necessarily abundant, and it is called *deleted* in P , otherwise we use $f_P(u)$ to denote the unique marker v such that $\{u, v\} \in P$. The *block relation* \sim_P of a CSP is defined as the (uniquely determined) equivalence relation such that each equivalence class is a substring of S_1 or S_2 and $u \sim_P r(u)$ if and only if u and $r(u)$ are not deleted, and $\{u, f_P(u)\}$ and $\{r(u), f_P(r(u))\}$ are parallel. Note that this implies that, for any two markers x and x' with $x \sim_P x'$ it holds that $\{x, f_P(x)\}$ and $\{x', f_P(x')\}$ are parallel. The blocks are precisely the equivalence classes of \sim_P of non-deleted markers, that is, two markers u and v are in the same block iff $u \sim_P v$.

Due to lack of space, some proofs are deferred to a full version of this work.

2 An Improved Fixed-Parameter Algorithm

We now describe our fixed-parameter algorithm. It is a branching algorithm that adds, one by one, candidate matches to a temporary solution. The main idea is that these candidate matches belong to different blocks of the CSP.

2.1 CSPs, Samples and Witnesses

As stated above, the algorithm gradually extends a temporary solution called sample. Formally, a *sample* T is a set of disjoint candidate matches. We use $\mathcal{M}(T)$ to denote the set of all markers belonging to a candidate match in T (thus, $|\mathcal{M}(T)| = 2|T|$). The algorithm tries to construct an optimal CSP by extending a sample T that describes this CSP and is furthermore non-redundant. That is, the sample contains only candidate matches that are in the CSP and at most one candidate match for each pair of matched blocks. We call such samples witnesses.

Definition 1. A sample $T = \{\{x_1, y_1\}, \{x_2, y_2\}, \dots, \{x_m, y_m\}\}$ is a witness of a CSP P if (1) $T \subseteq P$, that is, $y_i = f_P(x_i)$ for each i , and (2) for all $x, y \in \mathcal{M}(T)$ with $x \neq y$ we have $x \not\sim_P y$.

Given a witness T of some CSP P , a marker u is *seen by* T if $\exists x \in \mathcal{M}(T)$ such that $u \sim_P x$. We use $\text{See}(P, T)$ to denote the set of markers seen by T in P . Let $u \in \text{See}(P, T)$ be a marker seen by T in P , then we say that u is *colored black by* P and T if $u = x$; u is *colored green by* P and T if it is to the right of x ; or u is *colored red by* P and T if it is to the left of x . Note that the coloring is unique since for each marker u there is at most one $x \in \mathcal{M}(T)$ such that $u \sim_P x$.

The algorithm finds a witness describing an optimal CSP. More precisely, the aim is to see all rare markers eventually. A witness T is *complete* if it contains a marker from every block of P . Equivalently, T is complete if it sees every rare marker. We first show that if a rare marker is unseen by a witness T for some CSP P , then another witness for P can be obtained by extending T .

Lemma 1. Let u be a rare marker such that $u \notin \text{See}(P, T)$. Then there exists a candidate match $\{u, v\}$ such that $T \cup \{\{u, v\}\}$ is a witness of P .

Proof. Let $v = f_P(u)$ (u is rare, hence it is not deleted), then $\{u, v\}$ is clearly a candidate match. Furthermore, $T \cup \{\{u, v\}\}$ is a subset of P . It thus remains to show that T is non-redundant. Since $u \notin \text{See}(P, T)$, $u \not\sim_P x$ for all $x \in \mathcal{M}(T)$. Furthermore, this also implies $v \not\sim_P y$ for all $y \in \{f_P(x) \mid x \in \mathcal{M}(T)\} = \mathcal{M}(T)$. Thus $T \cup \{\{u, v\}\}$ is a witness of P . \square

The following lemma shows that when an optimal CSP contains parallel candidate matches, then the markers that are in the same string are also in the same blocks of the CSP. We will use this lemma to argue that the algorithm only considers samples without parallel edges.

Lemma 2. *If a CSP P contains two parallel candidate matches $\{x, y\}$ and $\{x', y'\}$ such that $S(x) = S(x')$ and $x \not\sim_P x'$, then it is not optimal.*

Proof. Aiming at a contradiction, assume that P is optimal. Moreover, assume without loss of generality that $S(x) = S(x') = S_1$, and that $\{x, y\}$ and $\{x', y'\}$ have been chosen so as to minimize the distance between x and x' , while satisfying the conditions of the lemma. Since the candidate matches $\{x, y\}$ and $\{x', y'\}$ are parallel, we have $[x, x'] \equiv [y, y']$. Let ℓ denote the number of markers in $[x, x']$, let x_i denote the i -th marker in $[x, x']$ and let y_i denote the i -th marker in $[y, y']$. Then, each $\{x_i, y_i\}$ is a candidate match, $\{x_i, y_i\}$ and $\{x_j, y_j\}$ are parallel for all $1 \leq i, j \leq \ell$, and, by the minimality of the distance between x and x' , $\{x_i, y_i\} \notin P$ for $1 < i < \ell$. Moreover, for all $1 < i < \ell$, $x \not\sim_P x_i \not\sim_P x'$ and $y \not\sim_P y_i \not\sim_P y'$. Create a CSP Q , starting with $Q := P$.

If one of x_2, y_2 is deleted (say x_2 , note that they cannot both be deleted since they cannot both be abundant), then let $u_2 := f_P(y_2)$. The pair $\{u_2, y_2\}$ is the left-most candidate match of its block in P . Remove $\{u_2, y_2\}$ from Q and add $\{x_2, y_2\}$, extending the block containing $\{x, y\}$. Then Q is also an optimal CSP.

If none of x_2, y_2 are deleted, then they are the left-most markers of blocks ending in x_p and y_q respectively (assume without loss of generality that $p \leq q$). Note that $p, q < \ell$, since these blocks are strictly contained between x and x' (y and y'). Write $u_i = f_P(y_i)$ for all $2 \leq i \leq q$, and $v_i = f_P(x_i)$ for all $2 \leq i \leq p$. For each $2 \leq i \leq p$, remove $\{\{x_i, v_i\}, \{y_i, u_i\}\}$ from Q and add $\{\{x_i, y_i\}, \{u_i, v_i\}\}$. Then Q has no more blocks than P and is an optimal CSP. Indeed, $[x_2, x_p]$ is now merged to the block containing x , and $[u_2, u_q]$ is now split in two blocks $[u_2, u_p]$ and $[u_{p+1}, u_q]$.

In both cases, Q is an optimal CSP where $\{x_2, y_2\}$ has been added to the block containing $\{x, y\}$. If $x_2 \sim_Q x'$, then the block containing $\{x, y\}$ and $\{x_2, y_2\}$ is merged with $\{x', y'\}$, and Q has one block less than P . Otherwise, $x_2 \not\sim_Q x'$, and Q satisfies the conditions of the lemma for $\{x_2, y_2\}$ and $\{x', y'\}$ with a smaller distance between x_2 and x' than between x and x' . Both cases lead to a contradiction. \square

2.2 The Sample Graph

We now describe a multigraph that is associated with the current sample T . We will use the structure in this graph to identify cases to which the branching applies. First, we describe the construction of this graph.

belong to T . Second, $\{l(u), l(v)\}$ is to the left of $\{u, v\}$ and thus it is also parallel to $\{l_T(u), l_T(v)\}$. \square

Property 2. Each vertex incident with a black edge has degree one. For each other vertex, green-degree and red-degree are at most one.

Proof. First, let $\{x, y\} \in T$ be a black edge. By the definitions of E_T^g and E_T^r , neither x nor y is incident with a red or green edge. Since the sample T has only pairwise disjoint candidate matchings, there is no other black edge in T incident with either x or y .

Now, let e_1, e_2 be two green edges incident with some vertex v . Clearly, e_1 and e_2 fulfill the conditions in the definition of E_T^g . Note that, by Property 2, $l_T(v)$ has degree one. Hence, e_1 and e_2 are parallel to the same edge. This implies $e_1 = e_2$. The proof for red edges is symmetrical. \square

Property 2 implies that every vertex has degree at most two. Thus, each connected component is either a singleton, a path or a cycle.

Property 3. Let u and $u' := l(u)$ be two consecutive markers such that G_T contains the edges $\{u, v\}$ and $\{u', v'\}$. If both edges are green (both edges are red), then $\{u, v\}$ and $\{u', v'\}$ are parallel, that is, $v' = l(v)$.

Proof. Assume that $\{u, v\}$ and $\{u', v'\}$ are green. By Property 2, vertices incident with black edges have degree one. Hence, $l_T(u) \neq u'$ and thus $l_T(u) = l_T(u')$. Consequently, $\{u, v\}$ and $\{u', v'\}$ are parallel to the same edge $\{l_T(u), l_T(v)\}$. Hence, they are also parallel to each other. The proof for red edges works analogously. \square

2.3 Branching on Odd Connected Components

We now show some further properties that the sample graph G_T has with respect to any CSP witnessed by the sample T . We then exploit these properties to devise branching rules that branch into $O(d^2)$ cases. Hence, consider an arbitrary CSP P witnessed by T . The following is a simple corollary of Lemma 2, the construction of the sample graph, and the definition of witness.

Lemma 3. *If G_T contains two parallel black edges, then P is not optimal.*

The following lemma relates the colors that markers receive by the CSP P to the edge colors in the sample graph.

Lemma 4. *Let $u \in \text{See}(P, T)$ be a marker seen by T . Then, there is at least one edge incident with u in G_T . In particular, if vertex u is colored black/green/red, then $\{u, f_P(u)\}$ is a black/green/red edge in G_T .*

Corollary 1. *If some vertex u has degree 0 in G_T , then $u \notin \text{See}(P, T)$.*

Combined with Lemma 1 this leads to the first branching rule.

Branching Rule 1. *If the sample graph G_T contains a rare degree-0 vertex u , then for each vertex $v \notin \mathcal{M}(T)$ such that $S(u) \neq S(v)$ and $u \equiv v$ branch into the case to add $\{u, v\}$ to T .*

The branching rule above deals with connected components that are singletons. Next, we develop a branching rule for connected components that are a certain type of path in the sample graph.

To this end, we distinguish the following types of paths. A *black path* is a path containing exactly one black edge. An *odd path* is a path with an odd number of vertices. An *even path* is a path with an even number of vertices. Note that by Property 2, all black edges are contained exclusively in black paths. Furthermore, also by Property 2, the colors of each other path alternate between green and red. We thus call an even path *green* if it starts and ends with a green edge and *red*, otherwise. An odd path is *abundant* if its first marker is abundant, *rare* otherwise (this definition is not ambiguous since the markers at the ends of an even path correspond to the same letter in the same string, they are thus both abundant or both rare).

Lemma 5. *Let T be a sample witnessing a CSP P . If a connected component of the sample graph G_T is a rare odd path $(u_1, v_1, \dots, v_{\ell-1}, u_\ell)$, then there is some u_i , $1 \leq i \leq \ell$, such that $u_i \notin \text{See}(P, T)$ and u_i is not deleted in P .*

Branching Rule 2. *If the sample graph G_T contains a connected component which is a rare odd path $(u_1, v_1, u_2, v_2, \dots, v_{\ell-1}, u_\ell)$, then do the following for each vertex u_i , $1 \leq i \leq \ell$: for each vertex $x \notin \mathcal{M}(T)$ such that $S(u_i) \neq S(x)$ and $u_i \equiv x$ branch into the case to add $\{u_i, x\}$ to T .*

2.4 Solving Instances without Rare Odd Paths or Singletons

We now show how to find an optimal CSP in the remaining cases. As we will show, the edge set defined as follows gives such an optimal CSP. See Fig. 3 for an example.

Definition 2. *Let G_T be a sample graph. The set P_T is the edge set containing*

- all black edges from G_T ,
- each green edge that is in a green path, in an odd path or in a cycle, and
- each red edge that is in a red path.

Lemma 6. *Let T be a sample such that G_T does not contain isolated vertices, parallel black edges, or rare odd paths. Then, P_T is a CSP for which T is a complete witness.*

Theorem 1. *MCSP can be solved in $O(d^{2k} \cdot kn)$ time.*

Proof. We use the algorithm MCSP outlined in Algorithm 1. We first show the correctness of MCSP, then we bound the running time. Consider a yes-instance, and let P be an optimal CSP of size k . We show that $\text{MCSP}(S_1, S_2, k, \emptyset)$ outputs at



Fig. 3. Left: Sample graph G_T with no isolated vertices, parallel black edges or rare odd paths. Right: CSP P_T obtained from G_T (Definition 2 and Lemma 6). Note that markers (with letter) u form a green path, markers v form a cycle, markers w form a red path, and markers x form an abundant odd path.

Algorithm 1. The fixed-parameter algorithm for parameter (d, k) .

```

MCSP( $S_1, S_2, k, T$ )
1  if  $|T| > k$  abort branch
2  Compute the sample graph  $G_T$ 
3  if  $G_T$  contains parallel black edges : abort branch
4  else if  $G_T$  contains an isolated vertex :
5    apply Branching Rule 1; in each case call  $\text{MCSP}(S_1, S_2, k, T \cup \{u, v\})$ 
6  else if  $G_T$  contains a rare odd path :
7    apply Branching Rule 2; in each case call  $\text{MCSP}(S_1, S_2, k, T \cup \{u_i, x\})$ 
8  else compute  $P_T$ , output  $P_T$ 
    
```

least one CSP of size k in this case. Since $T = \emptyset$ T in the first call, T is initially a witness of P . Combining Lemma 1 with Lemmas 4 and 5 shows that the algorithm creates in each application at least one branch such that the set T is a witness of P in this branch. Now, note that if a branch is aborted because $|T| > k$, then the current set T either is redundant (and thus not a sample) or any CSP that it witnesses has size at least k , thus it does not witness P in this case. Similarly, if the graph G_T contains parallel black edges, then the set T is either redundant, or any CSP that it witness is not optimal; thus it does not witness P . Hence, the algorithm eventually reaches a situation in which T is a witness of P and G_T contains no isolated vertices and no odd paths. Then it constructs and outputs a set P_T . By Lemma 6, P_T is a CSP. Furthermore, it has size $|T|$ and thus it is at most as large as P which also has size at least $|T|$ since T is a witness for P .

Now, assume that the instance is a no-instance, then the algorithm has empty output since all CSPs that are output have size at most k due to the condition in Line 1 of the algorithm.

It remains to bound the running time. We first bound the size of the search tree. After the application of each branching rule, the set T has contains one additional candidate match, so the depth of the search tree is at most k because of the check in Line 1 of the algorithm. We now bound the number of new cases for each branching rule. First, Branching Rule 1 branches into at most d cases. Second, Branching Rule 2 branches into at most d^2 cases: All vertices of a path

have the same letter since edges are candidate matches. Hence, there are at most d u_i 's. For each of them the algorithm creates at most d branches. Hence, the overall search tree size is $O((d^2)^k) = O(d^{2k})$. The time spent in each search tree node can be seen as follows. The sample graph can be constructed in $O(kn)$ time by adding for each of the $O(k)$ black edges the red and green edges in linear time. This is done by moving to the left and the right until either the next parallel marker pair is not a candidate match or contains a black vertex (this can also be used to find parallel black edges). The sample graph has size $O(n)$, hence isolated vertices and odd paths can also be found in $O(n)$ time. \square

3 Parameter Improvement

In this section, we show that the parameter k denoting the number of blocks in an optimal solution can be replaced by a potentially much smaller parameter $k' :=$ “number of blocks without unique letters”. Herein, a letter is called *unique* if it appears at most once in S_1 and at most once in S_2 . To deal with the blocks that contain unique letters we devise a simple rule for simplifying the instance. The algorithm makes use of a data reduction rule. A data reduction rule is *correct* if the new instance is a yes-instance if and only if the old one is. An instance is *reduced* with respect to a data reduction rule, if an application of the rule does not change the instance.

Rule 1. *If the input contains a pair of unique letters x and x' , where x' is to the right of x , such that the candidate matches $\{x, y\}$ for x and $\{x', y'\}$ for x' are parallel, then replace $[x, x']$ by x and $[y, y']$ by y .*

Lemma 7. *Rule 1 is correct.*

After this simplification, the resulting instance has the property that candidate matches between two different unique matches are in different blocks. This implies that a set T containing all different unique matches is a sample witnessing any optimal CSP. This leads to the following.

Theorem 2. *MCSP can be solved in $O(d^{2k'} \cdot kn)$ time where k' denotes the number of blocks in S_1 that contain no unique letter.*

4 Data Reduction Rules

In addition to the improvements described in previous sections which lead to an improved worst-case running time bound, we also devise the following data reduction rules. These rules proved crucial for solving larger instances of MCSP and may be of independent interest. The first of these reduction rules identifies unique letters that are in S_1 and S_2 surrounded by other unique letters.

Rule 2. *If the instance contains a unique candidate match $\{u, v\}$ and the letters to the right and left of u and v are also unique, then let $L(u)$, $R(u)$, $L(v)$, and $R(v)$ denote the uniquely defined candidate matches containing the left and right neighbor of u or v . Remove u and v from S_1 and S_2 and do the following.*

- If $L(u) = L(v)$ or $R(u) = R(v)$ leave k unchanged.
- Else, check whether removing u and v from S_1 and S_2 made either $L(u)$ and $R(u)$ parallel or $L(v)$ and $R(v)$ parallel. If it makes none of the two parallel, then decrease k by one, if it makes exactly one pair parallel, decrease k by two, otherwise decrease k by three.

Proof (of correctness). In the first case, $\{u, v\}$ is parallel to either $L(u)$ or $R(u)$ and thus the rule is simply a special case of the parallel rule. In the other cases, $\{u, v\}$ is parallel to none of $L(u), R(u), L(v), R(v)$. Hence, u and v will be in a block of size one in any CSP. In case the removal of $\{u, v\}$ makes no other edges parallel, the minimum size of a CSP in the reduced instance thus is one less. Hence, the parameter decrement is correct in this case. If the removal of u and v makes only $L(u)$ and $R(u)$ parallel, then the minimum size of a CSP after removing u is decreased by exactly two: Consider any CSP of the original instance, “merging” the blocks containing the left and the right neighbor of u and removing the blocks containing u and v gives a CSP for the reduced instance with size decreased by two. Similarly, re-adding $\{u, v\}$ to any CSP of the reduced instance increases the size by exactly two. By symmetry, the same holds for the case that the removal of u and v makes only $L(v)$ and $R(v)$ parallel.

Finally, if the removal makes $L(u)$ and $R(u)$ parallel and $L(v)$ and $R(v)$ parallel, then the size of the minimum CSP decreases by exactly three which follows from the above arguments with the additional observation that the two block merges are indeed “different”. \square

The next two rules “split” letters into two “subletters”. The first rule looks for letters that appear once in one sequence and twice in the other.

Rule 3. *If there is a marker v such that there is exactly one candidate match $\{u, v\}$ containing v , the marker u has at least one further candidate match $\{u, w\}$, and any CSP which contains $\{u, v\}$ has u and v in blocks of size one, then change the letter of v to some previously unused letter z .*

Proof (of correctness). Any CSP P of size k containing the candidate match $\{u, v\}$ can be transformed into a CSP of size at most k containing the candidate match $\{u, w\}$: Since u and v are in P in blocks of size one, replacing $\{u, v\}$ by $\{u, w\}$ does not decrease the number of adjacencies in the blocks of the CSP. Furthermore, this exchange is possible, since $\{u, w\}$ is the only candidate match containing w . Hence, there is an optimal CSP in which v is not contained in any candidate match. It is thus safe to assign v some new unused letter. \square

The next rule follows the same idea, only with letters that appear twice.

Rule 4. *If there is a set of four markers u, v, w , and z such that $\{u, w\}, \{u, z\}, \{v, w\}, \{v, z\}$ are the only four candidate matches containing at least one of these markers, and any CSP which contains $\{u, w\}$ and $\{v, z\}$ has u, v, w , and z in blocks of size one, then change the letter of u and z to some previously unused letter x .*

Proof (of correctness). The proof is similar to the proof of Rule 3. Since the blocks containing u , v , w , and z have size one, changing the candidate matches does not decrease the number of adjacencies in the blocks. Hence, replacing $\{u, w\}$ and $\{v, z\}$ by $\{u, z\}$ and $\{v, w\}$ gives a CSP of the same size. \square

Note that checking whether there is any CSP including some match $\{u, v\}$ that has u and v in blocks of size at least two can be done by simply checking whether $\{u, v\}$ is parallel to a candidate match of its right or left neighbor.

5 Implementation and Experiments

We implemented the described algorithm to assess its performance on genomic and on synthetic instances. We furthermore added three additional data reduction rules and demonstrate their effect on the genomic instances. Although our algorithm and experiments concern unsigned strings, they can be seen as a first step; the results being more than encouraging, we will adapt, in the near future, our algorithm to the signed (and unbalanced) case. We ran all our experiments on an Intel(R) Core(TM) i5 M 450 CPU 2.40GHz machine with 2GB memory under the Ubuntu 12.04 operating system. The program is implemented in Java and runs under Java 1.6. The source code is available from <http://fpt.akt.tu-berlin.de/mcsp/>. The search tree is implemented as described in Sections 2 and 3. In addition to the data reduction rules described in Section 4, we apply Rule 1. All data reduction rules are applied in the beginning and also in each search tree node.

Genomic Data. We performed experiments with genomic data from several bacteria. The data was obtained as follows. The raw data consists of a file containing transcripts and proteins of the species and positional information of the corresponding genes. This data was downloaded from the EnsemblBacteria database [8] and then filtered as described by Shi et al. [13] to obtain input data for MSOAR 2.0. Then, the MSOAR 2.0 pipeline was invoked, and the MCSP instances are output right before they are solved approximately by the vertex cover 2-approximation algorithm. These instances contain signed genes. Since the presented correctness proof only solves the unsigned MCSP problem, we removed all genes from the negative strand. Afterwards, we removed all non matched genes. Finally, we perform the following modification: the data from MSOAR actually can allow arbitrary candidate matches between markers in S_1 and S_2 . However in MCSP the candidate matches are “transitive”, that is, if $\{u, v\}$, $\{v, w\}$, and $\{w, x\}$ are candidate matches of an MCSP instance, then $\{u, x\}$ is also a candidate match. We achieve this property for the input data by adding the candidate match $\{u, x\}$, that is, every connected component of the “marker-match” graph is assigned one letter not used elsewhere.

The species under consideration are *Borrelia burgdorferi*, *Treponema pallidum*, *Escherichia coli*, *Bacillus subtilis*, and *Bacillus thuringiensis*. Our results are shown in Table 1; the main findings are as follows. We can solve instances with hundreds of genes if the average number d^* of occurrences for each letter and the

Table 1. Running time, instance properties and effect of data reduction on genomic data. Herein, n_1 is the number of markers in the first genome, n_2 the number of markers in the second genome, k is the CSP size, k' the number of blocks without fixed markers, d^* the average number of candidate matches for each marker, n'_1 and n'_2 denote the respective number of markers after data reduction, δ is the number of removed candidate matches during data reduction, and t is the running time in seconds.

Species 1	Species 2	n_1	n_2	k	k'	d	d^*	n'_1	n'_2	δ	t
<i>B. burg.</i>	<i>T. pall.</i>	91	93	68	0	3	1.02	13	15	4	0.06
<i>B. burg.</i>	<i>E. coli</i>	66	72	59	0	6	1.09	22	28	12	0.22
<i>B. burg.</i>	<i>B. sub.</i>	83	91	63	3	6	1.16	31	39	11	0.15
<i>B. burg.</i>	<i>B. thur.</i>	61	71	51	3	5	1.19	32	42	11	0.09
<i>T. pall.</i>	<i>E. coli</i>	89	93	78	2	5	1.09	22	26	7	0.35
<i>T. pall.</i>	<i>B. sub.</i>	136	144	82	0	7	1.12	23	31	11	0.18
<i>T. pall.</i>	<i>B. thur.</i>	116	128	76	0	6	1.16	30	42	16	0.15
<i>E. coli</i>	<i>B. sub.</i>	264	287	234	14	7	1.23	128	151	54	41.06
<i>E. coli</i>	<i>B. thur.</i>	249	282	221	12	10	1.24	129	162	59	18.64
<i>B. sub.</i>	<i>B. thur.</i>	673	693	340	14	8	1.17	173	193	51	249.71

Table 2. Average running time in seconds for synthetic instances with $d = 6$ and $d = 8$, $n = 1000$ and varying k ; for each parameter triple, 50 instances were generated

$d = 6$		$d = 8$	
k	running time	k	running time
50	0.06	50	0.07
60	0.06	60	0.06
70	0.07	70	0.08
80	0.09	80	0.09
90	0.10	90	0.12
100	0.12	100	0.16
110	0.13	110	0.26
120	0.18	120	1.62
130	0.21	130	30.42

number k' of blocks without unique letters is small. Moreover, the parameter k' is in these instances much smaller than the parameter k . Finally, the data reduction rules are very effective in decreasing the instance size and also decrease the overall number of candidate matches somewhat.

Synthetic Data. We also experimented with synthetic data to test how growth of k influences the running time. Each instance is generated randomly given five parameters: the string length n , the upper bound k on the number of blocks, the upper bound d on the number of occurrences, the upper bound f on the number of gene families (size of the alphabet), and finally the number δ of deleted markers (considered as *noise* between the blocks). We randomly generate k blocks using available markers (that is, each block is a random string of markers so that the number of occurrences is never more than d). The two input sequences are

generated by concatenating the blocks in different (random) orders, interleaving with noisy parts of the required total size.

We study the effect of varying parameters n , k and d . To this effect, we fix the number of deleted markers to $\delta = 0.1n$ (we observed that the behavior of the algorithm is uniform for $0 \leq \delta \leq 0.2n$). Values of $\delta > 0.2n$ are harder, however, we assume that deleting too many markers is of less relevance in genomic applications. The number f of gene families is fixed to $3n/d$. This way we obtain an average number of occurrences which is experimentally close to $d/2$. The average occurrence of each letter thus is roughly twice that of the genomic data; this was done to obtain more difficult input.

In the experiments, we set $n = 1000$, and varied k from 50 to 130. One run was performed for $d = 6$ and one for $d = 8$. Our results are shown in Table 2. For each set of parameter values, we generated 50 instances. We make the following main observations. First, increasing d makes the instances much harder. Second, for $d = 6$, the combinatorial explosion sets in at $k \approx 120$, for $d = 8$ this happens already at $k \approx 100$. Finally, the algorithm efficiently solves instances with $n = 1000$ and $k \approx 120$ when the average occurrence of each letter is roughly 3.5 (this is the average occurrence number in the experiments for $d = 8$).

6 Conclusion

We have presented an efficient fixed-parameter algorithm for the MINIMUM COMMON STRING PARTITION problem with parameters k and d . Our algorithm even allows for unbalanced strings, since it can delete superfluous markers between consecutive blocks of the string partition. Looking towards practical applications, it would be interesting to consider signed instances, that is, blocks can be read either from left to right or from right to left with opposite signs. We conjecture that our algorithm can be extended to solve the signed variant of MCSP. Another generalization of MCSP is as follows. Pairs of markers which form candidate matches are given in input, rather than being defined from classes of letters. From a graph theory point of view, the bipartite graph of candidate matches may contain arbitrary connected components, not only complete ones. It would be of interest to provide efficient algorithms for this extension of MCSP.

References

- [1] Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: Assignment of orthologous genes via genome rearrangement. *IEEE/ACM T. Comput. Bi.* 2(4), 302–315 (2005)
- [2] Damaschke, P.: Minimum common string partition parameterized. In: Crandall, K.A., Lagergren, J. (eds.) *WABI 2008*. LNCS (LNBI), vol. 5251, pp. 87–98. Springer, Heidelberg (2008)
- [3] Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: *Combinatorics of Genome Rearrangements*. Computational Molecular Biology (2009)

- [4] Fu, Z., Chen, X., Vacic, V., Nan, P., Zhong, Y., Jiang, T.: MSOAR: A high-throughput ortholog assignment system based on genome rearrangement. *J. Comput. Biol.* 14(9), 1160–1175 (2007)
- [5] Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partition problem: Hardness and approximations. *Electron. J. Comb.* 12 (2005)
- [6] Jiang, H., Zhu, B., Zhu, D., Zhu, H.: Minimum common string partition revisited. *J. Comb. Optim.* 23, 519–527 (2012)
- [7] Jiang, T.: Some algorithmic challenges in genome-wide ortholog assignment. *J. Comput. Sci. Technol.* 25(1), 42–52 (2010)
- [8] Kersey, P.J., Staines, D.M., Lawson, D., Kulesha, E., Derwent, P., Humphrey, J.C., Hughes, D.S.T., Keenan, S., Kerhornou, A., Koscielny, G., Langridge, N., McDowall, M.D., Megy, K., Maheswari, U., Nuhn, M., Paulini, M., Pedro, H., Toneva, I., Wilson, D., Yates, A., Birney, E.: Ensembl genomes: an integrative resource for genome-scale data from non-vertebrate species. *Nucleic Acids Res.* 40(Database-Issue), 91–97 (2012)
- [9] Kolman, P., Walen, T.: Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electr. J. Comb.* 14(1) (2007)
- [10] Lopresti, D.P., Tomkins, A.: Block edit models for approximate string matching. *Theor. Comput. Sci.* 181(1), 159–179 (1997)
- [11] Overbeek, R., Fonstein, M., D’Souza, M., Pusch, G.D., Maltsev, N.: The use of gene clusters to infer functional coupling. *PNAS* 96(6), 2896–2901 (1999)
- [12] Remm, M., Storm, C.E., Sonnhammer, E.L., et al.: Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J. Mol. Biol.* 314(5), 1041–1052 (2001)
- [13] Shi, G., Zhang, L., Jiang, T.: MSOAR 2.0: Incorporating tandem duplications into ortholog assignment based on genome rearrangement. *BMC Bioinformatics* 11, 10 (2010)
- [14] Shi, G., Peng, M.-C., Jiang, T.: Multisoar 2.0: An accurate tool to identify ortholog groups among multiple genomes. *PloS One* 6(6), e20892 (2011)
- [15] Swenson, K.M., Marron, M., Earnest-DeYoung, J.V., Moret, B.M.E.: Approximating the true evolutionary distance between two genomes. *ACM J. Exp. Alg.* 12 (2008)
- [16] Tatusov, R.L., Natale, D.A., Garkavtsev, I.V., Tatusova, T.A., Shankavaram, U.T., Rao, B.S., Kiryutin, B., Galperin, M.Y., Fedorova, N.D., Koonin, E.V.: The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucleic Acids Res.* 29(1), 22–28 (2001)

MSARC: Multiple Sequence Alignment by Residue Clustering

Michał Modzelewski and Norbert Dojer

Institute of Informatics, University of Warsaw, Poland
dojer@mimuw.edu.pl

Abstract. Progressive methods offer efficient and reasonably good solutions to the multiple sequence alignment problem. However, resulting alignments are biased by guide-trees, especially for relatively distant sequences.

We propose MSARC, a new graph-clustering based algorithm that aligns sequence sets without guide-trees. Experiments on the BALIBASE dataset show that MSARC achieves alignment quality similar to best progressive methods and substantially higher than the quality of other non-progressive algorithms. Furthermore, MSARC outperforms all other methods on sequence sets whose evolutionary distances are hardly representable by a phylogenetic tree. These datasets are most exposed to the guide-tree bias of alignments.

MSARC is available at <http://bioputer.mimuw.edu.pl/msarc>

Keywords: multiple sequence alignment, stochastic alignment, graph partitioning.

1 Introduction

Determining the alignment of a group of biological sequences is among the most common problems in computational biology. The dynamic programming method of pairwise sequence alignment can be readily extended to multiple sequences but requires the computation of an n -dimensional matrix to align n sequences. Consequently, this method has an exponential time and space complexity.

Progressive alignment [21] offers a substantial complexity reduction at the cost of possible loss of the optimal solution. Within this approach, subset alignments are sequentially pairwise aligned to build the final multiple alignment. The order of pairwise alignments is determined by a guide-tree representing the phylogenetic relationships between sequences.

There are two drawbacks of the progressive alignment approach. First, the accuracy of the guide-tree affects the quality of the final alignment. This problem is particularly important in the field of phylogeny reconstruction, because multiple alignment acts as a preprocessing step in most prominent methods of inferring a phylogenetic tree of sequences. It has been shown that, within this approach, the inferred phylogeny is biased towards the initial guide-tree [23,11].

Second, only sequences belonging to currently aligned subsets contribute to their pairwise alignment. Even if a guide-tree reflects correct phylogenetic relationships, these alignments may be inconsistent with remaining sequences and the inconsistencies are propagated to further steps. To address this problem, in recent programs [15,2,8,1,17] progressive alignment is usually preceded by *consistency transformation* (incorporating information from all pairwise alignments into the objective function) and/or followed by *iterative refinement* of the multiple alignment of all sequences.

In the present paper we propose MSARC, a new multiple sequence alignment algorithm that avoids guide-trees altogether. MSARC constructs a graph with all residues from all sequences as nodes and edges weighted with alignment affinities of its adjacent nodes. Columns of best multiple alignments tend to form clusters in this graph, so in the next step residues are clustered (see Figure 1a). Finally, MSARC refines the multiple alignment corresponding to the clustering.

Experiments on the BALiBASE dataset [22] show that our approach is competitive with the best progressive methods and significantly outperforms current non-progressive algorithms [20,19]. Moreover, MSARC is the best aligner for sequence sets with very low levels of conservation. This feature makes MSARC a promising preprocessing tool for phylogeny reconstruction pipelines.

2 Methods

MSARC aligns sequence sets in several steps. In a preprocessing step, following Probalign [17], *stochastic alignments* are calculated for all pairs of sequences and consistency transformation is applied to resulting posterior probabilities of residue correspondences. Transformed probabilities, called residue alignment affinities, represent weights of an *alignment graph*¹. MSARC clusters this graph with a top-down hierarchical method (Figure 1c). Division steps are based on the Fiduccia-Mattheyses graph partitioning algorithm [3], adapted to satisfy constraints imposed by the sequence order of residues. Finally, multiple alignment corresponding to resulting clustering is refined with the iterative improvement strategy proposed in Probcons [1], adapted to remove clustering artefacts.

2.1 Pairwise Stochastic Alignment

The concept of stochastic (or probability) alignment was proposed in [13]. Given a pair of sequences, this framework defines statistical weights of their possible alignments. Based on these weights, for each pair of residues from both sequences, the posterior probability of being aligned may be computed. A consensus of highly weighted suboptimal alignments was shown to contain pairs with significant probabilities that agree with structural alignments despite the optimal alignment deviating significantly. Mückstein et al. [14] suggest the use

¹ Our notion of alignment graph slightly differs from the one of Kececioğlu [9]: removing edges between clusters transforms the former into the latter.

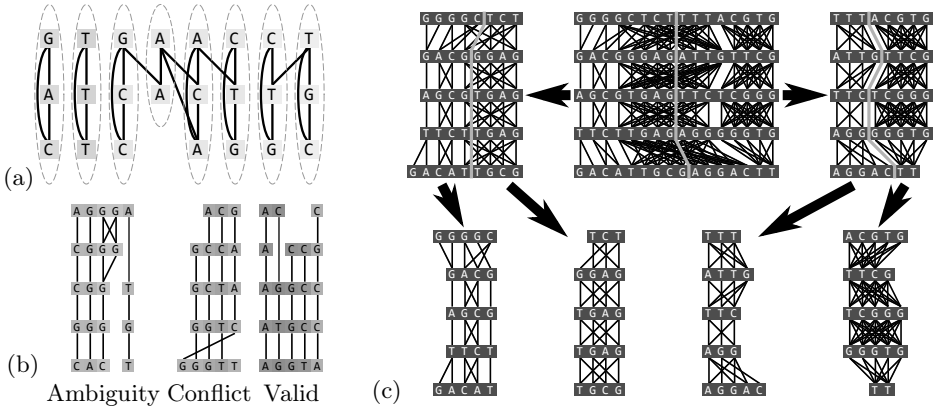


Fig. 1. Overview of our residue clustering approach. (a) Alignment graph and its desired clustering. Clusters form columns of a corresponding multiple sequence alignment. (b) Clusterings inconsistent (left and middle) and consistent (right) with the alignment structure. (c) An example of hierarchical divisive clustering of residues. The graph is recursively partitioned by finding a balanced minimal cut while maintaining the ordering of residues until all parts have at most one residue from each sequence. Final alignment is constructed by concatenating these parts (alignment columns) from left to right.

of the method as a starting point for improved multiple sequence alignment procedures.

The statistical weight $\mathcal{W}(\mathcal{A})$ of an alignment \mathcal{A} is the product of the individual weights of (mis-)matches and gaps [24]. It may be obtained from the standard similarity scoring function $S(\mathcal{A})$ with the following formula:

$$\mathcal{W}(\mathcal{A}) = e^{\beta S(\mathcal{A})} \quad (1)$$

where β corresponds to the inverse of Boltzmann's constant and should be adjusted to the match/mismatch scoring function $s(x, y)$ (in fact, β simply rescales the scoring function).

The probability distribution over all alignments \mathcal{A}^* is achieved by normalizing this value. The normalization factor Z is called the *partition function* of the alignment problem [13], and is defined as

$$Z = \sum_{\mathcal{A} \in \mathcal{A}^*} \mathcal{W}(\mathcal{A}) = \sum_{\mathcal{A} \in \mathcal{A}^*} e^{\beta S(\mathcal{A})} \quad (2)$$

The probability $P(\mathcal{A})$ of an alignment can be calculated by

$$P(\mathcal{A}) = \frac{\mathcal{W}(\mathcal{A})}{Z} = \frac{e^{\beta S(\mathcal{A})}}{Z} \quad (3)$$

Let $\mathbf{P}(a_i \sim b_j)$ denote the posterior probability that residues a_i and b_j are aligned. We can calculate it as the sum of probabilities of all alignments with a_i and b_j in a common column (denoted by $\mathcal{A}_{a_i \sim b_j}^*$):

$$\begin{aligned}
 \mathbf{P}(a_i \sim b_j) &= \sum_{\mathcal{A} \in \mathcal{A}_{a_i \sim b_j}^*} \mathbf{P}(\mathcal{A}) = \frac{\sum_{\mathcal{A} \in \mathcal{A}_{a_i \sim b_j}^*} e^{\beta S(\mathcal{A})}}{Z} = \\
 &= \frac{\left(\sum_{\mathcal{A}_{i-1,j-1}} e^{\beta S(\mathcal{A}_{i-1,j-1})} \right) e^{\beta s(a_i, b_j)} \left(\sum_{\widehat{\mathcal{A}}_{i+1,j+1}} e^{\beta S(\widehat{\mathcal{A}}_{i+1,j+1})} \right)}{Z} = \\
 &= \frac{Z_{i-1,j-1} e^{\beta s(a_i, b_j)} \widehat{Z}_{i+1,j+1}}{Z} \quad (4)
 \end{aligned}$$

Here we use the notation $\mathcal{A}_{i,j}$ for an alignment of the sequence prefixes $a_1 \cdots a_i$ and $b_1 \cdots b_j$, and $\widehat{\mathcal{A}}_{i,j}$ for an alignment of the sequence suffixes $a_i \cdots a_m$ and $b_j \cdots b_n$. Analogously, $Z_{i,j}$ is the partition function over the prefix alignments and $\widehat{Z}_{i,j}$ is the (reverse) partition function over the suffix alignments.

An efficient algorithm for calculating the partition function can be derived from the Gotoh maximum score algorithm [5] by replacing the maximum operations with additions. From a few possible approaches [13,24,14] we chose a variant proposed by Miyazawa [13] and applied in Probalign [17], where insertions and deletions must be separated by at least one match/mismatch position:

$$Z_{i,j}^M = (Z_{i-1,j-1}^M + Z_{i-1,j-1}^E + Z_{i-1,j-1}^F) e^{\beta s(a_i, b_j)} \quad (5)$$

$$Z_{i,j}^E = Z_{i,j-1}^M e^{\beta g_o} + Z_{i,j-1}^E e^{\beta g_{ext}} \quad (6)$$

$$Z_{i,j}^F = Z_{i-1,j}^M e^{\beta g_o} + Z_{i-1,j}^F e^{\beta g_{ext}} \quad (7)$$

$$Z_{i,j} = Z_{i,j}^M + Z_{i,j}^E + Z_{i,j}^F \quad (8)$$

The reverse partition function can be calculated using the same recursion in reverse, starting from the ends of the aligned sequences.

2.2 Alignment Graphs

Probabilities $\mathbf{P}(a_i \sim b_j)$ may be viewed as a representation of a bipartite graph with nodes corresponding to residues a_i and b_j and edges weighted with residue alignment affinity.

Given a set S of k sequences to be aligned, we would like to analogously represent their residue alignment affinity by a k -partite weighted graph. It may be obtained by joining pairwise alignment graphs for all pairs of S -sequences. However, separate computation of edge weights for each pair of sequences does not exploit information included in the remaining alignments. In order to incorporate correspondence with residues from other sequences, we perform a *consistency transformation* [15,1]. It re-estimates the residue alignment affinity according to the following formula:

$$\mathbf{P}'(x_i \sim y_j) \leftarrow \frac{\sum_{z \in S} \sum_{l=0}^{|z|} \mathbf{P}(x_i \sim z_l) \mathbf{P}(z_l \sim y_j)}{|S|} \quad (9)$$

If P_{xy} is a matrix of current residue alignment affinities for sequences x and y , the matrix form equivalent transformation is

$$P'_{xy} \leftarrow \frac{\sum_{z \in S} P_{xz} P_{zy}}{|S|} \quad (10)$$

The consistency transformation may be iterated any number of times, but excessive iterations blur the structure of residue affinity. Following Probalign [17] and ProbCons [1] MSARC performs it twice by default.

2.3 Residue Clustering

Columns of any multiple alignment form a partition of the set of sequence residues. The main idea of MSARC is to reconstruct the alignment by clustering an alignment graph into columns. The clustering method must satisfy constraints imposed by alignment structure. First, each cluster may contain at most one residue from a single sequence. Second, the set of all clusters must be orderable consistently with sequence orders of their residues. Violation of the first constraint will be called *ambiguity*, while violation of the second one – *conflict* (see Figure 1b).

Towards this objective, MSARC applies top-down hierarchical clustering (see Figure 1c). Within this approach, the alignment graph is recursively split into two parts until no ambiguous cluster is left. Each partition step results from a single cut through all sequences, so clusterings are conflict-free at each step of the procedure. Consequently, the final clustering represents a proper multiple alignment.

Optimal clustering is expected to maximize residue alignment affinity within clusters and minimize it between them. Therefore, the partition selection in recursive steps of the clustering procedure should minimize the sum of weights of edges cut by the partition. This is in fact the objective of the well-known problem of *graph partitioning*, i.e. dividing graph nodes into roughly equal parts such that the sum of weights of edges connecting nodes in different parts is minimized.

The Fiduccia-Mattheyses algorithm [3] is an efficient heuristic for the graph partitioning problem. After selecting an initial, possibly random partition, it calculates for each node the change in cost caused by moving it between parts, called *gain*. Subsequently, single nodes are greedily moved between partitions based on the maximum gain and gains of remaining nodes are updated. The process is repeated in *passes*, where each node can be moved only once per pass. The best partition found in a pass is chosen as the initial partition for the next pass. The

algorithm terminates when a pass fails to improve the partition. Grouping single moves into passes helps the algorithm to escape local optima, since intermediate partitions in a pass may have negative gains. An additional balance condition is enforced, disallowing movement from a partition that contains less than a minimum desired number of nodes.

Fiduccia-Mattheyses algorithm needs to be modified in order to deal with alignment graphs. Mainly, residues are not moved independently; since the graph topology has to be maintained, moving a residue involves moving all the residues positioned between it and a current cut point on its sequence. This modification implies further changes in the design of data structures for gain processing. Next, the sizes of parts in considered partitions cannot differ by more than the maximum cluster size in a final clustering, i.e., the number of aligned sequences. This choice implies minimal search space containing partitions consistent with all possible multiple alignment. In the initial partition sequences are cut in their midpoints.

The Fiduccia-Mattheyses heuristic may be optionally extended with a *multi-level* scheme [7]. In this approach increasingly coarse approximations of the graph are created by an iterative process called *coarsening*. At each iteration step selected pairs of nodes are merged into single nodes. Adjacent edges are merged accordingly and weighted with sums of original weights. The final coarsest graph is partitioned using Fiduccia-Mattheyses algorithm. Then the partition is projected back to the original graph through the series of *uncoarsening* operations, each of which is followed by a Fiduccia-Mattheyses based refinement. Because the last refinement is applied to the original graph, the multilevel scheme in fact reduces the problem of selecting an initial partition to the problem of selecting pairs of nodes to be merged. In alignment graphs only neighboring nodes can be merged, so MSARC just merges consecutive pairs of neighboring nodes.

2.4 Refinement

An example of alignment columns produced by residue clustering can be seen in Figure 2(ab). Unfortunately, right parts of alignments contain many superfluous spaces that could easily be removed manually.

Therefore we decided to add a refinement step, following the method used in ProbCons [1]. Sequences are split into two groups and the groups are pairwise re-aligned. Re-alignment is performed using the Needleman-Wunsch algorithm with the score for each pair of positions defined as the sum of posterior probabilities for all non-gap pairs and zero gap-penalty. Since gap-penalties are not used, every such refinement iteration creates a new alignment of equal or greater expected accuracy. First each sequence is re-aligned with the remaining sequences, since such division is very efficient in removing superfluous spaces. Next, several randomly selected sequence subsets are re-aligned against the rest.

Figures 2(cd) show the results of refining the alignments from Figures 2(ab). Refinement removed superfluous spaces from the clustering process and optimized the alignment. Note that the final post-refinement alignments turned out to be the same for both Fiduccia-Mattheyses and multilevel method of graph partitioning.

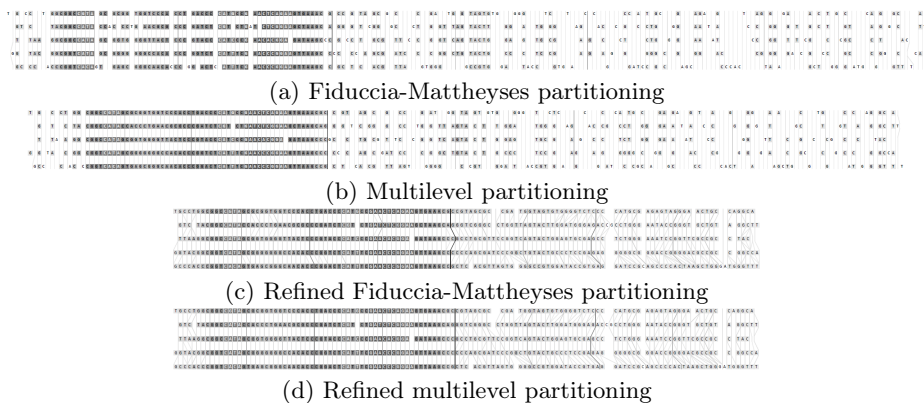


Fig. 2. Example visualization of the alignment produced by the graph partitioning methods alone (ab) and graph partitioning followed by refinement (cd). Residue colors reflect how well the column is aligned based on residue match probabilities (darker is better). Partition cuts are colored to show the order of partitioning with darker cuts being performed earlier.

3 Results

3.1 Benchmark Data and Methodology

MSARC was tested against the BALiBASE 3.0 benchmark database [21]. It contains manually refined reference alignments based on 3D structural superpositions. Each alignment contains core-regions that correspond to the most reliably alignable sections of the alignment. Alignments are divided into five sets designed to evaluate performance on varying types of problems:

RV1X Equidistant sequences with two different levels of conservation

RV11 very divergent sequences (<20% identity)

RV12 medium to divergent sequences (20-40% identity)

RV20 Families aligned with a highly divergent “orphan” sequence

RV30 Subgroups with <25% residue identity between groups

RV40 Sequences with N/C-terminal extensions

RV50 Internal insertions

BALiBASE 3.0 also provides a program comparing given alignments with a reference one. Alignments are scored according to two metrics. A sum-of-pairs score (SP) showing the ratio of residue pairs that are correctly aligned, and a total column (TC) score showing the ratio of correctly aligned columns. Both scores can be applied to full sequences or just the core-regions.

Two variants of MSARC: with multilevel Fiduccia-Mattheyses algorithm (MSARC-ML) and with basic Fiduccia-Mattheyses algorithm (MSARC-FM)

were tested on the full length sequences and scored based on the correct alignment of core-regions. The results were compared to CLUSTAL Ω [21,18] ver. 1.1.0, DIALIGN-T [20] ver. 0.2.2, DIALIGN-TX [19] ver. 1.0.2, MAFFT [8] ver. 6.903, MUSCLE [2] ver. 3.8.31, MSAProbs [10] ver. 0.9.7, Probalign [17] ver. 1.4, ProbCons [1] ver. 1.12 and T-Coffee [15] ver. 9.02.

All the programs were executed with their default parameters. In the case of MSARC, default parameters of *stochastic alignment*, *consistency transformation* and *iterative refinement* steps follow the defaults of corresponding steps of Probalign and ProbCons. Namely, MSARC was run with Gonnet 160 similarity matrix [4], gap penalties of -22 , -1 and 0 for gap open, extension and terminal gaps respectively, $\beta = 0.2$, a cut-off value for posterior probabilities of 0.01 (values smaller than the cutoff are set to 0 and operations designed for sparse matrices are used in order to speed up computations), two iterations of the consistency transformation and 100 iterations of iterative refinement.

3.2 Aligner Comparison

Table 1 shows the SP and TC scores obtained by the alignment algorithms on the BALiBASE 3.0 benchmark. MSARC-ML has slightly better accuracy than MSARC-FM. Both variants of MSARC substantially outperform DIALIGN-T (the only non-progressive method in the test) and DIALIGN-TX (a progressive extension of DIALIGN-T). Moreover, MSARC achieves accuracy similar to the leading alignment methods: MSAProbs, Probalign and ProbCons.

The differences are not significant in most cases (see Table 2) and correspond with the structure of benchmark series – MSARC shows the best results for test series RV11 and RV40, and the worst performance on RV20 and RV30. Distances in RV20 and RV30 families are particularly well represented by phylogenetic trees (low similarity between highly conserved subgroups). On the other hand, series RV11 contains highly divergent sequences for which guide-tree is poorly informative, even if it represents the correct phylogeny, and RV40 contains sequences with N/C-terminal extensions which may affect the accuracy of the estimated phylogeny.

We illustrate this observation with an example of test case BB40037. As is shown in column 9 of Table 1, MSARC outperforms other methods by a large margin. The TC scores of zero means that each alignment method has shifted at least one sequence from its correct position relative to the other sequences. Figure 3 presents the structure of the reference alignment, as well as alignments generated by MSARC, Probalign and MSAProbs. The large family of red, orange and yellow colored sequences near the bottom has been misaligned by the progressive methods. The reason for this is more visible in Figure 4, where sequences in alignments are reordered according to related guide-trees.

Probalign aligns separately the first half of the sequences (blue and green) and the second half of the sequences (from yellow to red). Next, the prefixes of the second group are aligned with the suffixes of the first group, propagating an error within a yellow sub-alignment.

Table 1. Performance on BALiBASE 3.0

Aligner	SP/TC scores								Computation Time
	all	RV11	RV12	RV20	RV30	RV40	RV50	BB40037	
MSARC-ML	<u>87.6</u> <u>57.3</u>	70.1 46.1	<u>94.5</u> <u>85.6</u>	<u>92.5</u> <u>40.7</u>	<u>83.4</u> <u>45.7</u>	93.1 63.3	<u>88.7</u> <u>51.6</u>	97.1 70.0	33 : 49 : 37
MSARC-FM	<u>87.5</u> <u>57.1</u>	<u>70.0</u> <u>46.0</u>	<u>94.5</u> <u>85.6</u>	<u>92.5</u> <u>40.9</u>	<u>82.8</u> <u>45.0</u>	<u>93.0</u> <u>62.9</u>	<u>88.6</u> <u>51.7</u>	97.1 70.0	22 : 14 : 19
CLUSTAL Ω	<u>84.0</u> <u>55.4</u>	<u>59.0</u> <u>35.8</u>	<u>90.6</u> <u>78.9</u>	<u>90.2</u> <u>45.0</u>	<u>86.2</u> <u>57.5</u>	<u>90.2</u> <u>57.9</u>	<u>86.2</u> <u>53.3</u>	<u>61.2</u> <u>0.0</u>	12 : 15
DIALIGN-T	<u>77.3</u> <u>42.8</u>	<u>49.3</u> <u>25.3</u>	<u>88.8</u> <u>72.5</u>	<u>86.3</u> <u>29.2</u>	<u>74.7</u> <u>34.9</u>	<u>82.0</u> <u>45.2</u>	<u>80.1</u> <u>44.2</u>	<u>52.6</u> <u>0.0</u>	1 : 13 : 21
DIALIGN-TX	<u>78.8</u> <u>44.3</u>	<u>51.5</u> <u>26.5</u>	<u>89.2</u> <u>75.2</u>	<u>87.9</u> <u>30.5</u>	<u>76.2</u> <u>38.5</u>	<u>83.6</u> <u>44.8</u>	<u>82.3</u> <u>46.6</u>	<u>52.8</u> <u>0.0</u>	1 : 36 : 05
MAFFT	<u>86.7</u> <u>58.4</u>	<u>65.3</u> <u>42.8</u>	<u>93.6</u> <u>83.8</u>	<u>92.5</u> <u>44.6</u>	<u>85.9</u> <u>58.1</u>	<u>91.5</u> <u>59.0</u>	<u>90.1</u> <u>59.4</u>	<u>56.4</u> <u>0.0</u>	54 : 04
MUSCLE	<u>81.9</u> <u>47.5</u>	<u>57.2</u> <u>31.8</u>	<u>91.5</u> <u>80.4</u>	<u>88.9</u> <u>35.0</u>	<u>81.4</u> <u>40.9</u>	<u>86.5</u> <u>45.0</u>	<u>83.5</u> <u>45.9</u>	<u>48.4</u> <u>0.0</u>	23 : 32
MSAProbs	87.8 60.7	<u>68.2</u> <u>44.1</u>	<u>94.6</u> 86.5	92.8 46.4	86.5 60.7	<u>92.5</u> <u>62.2</u>	90.8 60.8	<u>59.5</u> <u>0.0</u>	6 : 43 : 51
Probalign	<u>87.6</u> <u>58.9</u>	<u>69.5</u> <u>45.3</u>	94.6 <u>86.2</u>	<u>92.6</u> <u>43.9</u>	<u>85.3</u> <u>56.6</u>	<u>92.2</u> <u>60.3</u>	<u>88.7</u> <u>54.9</u>	<u>54.2</u> <u>0.0</u>	4 : 31 : 41
ProbCons	<u>86.4</u> <u>55.8</u>	<u>67.0</u> <u>41.7</u>	<u>94.1</u> <u>85.5</u>	<u>91.7</u> <u>40.6</u>	<u>84.5</u> <u>54.4</u>	<u>90.3</u> <u>53.2</u>	<u>89.4</u> <u>57.3</u>	<u>59.3</u> <u>0.0</u>	6 : 56 : 32
T-Coffee	<u>85.7</u> <u>55.1</u>	<u>65.5</u> <u>40.9</u>	<u>93.9</u> <u>84.8</u>	<u>91.4</u> <u>40.1</u>	<u>83.7</u> <u>49.0</u>	<u>89.2</u> <u>54.5</u>	<u>89.4</u> <u>58.5</u>	<u>50.9</u> <u>0.0</u>	13 : 53 : 02

Columns 2-9 show the mean SP and TC scores for each alignment algorithm on the whole BALiBASE dataset, each of its series and case BB40037. The last column presents total CPU computation time (hh:mm:ss). All scores are multiplied by 100. Best results in each column are shown in bold.

MSAprobs aligns separately the dark blue, light blue and red sequences. Next the blue sub-alignments are aligned together. Resulting alignment has erroneously inserted gaps near the right ends of dark blue sequences. This error is propagated in next step, where the suffix of the blue alignment is aligned with

Table 2. Significance of differences in BALiBASE 3.0 SP/TC scores

SP scores	RV11	RV12	RV20	RV30	RV40	RV50	Total
Clustal Ω	+3.8e-7	+1.1e-5	+0.0031	-0.047	+4.2e-6	+0.012	+8.7e-15
DIALIGN-T	+8.6e-8	+7.7e-9	+1.3e-7	+2.7e-6	+2.1e-9	+0.00098	+5.3e-36
DIALIGN-TX	+1.0e-7	+6.2e-8	+2.3e-7	+8.7e-6	+2.8e-9	+0.0017	+3.1e-34
MAFFT	+0.0031	+0.00085	-(0.64)	-0.0009	+0.0005	-(0.072)	+0.028
MUSCLE	+4.5e-6	+1.3e-6	+0.0002	+(0.24)	+2.5e-8	+0.006	+6.8e-22
MSAProbs	+0.015	-(0.56)	-0.016	-1.9e-5	+(0.39)	-0.0041	-0.0025
Probalign	+(0.16)	-(0.77)	-0.048	-0.0099	+(0.66)	-(0.85)	-(0.067)
ProbCons	+0.0070	+0.037	+0.032	-(0.11)	+0.0014	-(0.17)	+0.0018
T-Coffee	+0.001	+0.005	+0.021	-(0.40)	+0.0001	-(0.077)	+7.1e-6
TC scores	RV11	RV12	RV20	RV30	RV40	RV50	Total
Clustal Ω	+2.8e-5	+0.0004	-0.025	-0.0018	+(0.11)	-(0.84)	+(0.096)
DIALIGN-T	+1.5e-6	+2.2e-8	+9.6e-5	+0.0024	+4.9e-8	+0.027	+3.6e-26
DIALIGN-TX	+1.3e-6	+4.0e-7	+0.00040	+0.038	+1.3e-7	+(0.066)	+9.5e-23
MAFFT	+(0.11)	+0.005	-(0.052)	-0.0007	+(0.07)	-(0.062)	-(0.55)
MUSCLE	+9.9e-5	+0.0002	+(0.06)	+(0.76)	+2.2e-6	+0.009	+5.8e-13
MSAProbs	+(0.13)	-(0.22)	-0.0016	-8.5e-5	+(0.076)	-0.0014	-5.4e-7
Probalign	+(0.54)	-(0.11)	-0.00062	-0.0006	+(0.087)	-(0.36)	-1.9e-6
ProbCons	+0.043	-(0.69)	-(0.31)	-0.011	+0.017	-(0.062)	+(0.84)
T-Coffee	+0.003	+(0.10)	+(0.75)	-(0.11)	+(0.12)	-0.0072	+(0.61)

Entries show p -values indicating the significance of the mean difference of SP/TC scores between MSARC-ML and other aligners as measured using the Wilcoxon matched-pair signed-rank test. A + means that MSARC had a higher mean score while a - means MSARC had a lower mean score. Nonsignificant p -values (>0.05) are shown in parentheses.

the prefix of the red alignment. Finally the single violet sequence is added to the alignment, splitting it in two.

For both programs, alignment errors introduced in the earlier steps are propagated to the final alignment. On the other hand, the non-progressive strategy used in MSARC yields a reasonable approximation of the reference alignment (see Figure 3(ab)).

4 Discussion

The progressive principle dominates multiple alignment algorithms for nearly 20 years. Throughout this time, many groups have dedicated their effort to refine its accuracy to the current state. Other approaches were omitted due to high computational complexity and/or unsatisfactory quality. To our best knowledge, MSARC is the only non-progressive aligner of quality comparable to best progressive programs. Moreover, due to a guide-tree bias of alignments computed with progressive methods, MSARC is a quality leader for sequence sets with evolutionary distances hardly representable by a phylogenetic tree.

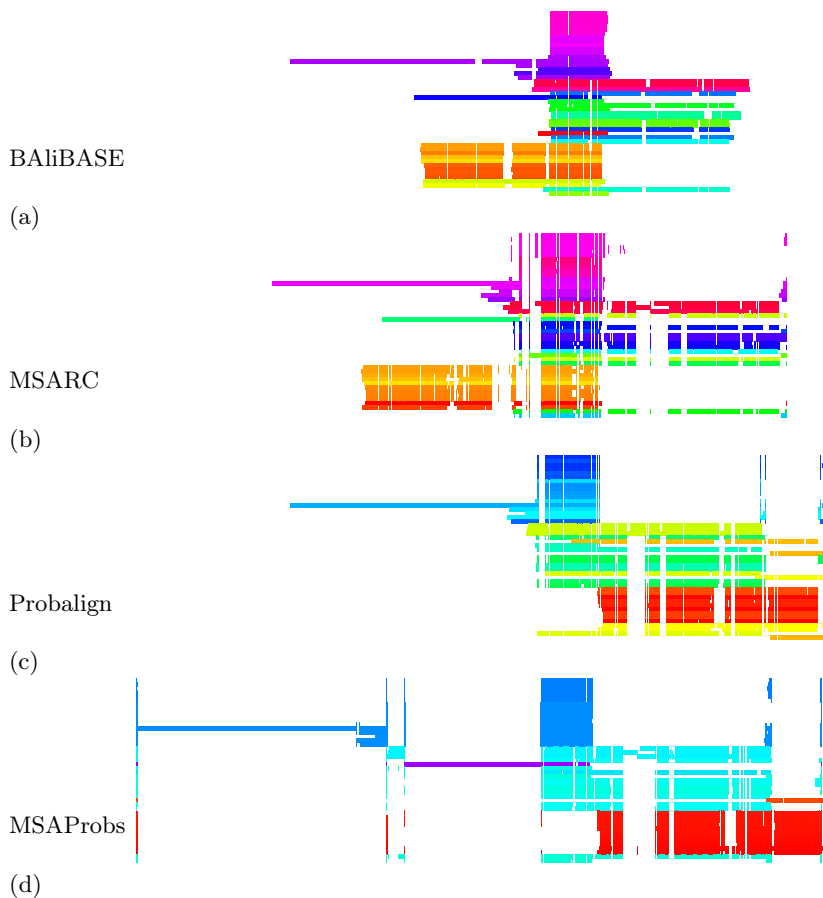


Fig. 3. Visualization of reference (a) and reconstructed (bcd) alignments for test case BB40037. In all alignments sequences are ordered accordingly. Each sequence is colored based on the evolutionary distance to its neighbors in a phylogenetic tree, such that families of related sequences have similar colors. Trees for (a) and (b) are computed with the PhyML 3.0 program [6], using the maximum parsimony method. Trees for (c) and (d) are the guide-trees used by those aligners.

Despite of the algorithmic novelty, the non-progressive approach to multiple alignment makes MSARC an interesting tool for phylogeny reconstruction pipelines. The objective of these procedures is to infer the structure of a phylogenetic tree from a given sequence set. Multiple alignment is usually the first pipeline step. When alignment is guided by a tree, the reconstructed phylogeny is biased towards this tree. In order to minimize this effect, some phylogenetic pipelines alternately optimize a tree and an alignment [16,12,10]. Unbiased alignment process of MSARC may simplify this procedure and improve the reconstruction accuracy, especially in most problematic cases.

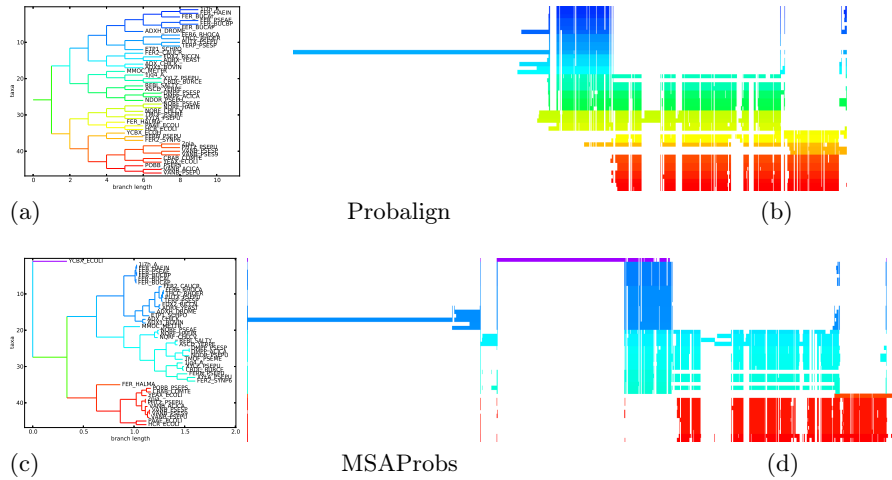


Fig. 4. Guide trees (ac) and alignment visualizations (bd) for test case BB40037 and programs Probalign (ab) and MSARCs (cd). Tree branches and aligned sequences are colored based on the evolutionary distances to their neighbors, as computed from the guide-trees used during alignment. Sequences in alignments are ordered following their order in trees, so related sequences have similar color and are positioned together.

The main disadvantage of MSARC is its computational complexity, especially in the case of the multilevel scheme variant (MSARC-FM is $\sim 3\times$ slower than MSARCs and $\sim 5\times$ slower than Probalign, MSARC-ML is $1.5\times$ slower than MSARC-FM). However, the running time can be greatly improved by using multiple cores to parallel computations, because every step of its algorithm can be parallelized. Since multiple cores are becoming more and more common, this should allow for the computation time comparable with other alignment algorithms.

MSARC has also the potential for quality improvements. Alternative methods of computing residue alignment affinities could be used to improve the accuracy of both MSARC and Probalign based methods. Other approaches to alignment graph partitioning may also lead to improvements in the accuracy of MSARC, for example a better method of pairing residues for multilevel coarsening than currently used naive consecutive neighbors merging.

Acknowledgements. This work was supported by the Polish Ministry of Science and Higher Education [N N519 652740].

References

1. Do, C.B., Mahabhashyam, M.S.P., Brudno, M., Batzoglou, S.: Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome. Res.* 15(2), 330–340 (2005), <http://dx.doi.org/10.1101/gr.2821705>

2. Edgar, R.C.: Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* 32(5), 1792–1797 (2004), <http://dx.doi.org/10.1093/nar/gkh340>
3. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: *Proceedings of the 19th Design Automation Conference, DAC 1982*, pp. 175–181. IEEE Press, Piscataway (1982), <http://dl.acm.org/citation.cfm?id=800263.809204>
4. Gonnet, G.H., Cohen, M.A., Benner, S.A.: Exhaustive matching of the entire protein sequence database. *Science* 256(5062), 1443–1445 (1992)
5. Gotoh, O.: An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162(3), 705–708 (1982)
6. Guindon, S., Dufayard, J.F., Lefort, V., Anisimova, M., Hordijk, W., Gascuel, O.: New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phylml 3.0. *Syst. Biol.* 59(3), 307–321 (2010), <http://dx.doi.org/10.1093/sysbio/syq010>
7. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM). Supercomputing 1995*. ACM, New York (1995), <http://doi.acm.org/10.1145/224170.224228>
8. Katoh, K., Kuma, K.-I., Toh, H., Miyata, T.: Mafft version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res.* 33(2), 511–518 (2005), <http://dx.doi.org/10.1093/nar/gki198>
9. Kececioğlu, J.: The maximum weight trace problem in multiple sequence alignment. In: Apostolico, A., Crochemore, M., Galil, Z., Manber, U. (eds.) *CPM 1993*. LNCS, vol. 684, pp. 106–119. Springer, Heidelberg (1993)
10. Liu, K., Raghavan, S., Nelesen, S., Linder, C.R., Warnow, T.: Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science* 324(5934), 1561–1564 (2009), <http://dx.doi.org/10.1126/science.1171243>
11. Löytynoja, A., Goldman, N.: Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science* 320(5883), 1632–1635 (2008), <http://dx.doi.org/10.1126/science.1158395>
12. Lunter, G., Miklós, I., Drummond, A., Jensen, J.L., Hein, J.: Bayesian coestimation of phylogeny and sequence alignment. *BMC Bioinformatics* 6, 83 (2005), <http://dx.doi.org/10.1186/1471-2105-6-83>
13. Miyazawa, S.: A reliable sequence alignment method based on probabilities of residue correspondences. *Protein Eng.* 8(10), 999–1009 (1995)
14. Mückstein, U., Hofacker, I.L., Stadler, P.F.: Stochastic pairwise alignments. *Bioinformatics* 18(suppl. 2), S153–S160 (2002)
15. Notredame, C., Higgins, D.G., Heringa, J.: T-coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.* 302(1), 205–217 (2000), <http://dx.doi.org/10.1006/jmbi.2000.4042>
16. Redelings, B.D., Suchard, M.A.: Joint bayesian estimation of alignment and phylogeny. *Syst. Biol.* 54(3), 401–418 (2005), <http://dx.doi.org/10.1080/10635150590947041>
17. Roshan, U., Livesay, D.R.: Probalgn: multiple sequence alignment using partition function posterior probabilities. *Bioinformatics* 22(22), 2715–2721 (2006), <http://dx.doi.org/10.1093/bioinformatics/btl472>

18. Sievers, F., Wilm, A., Dineen, D., Gibson, T.J., Karplus, K., Li, W., Lopez, R., McWilliam, H., Remmert, M., Söding, J., Thompson, J.D., Higgins, D.G.: Fast, scalable generation of high-quality protein multiple sequence alignments using clustal omega. *Mol. Syst. Biol.* 7, 539 (2011), <http://dx.doi.org/10.1038/msb.2011.75>
19. Subramanian, A.R., Kaufmann, M., Morgenstern, B.: Dialign-tx: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms Mol. Biol.* 3, 6 (2008), <http://dx.doi.org/10.1186/1748-7188-3-6>
20. Subramanian, A.R., Weyer-Menkhoff, J., Kaufmann, M., Morgenstern, B.: Dialign-t: an improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics* 6, 66 (2005), <http://dx.doi.org/10.1186/1471-2105-6-66>
21. Thompson, J.D., Higgins, D.G., Gibson, T.J.: Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22(22), 4673–4680 (1994)
22. Thompson, J.D., Koehl, P., Ripp, R., Poch, O.: Balibase 3.0: latest developments of the multiple sequence alignment benchmark. *Proteins* 61(1), 127–136 (2005), <http://dx.doi.org/10.1002/prot.20527>
23. Wong, K.M., Suchard, M.A., Huelsenbeck, J.P.: Alignment uncertainty and genomic analysis. *Science* 319(5862), 473–476 (2008), <http://dx.doi.org/10.1126/science.1151532>
24. Yu, Y.K., Hwa, T.: Statistical significance of probabilistic sequence alignment and related local hidden markov models. *J. Comput. Biol.* 8(3), 249–282 (2001), <http://dx.doi.org/10.1089/10665270152530845>

Mutual Enrichment in Ranked Lists and the Statistical Assessment of Position Weight Matrix Motifs

Limor Leibovich¹ and Zohar Yakhini^{1,2}

¹ Department of Computer Science, Technion – Israel Institute of Technology,
Technion City, Haifa 32000, Israel

`llimor@cs.technion.ac.il`

² Agilent Laboratories Israel, 94 Em Hamoshavot Road, 49527 Petach-Tikva, Israel
`zohar_yakhini@agilent.com`

Abstract. Statistics in ranked lists is important in analyzing molecular biology measurement data, such as ChIP-seq, which yields ranked lists of genomic sequences. State of the art methods study fixed motifs in ranked lists. More flexible models such as position weight matrix (PWM) motifs are not addressed in this context. To assess the enrichment of a PWM motif in a ranked list we use a PWM induced second ranking on the same set of elements. Possible orders of one ranked list relative to the other are modeled by permutations. Due to sample space complexity, it is difficult to characterize tail distributions in the group of permutations. In this paper we develop tight upper bounds on tail distributions of the size of the intersection of the top of two uniformly and independently drawn permutations and demonstrate advantages of this approach using our software implementation, mmHG-Finder, to study PWMs in several datasets.

1 Introduction

Modern data analysis often faces the task of extracting characteristic features from sets of elements singled out according to some measurement. In molecular biology, for example, an experiment may lead to measurement results pertaining to genes and then questions are asked about the properties of genes for which these were high or low. This is an example, of course, and the set of elements does not have to be genes. They can be genomic regions, proteins, structures, etc. A central technique for addressing the analysis of characteristic properties of sets of elements is statistical enrichment. More specifically – the experiment results are often representable as ranked lists of elements and we then seek enrichment of other properties of these elements at the top or bottom of the ranked list. GSEA [29], for example, is a tool that addresses characteristic features of genes that are found to be differentially expressed in a comparative transcriptomics study. GOrilla [6] addresses GO terms enriched in ranked lists of genes where the ranking can be, for example, the result of processing differential expression

data or of correlations computed between genomic DNA copy number and expression [19],[2],[5]. FATIGO [21] is also a tool that is useful in the context of analyzing GO terms in ranked lists of genes. DRIMust [15], [16] searches for sequence motifs that are enriched, in a statistically significant manner, in the top of a ranked list of sequences, such as one produced by techniques like CHIP-seq.

All the aforementioned tools utilize a statistical approach that is based on assessing enrichment of an input set in an input ranked list by assessing the enrichment obtained at various cutoffs applied to the ranked list. It is often the case, however, that two quantitative properties need to be compared to each other. For example, when the elements are genes, we may have measured differential expression values, as well as measured CHIP-seq signals. We are therefore interested in assessing mutual enrichment in two ranked lists. Another example consists of one ranking according to differential expression and one according to prediction scores for miRNA targets. miTEA [25] addresses this latter case by statistically assessing the enrichment of miRNA targets in a ranked list of genes (also see [8]). To address mutual enrichment in two ranked lists over the same set of N elements, miTEA [25] performs analysis on permutations. Mutual enrichment in the top of two ranked lists can be simplified, from a mathematical point of view, by arbitrarily setting the indices of one list to the identity permutation $(1, 2, \dots, N)$ and treating the other list as a permutation π over these numbers. For the purpose of assessing the intersection of the top of the two ranked lists in a data driven manner, miTEA asks which prefix $[1, \dots, n_1]$ is enriched in the first n_2 elements of that permutation, $\pi = \pi(1), \dots, \pi(N)$. The statistics introduced by miTEA is called mmHG (min-min-Hyper-Geometric). A variant of mmHG is explained in detail in Section 2 of the current manuscript.

Statistics in the group of permutations S_N is often difficult because the number of entities to be considered by any null model is $N!$. Direct exhaustive calculation of tail distributions over S_N is therefore impractical for all but very small values of N . This difficulty is addressed by several heuristic techniques. Mapping into continuous spaces, such as in [18], has proven useful in certain cases but not for studying large deviations. In the case of enrichment statistics that focuses on the top of the permutation and seeks to assess extreme events, such as mmHG, we prefer to use bounds on tail probabilities. Tail probabilities are useful constructs when applied to analyzing molecular biology measurement data as they enable statistical assessment of observed results.

In this work we derive a tight bound on the tail probabilities of the mutual enrichment at the top of two random permutations uniformly drawn over S_N and demonstrate the utility of this approach in the context of flexible motif discovery. Our bounds are computable in polynomial time and potentially add to the accuracy of reported position weight matrix (PWM) motifs for nucleic acid sequences.

2 Background and Definitions

2.1 Mutual Enrichment in Ranked Lists – The mmHG Statistics

The mmHG statistics [25] is a generalization of the mHG statistics [6],[7],[26], [28]. While the mHG statistics quantifies the enrichment level of a set of elements at the top of a ranked list of elements of the same type, the mmHG statistics quantifies the level of mutual enrichment in two ranked lists over the same set of elements. While any parametric or non-parametric correlation statistics (e.g. Spearman’s correlation coefficient), that takes the same input, calculates the overall agreement between the two ranked lists, the mmHG statistic focuses only on agreement at the top of the two ranked lists. mmHG counts elements common to the top of both lists, without predefining what top is. Its intended output is the probability for observing an intersection at least as large in two randomly ranked lists (the enrichment mmHG *P-value*). In this section we describe the mmHG statistics and in later sections we suggest a tight bound for the p-value. Our definition of the mmHG statistic varies slightly from that of Steinfeld *et al.* [25].

Mutual enrichment in the top of two ranked lists can be simplified, from a mathematical point of view, by arbitrarily setting the indices of one list to the identity permutation $(1, 2, \dots, N)$ and treating the other list as a permutation. Details of this transform are given in Section 2.3. We now define mmHG for the simple case of one permutation. Consider a permutation $\pi = \pi(1), \dots, \pi(N) \in S_N$ - the group of all permutations over the numbers $1, \dots, N$. mmHG is a function that takes π and calculates two numbers $1 \leq n_1, n_2 \leq N$ such that the observed intersection between the numbers $1, \dots, n_1$ and the first n_2 elements of $\pi - \pi(1), \dots, \pi(n_2)$ - is the most surprising in terms of the hypergeometric p-value. Additionally, mmHG further calculates this aforementioned p-value.

Formally, given $\pi \in S_N$ and for every $1 \leq n_1, n_2 \leq N$, let $b_\pi(n_1, n_2)$ be the size of the intersection of $1, \dots, n_1$ with $\pi(1), \dots, \pi(n_2)$. Set

$$mmHG\ score(\pi) = \min_{1 \leq n_1 \leq N} \min_{1 \leq n_2 \leq N} HGT(N, n_1, n_2, b_\pi(n_1, n_2))$$

where HGT is the tail distribution of an hypergeometric random variable:

$$HGT(N, n_1, n_2, b) = \sum_{i=b}^{\min(n_1, n_2)} \frac{\binom{n_1}{i} \binom{N-n_1}{n_2-i}}{\binom{N}{n_2}}$$

The mmHG score cannot be considered as a significance measure, due to the multiple testing involved in finding n_1 and n_2 . A simple way to correct an mmHG score s for multiple testing and report a p-value bound would be to use the Bonferroni correction. That is done by multiplying s by the number of multiple tests conducted which is N^2 . Therefore:

$$mmHG\ p - value(s, N) \leq s \cdot N^2$$

In Section 3 we present significantly tighter bounds.

2.2 PWM Motifs

Data produced by techniques such as ChIP-seq [14], ChIP-exo [20], CLIP [13], PAR-CLIP [9] and others are readily representable as ranked lists of sequences, where the ranking is according to measured binding affinity. Computational tools and approaches to motif discovery form part of the data analysis workflow that is used to extract knowledge and understanding from this type of studies. We are often interested in sequence motifs that are observed to be enriched in sequences where strong binding affinity is measured. A position weight matrix (PWM) is a commonly used representation of motifs in biological sequences [24],[27],[11]. This representation is more faithful to biology than representation by exact words. A PWM is a matrix of score values that gives a weighted match to any given substring of fixed length. It has one row for each symbol of the alphabet, and one column for each position in the pattern. The score assigned by a PWM to a substring $S = S_1 \dots S_K$ is defined as $\sum_{j=1}^K m_{s_j, j}$, where j represents position in the substring, S_j is the symbol at position j in the substring, and $m_{\alpha, j}$ is the score in row α , column j of the matrix. In other words, a PWM score is the sum of position-specific scores for each symbol in the substring. This definition can be generalized to yield a score for a sequence $S = S_1 \dots S_M$ longer than the PWM by calculating $\max_{1 \leq i \leq M-K+1} \sum_{j=1}^K m_{s_{i+j-1}, j}$. Alternatively, an enhanced model that takes into account multiple occurrences of the PWM in the sequence can be applied by summing over sufficiently strong occurrences of the PWM or by other more sophisticated approaches [22].

2.3 mmHG Statistics for PWM Motifs

Given a set of sequences that were tested in a high throughput experiment such as ChIP-seq [14], CLIP [13] and others, they can be ranked according to the measured binding affinities, yielding a ranked list L_1 . Since usually we are interested in finding motifs amongst sequences having strong binding affinities, we actually search for motifs that are more prevalent at the top of this list. It is clear that any algorithm for de-novo flexible motif search would need to evaluate candidate PWMs. Given a PWM which we want to assess, the sequences can also be ranked according to their PWM scores, yielding another ranked list L_2 , different from L_1 . A significant PWM motif would yield significant scores for sequences having strong binding affinities. Therefore, the question of PWM motif discovery from ranked experimental data can be formulated as quantifying the mutual enrichment level for the two ranked lists L_1 and L_2 . Given two ranked lists L_1 and L_2 over the universe of N sequences, they can be transformed into two respective permutations, $\pi_1 = (\pi_1(1), \dots, \pi_1(N))$ and $\pi_2 = (\pi_2(1), \dots, \pi_2(N))$. The relative permutation π , of π_2 w.r.t. π_1 , is defined by $\pi(\pi_1(j)) = \pi_2(j)$, for every $j = 1, \dots, N$ or simply, using the operations in the group S_N : $\pi = \pi_2 \cdot \pi_1^{-1}$. Using the relative permutation π , we can represent the mutual enrichment of the top parts of L_1 and L_2 as *mmHG score*(π), defined above.

3 Algorithms and Results

3.1 Estimation of the mmHG p-Value – Introducing First Upper Bound

Given an mmHG score s , observed in analyzing real measurement data, we would like to assess the statistical significance of this observation. Assuming endless computational power, we would enumerate all permutations and calculate the mmHG score for each, in order to characterize the distribution of mmHG as a random variable over S_N . The p-value for s is then simply:

$$mmHG\ p - value(s, N) = \frac{\text{The number of permutations having mmHG score } \leq s}{N!}$$

Since the number of permutations is huge, the process described above is very far from feasible. Therefore, we seek a computationally tractable upper bound, preferably tight.

A trivial upper bound is the Bonferroni corrected mmHG score defined by $s \cdot N^2$. A more subtle upper bound was suggested by Steinfeld *et al.* [25] and is briefly described in Section 3.3. In this work we introduce a tighter bound that is polynomially computable.

We will next describe an intuitive upper bound and later refine it to produce a tighter bound. Our input comprises an mmHG score s , and the total number of elements N . The output will be an upper bound for the p-value. The efficiency of our approach relies on enumerating all possible HGT scores rather than enumerating all permutations in S_N . This approach is computationally efficient as HGT is a function of four input parameters: N , n_1 , n_2 , and b . Given N , there are $O(N^3)$ possible combinations of n_1 , n_2 and b . Next, all is left to do is to determine how many permutations stand behind each HGT score. To this end, we will define the function $A(N, n_1, n_2, b)$ to be the number of permutations for which it holds that out of the first n_2 entries, b of them are taken from the range $[1, \dots, n_1]$. This formulation is equivalent to counting permutations for which we attain, at some point, the value $HGT(N, n_1, n_2, b)$, had we taken the exhaustive approach. $A(N, n_1, n_2, b)$ can be represented as:

$$A(N, n_1, n_2, b) = \binom{n_1}{b} \binom{n_2}{b} b! \binom{N - n_1}{n_2 - b} (n_2 - b)! (N - n_2)!$$

as we first choose b elements from the range $[1, \dots, n_1]$ to appear at the first n_2 entries of the permutation (there are $\binom{n_1}{b}$ possibilities). Then, we choose where to position these b elements at the first n_2 entries of the permutation and consider all internal arrangements (for each choice of b elements there are $\binom{n_2}{b} b!$ possibilities). We next choose $n_2 - b$ elements from the range $[n_1 + 1, \dots, N]$ to appear at the rest of the first n_2 entries of the permutation (there are $\binom{N - n_1}{n_2 - b}$ possibilities for that) and consider all possible $(n_2 - b)!$ arrangements. Finally, we take into account all possible $(N - n_2)!$ arrangements of the rest $N - n_2$ entries of the permutation.

A straightforward upper bound for the number of permutations in S_N having mmHG score better than s follows:

$$|\{\pi' \in S_N : mmHG(\pi') \leq s\}| \leq \sum_{\substack{n_1, n_2, b: \\ HGT(N, n_1, n_2, b) \leq s}} \Lambda(N, n_1, n_2, b)$$

From which an upper bound is easily derived:

$$mmHG \text{ p-value}(s, N) \leq \frac{\sum_{\substack{n_1, n_2, b: \\ HGT(N, n_1, n_2, b) \leq s}} \Lambda(N, n_1, n_2, b)}{N!}$$

By algebraic manipulations we get:

$$mmHG \text{ p-value}(s, N) \leq \sum_{\substack{n_1, n_2, b: \\ HGT(N, n_1, n_2, b) \leq s}} \frac{\binom{n_1}{b} \binom{N-n_1}{n_2-b}}{\binom{N}{n_2}}$$

This upper bound is simple and requires $O(N^3)$ HGT calculations. An HGT calculation takes $O(N)$ time, assuming binomial coefficients can be calculated in $O(1)$ time, for example by using Stirling’s approximation [1]:

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \frac{1}{e^{1/12n+1}} \leq n! \leq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \frac{1}{e^{1/12n}}$$

, which is tight for large factorials.

3.2 A Refined Upper Bound for the p-Value

The upper bound introduced in the previous section counts the number of permutations for which the value $HGT(N, n_1, n_2, b)$ is calculated when taking the non-practical exhaustive approach that enumerates over all $N!$ permutations. Ideally, we wish to count the number of permutations for which the value $HGT(N, n_1, n_2, b)$ is also their mmHG score, as a permutation may have several HGT values that are better than s , so it can be counted more than once. This explains why the formula introduced earlier is an upper bound and not an exact p-value. A second observation that follows is that the smaller the mmHG score s is, the tighter the bound, because a permutation will have fewer combinations (N, n_1, n_2, b) having HGT score better than s .

Therefore, if we can reduce the extent of multiple counting of the same permutation, we will get a tighter bound. We do this by looking one step backwards. If, for example, $HGT(N, n_1, n_2, b) \leq s$, we can exclude from the counting permutations that contain b elements from the range $[1, \dots, n_1 - 1]$ at their first n_2 entries because they are already taken into account in $\Lambda(N, n_1 - 1, n_2, b)$ (because necessarily $HGT(N, n_1 - 1, n_2, b) \leq s$, as we will later explain).

Let $\Psi(N, n_1, n_2, b)$ be the set of permutations for which it holds that out of the first n_2 entries, b of them are taken from the range $[1, \dots, n_1]$ (note that $\Lambda(N, n_1, n_2, b)$ introduced earlier is, therefore, the size of $\Psi(N, n_1, n_2, b)$). Assuming $HGT(N, n_1, n_2, b) \leq s$, we can partition the set $\Psi(N, n_1, n_2, b)$ into five disjoint subsets ψ_1, \dots, ψ_5 such that $\psi = \psi_1 \cup \psi_2 \cup \psi_3 \cup \psi_4 \cup \psi_5$, as follows:

$$\begin{aligned} \psi_1 &= \Psi(N, n_1, n_2, b) \cap \Psi(N, n_1 - 1, n_2 - 1, b - 1) \cap \Psi(N, n_1 - 1, n_2, b) \\ \psi_2 &= \Psi(N, n_1, n_2, b) \cap \Psi(N, n_1 - 1, n_2 - 1, b - 1) \cap \Psi(N, n_1, n_2 - 1, b) \\ \psi_3 &= \Psi(N, n_1, n_2, b) \cap \Psi(N, n_1 - 1, n_2 - 1, b - 1) \cap \Psi(N, n_1 - 1, n_2, b - 1) \\ &\quad \cap \Psi(N, n_1, n_2 - 1, b - 1) \\ \psi_4 &= \Psi(N, n_1, n_2, b) \cap \Psi(N, n_1 - 1, n_2 - 1, b) \\ \psi_5 &= \Psi(N, n_1, n_2, b) \cap \Psi(N, n_1 - 1, n_2 - 1, b - 2) \cap \Psi(N, n_1 - 1, n_2, b - 1) \\ &\quad \cap \Psi(N, n_1, n_2 - 1, b - 1) \end{aligned}$$

The properties of the hypergeometric distribution imply that ψ_1, ψ_2, ψ_4 can be disregarded, in the current counting stage. To explain why, we will demonstrate the argument on ψ_1 . The permutations in ψ_1 contain b elements from the range $[1, \dots, n_1 - 1]$ at the first n_2 entries. We also assume that $\text{HGT}(N, n_1, n_2, b) \leq s$. Therefore $\text{HGT}(N, n_1 - 1, n_2, b) \leq s$ also holds, as the same intersection is observed for even a smaller set. Thus, the permutations in ψ_1 should have been counted when handling the triplet $n_1 - 1, n_2$ and b and disregarded for the combination n_1, n_2 and b . Similar arguments hold for ψ_2 and ψ_4 .

ψ_3 should be counted if it holds that $\text{HGT}(N, n_1 - 1, n_2 - 1, b - 1) > s$ and $\text{HGT}(N, n_1 - 1, n_2, b - 1) > s$ and $\text{HGT}(N, n_1, n_2 - 1, b - 1) > s$, otherwise ψ_3 would have been counted by former triplets. Similarly, ψ_5 should be counted if $\text{HGT}(N, n_1 - 1, n_2 - 1, b - 2) > s$ and $\text{HGT}(N, n_1 - 1, n_2, b - 1) > s$ and $\text{HGT}(N, n_1, n_2 - 1, b - 1) > s$. Finally, we calculate the sizes of ψ_3 and ψ_5 , in the relevant cases. The permutations in ψ_3 contain $b-1$ elements taken from the range $[1, \dots, n_1 - 1]$ located at the first $n_2 - 1$ entries, where the number n_1 is positioned at entry n_2 . Therefore:

$$|\psi_3| = \binom{n_1 - 1}{b - 1} \binom{n_2 - 1}{b - 1} (b - 1)! \binom{N - n_1}{n_2 - b} (n_2 - b)! (N - n_2)!$$

The permutations in ψ_5 contain $b-2$ elements taken from $[1, \dots, n_1 - 1]$ located at the first $n_2 - 1$, where n_1 is positioned at one of the first $n_2 - 1$ entries, and also entry n_2 contains an element from $[1, \dots, n_1 - 1]$. Therefore:

$$|\psi_5| = \binom{n_1 - 1}{b - 2} \binom{n_2 - 1}{b - 2} (b - 2)! (n_2 - b + 1) \binom{N - n_1}{n_2 - b} (n_2 - b)! (n_1 - b + 1) (N - n_2)!$$

From the above we next conclude an upper bound. Denote

$$I(\text{HGT}(N, n_1, n_2, b) > s) = \begin{cases} 1, & \text{if } \text{HGT}(N, n_1, n_2, b) > s \\ 0, & \text{otherwise} \end{cases} .$$

$$\begin{aligned}
 \Lambda^*(N, n_1, n_2, b) = & \\
 & |\psi_3| \times I(\text{HGT}(N, n_1 - 1, n_2 - 1, b - 1) > s) \\
 & \quad \times I(\text{HGT}(N, n_1 - 1, n_2, b - 1) > s) \\
 & \quad \times I(\text{HGT}(N, n_1, n_2 - 1, b - 1) > s) \\
 & + \\
 & |\psi_5| \times I(\text{HGT}(N, n_1 - 1, n_2 - 1, b - 2) > s) \\
 & \quad \times I(\text{HGT}(N, n_1 - 1, n_2, b - 1) > s) \\
 & \quad \times I(\text{HGT}(N, n_1, n_2 - 1, b - 1) > s)
 \end{aligned}$$

Yielding the following upper bound for the p-value:

$$\text{mmHG } p\text{-value}(s, N) \leq \frac{\sum_{\substack{n_1, n_2, b: \\ \text{HGT}(N, n_1, n_2, b) \leq s}} \Lambda^*(N, n_1, n_2, b)}{N!}$$

Note that when n_1 or $n_2 \leq 1$, $\Lambda^*(N, n_1, n_2, b)$ is defined as $\Lambda(N, n_1, n_2, b)$. Also, given N, n_1 and n_2, b can be any integer in $[\max(0, n_2 - N + n_1), \min(n_1, n_2)]$.

This upper bound uses more delicate counting than the bound introduced in the previous section. In the following sections we assess the tightness of this bound. In later sections we demonstrate an application for PWM motif search.

3.3 Comparison to a Different Variant

We note that the bound described in Steinfeld *et al.* [25] addresses a slightly different variant of mmHG as a random variable over S_N . The definition with which we work here is more amenable to deriving tight bounds as described above. Given a single permutation $\pi \in S_N$ and for every $i = 1, \dots, N$, a binary vector λ_i is defined in which exactly i entries are 1 and $N-i$ entries are 0, as follows: $\lambda_i(j) = 1$ if $\pi(j) \leq i$. The mmHG score of a permutation π is then defined by Steinfeld *et al.* [25] as:

$$\text{mmHG}(\pi) = \min_{1 \leq i \leq N} P\text{-value}(\text{mHG}(\lambda_i)),$$

where $\text{mHG}(\lambda) = \min_{1 \leq i \leq N} \text{HGT}(N, B, n, b_n), N = |\lambda|, b_n = \sum_{i=1}^n \lambda_i$ and $B = b_N$. A possible upper bound is then given by:

$$(*) \quad P\text{-value}(\text{mmHG}(\pi)) \leq \min_{1 \leq i \leq N} \text{mHG}(\lambda_i) \cdot i \cdot N$$

Computing the latter quantity requires $O(N^2)$ HGT calculations and is therefore more computationally efficient than the two bounds described in Sections 3.1 and 3.2 of this current work, that require $O(N^3)$ HGT calculations. We observed that our bound was tighter than the bound in (*), as later shown in Figure 1D. For example, for a permutation having mmHG score = $7.8 \cdot 10^{-25}$ ($N = 100$), our

bound was $3.5 \cdot 10^{-23}$ while (*) yielded $4.2 \cdot 10^{-21}$. For one permutation with mmHG score = $5.1 \cdot 10^{-5}$ ($N = 100$), our bound was 0.026 while (*) yielded 0.2. The latter example demonstrates that a tighter bound is important for classifying an observation as statistically significant (assuming a significance threshold of 0.05).

3.4 Assessment of Tightness

In order to assess the quality of our bound, we compared it to the exact p-value, which can be calculated for small values of N (that is, in cases where $N!$ is not too large). Figure 1A compares the mmHG score (which also serves as a lower bound for the p-value), the exact p-value (calculated by exhaustive enumeration of all $10!$ permutations), our upper bound and the Bonferroni corrected p-value for $N=10$. Figure 1B shows the same for $N=20$, except that exact p-values cannot be calculated exhaustively, and therefore an empirical p-value is produced by randomly sampling 10^7 permutations. In both cases our upper bound is significantly tighter than the Bonferroni bound. We also observed that the smaller the mmHG scores are – the tighter is our bound, consistent with lesser over-counting for smaller scores, as explained in previous sections. Comparison between the first bound described in Section 3.1 and the bound described in Section 3.2 is shown in Figure 1C (for $N=20$). We observed that enumerating all HGT scores rather than enumerating all permutations in S_N significantly improves the p-value estimation. Moreover, the refinement of this approach produced by reducing the extent of multiple counting of permutation further improves the upper bound. In Figure 1D the bounds, including the bound introduced in Section 3.3 (Steinfeld bound), are shown for $N=100$. An empirical p-value was not calculated here as even if we sample 10^7 permutations, a p-value smaller than 10^{-7} cannot be obtained. The bound suggested in this paper was almost always observed to be tighter than the bound introduced in Section 3.3.

3.5 Application in PWM Motif Search

In this section we discuss mmHG as a framework for assessing the significance of PWM motifs in ranked lists. Given a ranked list of sequences and a PWM motif, by using the mmHG statistics and the bounds introduced earlier, we can assign a p-value to represent the significance of that PWM being enriched at the top of the list. To apply this approach for de-novo motif search, one needs to theoretically consider all possible PWMs. This is not feasible and as a heuristic approach we wrote mmHG-Finder which takes as input a ranked list of DNA or RNA sequences and returns significant motifs in PWM format. In cases where sequence ranking is not relevant or not available, it allows the use of positive and negative sets of sequences, searching for enriched motifs in the positive set using the negative set as the background.

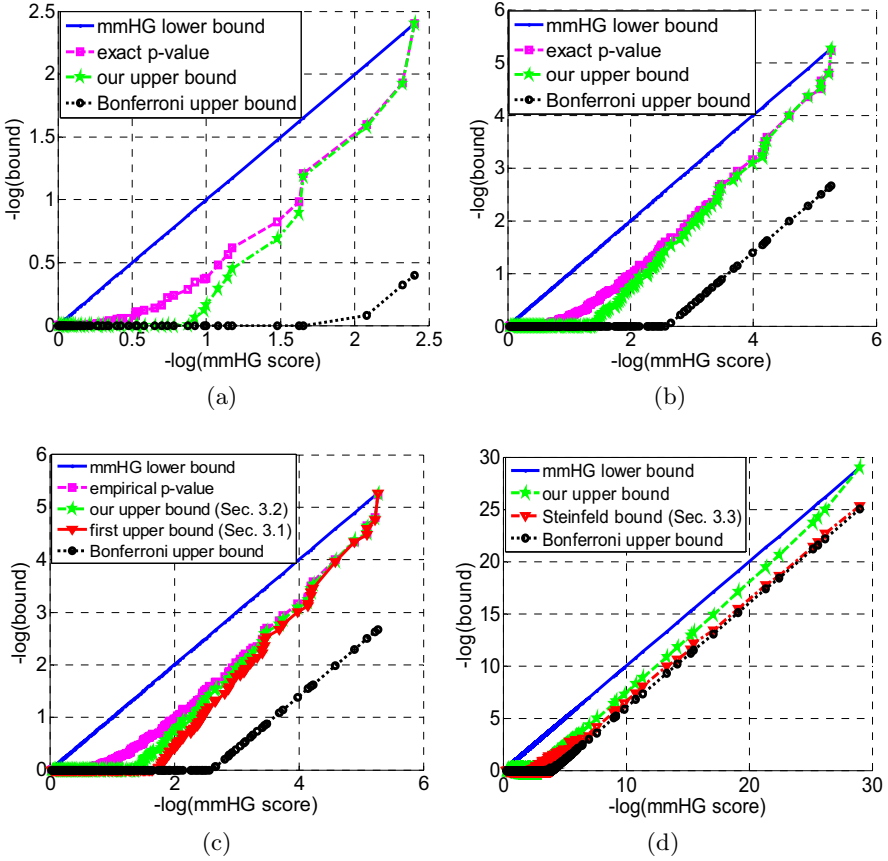


Fig. 1. (A) Four lines are shown for $N=10$: the mmHG score, which also serves as a lower bound for the p-value; the exact p-value calculated by enumerating all $10!$ permutations; our upper bound, in its refined version; and the Bonferroni corrected p-value. (B) Here again the four lines are shown – for $N=20$. However, instead of an exact p-value, which cannot be calculated exhaustively, an empirical p-value is produced by randomly sampling 10^7 permutations. (C) In addition to the four lines shown in B, the first upper bound – introduced in Section 3.1 – is shown ($N=20$). (D) Four lines are shown for $N=100$: the mmHG score, our upper bound, the bound introduced in Section 3.3 (Steinfeld bound) and the Bonferroni corrected p-value. The exact p-value line is positioned between the green and the blue lines.

We will now describe the methodology implemented in mmHG-Finder:

Input:

- a ranked list of sequences (or, alternatively, two sets of sequences representing target and background)
- motif width, given as a range between k_1 and k_2

Algorithm:

1. Build a generalized suffix tree for the sequences
2. Traverse the tree to find all k -mers for $k=k_1, \dots, k_2$
3. Sort the k -mers according to their enrichment at the top of the list (this is done using the mHG statistics), as explained in Leibovich *et al.* [15]
4. Take the most significant fifty k -mers, to be used as starting points for the next step. This set of candidates is chosen such that the members are quite different. Note that this is a heuristic approach and the number 50 is somewhat arbitrary, chosen to succeed in catching the best performing PWMs without heavily paying in complexity.
5. For each starting point, we iteratively replace one position in the k -mers by considering all possible IUPAC replacements and taking the one that improves the enrichment the most. We repeat this process for all positions several times. Eventually we get a motif in the IUPAC alphabet which is then converted to a PWM.
6. The PWMs found in the previous step are assessed using the mmHG statistics and the best is returned as output, together with the p-value. The score assigned by a PWM to a string $S = S_1, \dots, S_M$ is defined as $\max_{1 \leq i \leq M-K+1} \sum_{j=1}^K m_{S_{i+j-1},j}$ (assuming $M \geq K$, otherwise it is $-\infty$), where $m_{\alpha,c}$ is the score in row α , column c of the position weight matrix. In other words, the PWM score calculated for S is the maximal score obtained for a substring of S .

To evaluate the performance of mmHG-Finder in comparison to other state-of-the-art methods we ran it on 18 example cases – 3 synthetically generated cases and 15 generated from high throughput binding experiments (6 transcription factors and 9 RNA-binding proteins). We compared the results to those obtained by using three other methods: the standard MEME program [3], DREME [4], and XXmotif [17]. Some of the results of this comparison are summarized in Table 1. The synthetic examples were generated by randomizing 500 sequences of length 100. An IUPAC motif was generated and planted in all top 64 sequences. mmHG-Finder outperformed all the other three tools on the synthetic examples, which contained degenerate motifs. MEME and DREME did not find the motifs in any case, while XXmotif found a similar result in 1 out of the 3 tests. The other 15 examples were taken from DNA and RNA high-throughput experiments [23],[10],[12]. In 12 out of these 15 datasets, mmHG-Finder found the motifs which were compatible with the known literature motifs as the most significant result. In comparison, DREME found the known consensus in 11 cases; XXmotif detected the literature motif in 9 cases while MEME detected the known motif in only 7 cases. In several datasets, such as for GCN4 and Pin4, mmHG-Finder identified the consensus motif while other tools returned repetitive sequences as their top results. The mmHG statistics avoids such spurious results as they typically do not correlate with the measurement driven ranking.

Computing p-value bounds for the synthetic examples ($N=500$) took 7-17 seconds on a simple single-core laptop. The running time depends on both the number of elements N as well as the mmHG score. The computation is optimized such that it is quicker for smaller mmHG scores. It took 33 minutes for $N=5000$ where the mmHG score was $3.3 \cdot 10^{-69}$, and 39 minutes for $N=4000$ and mmHG score = $5.9 \cdot 10^{-31}$.

Table 1. We evaluated the performance of mmHG-Finder in comparison to other state-of-the-art methods: MEME, DREME and XXmotif. Almost all input examples comprised ranked lists, except for p53 (comprising target and background sets). Since MEME, DREME, and XXmotif expect a target set as input, we converted the ranked lists into target sets by taking the top 100 sequences for MEME (restricted by MEME’s limitation of 60,000 characters) and the top 20 % sequences for the other tools. In the synthetic examples the entire ranked lists were taken as they are sufficiently small. Data and consensus motifs for p53 were taken from [23]; for REB1, CBF1, UME6, TYE7, GCN4 from [10]; and for the RNA binding proteins from [12]. Selected results are shown below.

The protein and its consensus binding motif	mmHG-Finder	MEME	DREME	XXmotif
Synthetic TNWMNG W=[A/T], M=[A/C]	$P < 2.76e-14$ 	7.0e+003 	Nothing found	2.98e+00
Synthetic CTNNNAT	$P < 1.32e-28$ 	7.1e+001 	Nothing found	1.84e+01
Synthetic MMMMMMMM M=[A/C]	$P < 1.07e-39$ 	1.8e+002 	Nothing found	1.58e+01
P53 (DNA) 	$P < 1.09e-174$ 	1.8e-100 	4.9e-133 	1e-490
GCN4 (DNA) TGAsTCa	$P < 2.05e-44$ 	1.3e-85 	2.0e-32 	4.00e-17
Puf5 (RNA) 	$P < 7.93e-79$ 	3.6e-9 	6.8e-42 	9.76e-21
		3.1e-004 	3.1e-012 	1.61e-20
Pin4 (RNA) 	$P < 8.18e-8$ 	1.3e+0 	3.1e-51 	1.65e-28

4 Concluding Remarks

Due to the size of the measure space, statistics over S_N is difficult to implement. We derive polynomially computable bounds for the tail distribution of the mutual enrichment at the top of two permutations uniformly and independently drawn over S_N . We assess tightness using simulated data. We also demonstrate utility of the mmHG statistics in identifying motifs in experimental binding affin-

ity data. For several representative datasets, including synthetically generated data, we note that our bound improves the p-value estimates by a factor of 50. The full characterization of the distribution of mmHG as a random variable over S_N remains an open question.

Acknowledgments. We thank Israel Steinfeld for critical and inspiring discussions. We also thank the anonymous reviewers for their useful comments. LL was partially supported by Israel Ministry of Science and Technology and by ISEF Fellowship.

References

1. Abramowitz, M., Stegun, I.A.: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables (1964)
2. Akavia, U.D., Litvin, O., Kim, J., Sanchez-Garcia, F., Kotliar, D., Causton, H.C., Pochanard, P., Mozes, E., Garraway, L.A., Pe'er, D.: An Integrated Approach to Uncover Drivers of Cancer. *Cell* 143(6), 1005–1017 (2010)
3. Bailey, T.L., Elkan, C.: Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning* 21(1-2), 51–80 (1995)
4. Bailey, T.L.: DREME: motif discovery in transcription factor ChIP-seq data. *Bioinformatics* 27(12), 1653–1659 (2011)
5. Dehan, E., Ben-Dor, A., Liao, W., Lipson, D., Frimer, H., Riesenstein, S., Simansky, D., Krupsky, M., Yaron, P., Friedman, E., et al.: Chromosomal aberrations and gene expression profiles in non-small cell lung cancer. *Lung Cancer* 56(2), 175–184 (2007)
6. Eden, E., Navon, R., Steinfeld, I., Lipson, D., Yakhini, Z.: GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists. *BMC Bioinformatics* 10(1), 48 (2009)
7. Eden, E., Lipson, D., Yogev, S., Yakhini, Z.: Discovering Motifs in Ranked Lists of DNA Sequences. *PLoS Comput. Biol.* 3(3), e39 (2007)
8. Enerly, E., Steinfeld, I., Kleivi, K., Leivonen, S.-K., Ragle-Aure, M., Russnes, H.G., Rønneberg, J.A., Johnsen, H., Navon, R., Rødland, E., et al.: miRNA-mRNA Integrated Analysis Reveals Roles for miRNAs in Primary Breast Tumors. *PLoS ONE* 6(2), e16915 (2011)
9. Hafner, M., Landthaler, M., Burger, L., Khorshid, M., Hausser, J., Berninger, P., Rothbaler, A., Ascano Jr., M., Jungkamp, A.-C., Munschauer, M., et al.: Transcriptome-wide Identification of RNA-Binding Protein and MicroRNA Target Sites by PAR-CLIP. *Cell* 141(1), 129–141 (2010)
10. Harbison, C.T., Gordon, D.B., Lee, T.I., Rinaldi, N.J., Macisaac, K.D., Danford, T.W., Hannett, N.M., Tagne, J.-B., Reynolds, D.B., Yoo, J., et al.: Transcriptional regulatory code of a eukaryotic genome. *Nature* 431(7004), 99–104 (2004)
11. Hertz, G.Z., Stormo, G.D.: Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15(7-8), 563–577 (1999)
12. Hogan, D.J., Riordan, D.P., Gerber, A.P., Herschlag, D., Brown, P.O.: Diverse RNA-Binding Proteins Interact with Functionally Related Sets of RNAs, Suggesting an Extensive Regulatory System. *PLoS Biol.* 6(10), e255 (2008)
13. Lebedeva, S., Jens, M., Theil, K., Schwanhäusser, B., Selbach, M., Landthaler, M., Rajewsky, N.: Transcriptome-wide Analysis of Regulatory Interactions of the RNA-Binding Protein HuR. *Molecular Cell* 43(3), 340–352 (2011)

14. Lee, B.-K., Bhinge, A.A., Iyer, V.R.: Wide-ranging functions of E2F4 in transcriptional activation and repression revealed by genome-wide analysis. *Nucleic Acids Research* 39(9), 3558–3573 (2011)
15. Leibovich, L., Yakhini, Z.: Efficient motif search in ranked lists and applications to variable gap motifs. *Nucleic Acids Research* 40(13), 5832–5847 (2012)
16. Leibovich, L., Paz, I., Yakhini, Z., Mandel-Gutfreund, Y.: DRIMust: a web server for discovering rank imbalanced motifs using suffix trees. *Nucleic Acids Research* 41(W1), W174–W179 (2013)
17. Luehr, S., Hartmann, H., Söding, J.: The XXmotif web server for eXhaustive, weight matrix-based motif discovery in nucleotide sequences. *Nucleic Acids Research* 40(W1), W104–W109 (2012)
18. Plis, S.M., Weisend, M.P., Damaraju, E., Eichele, T., Mayer, A., Clark, V.P., Lane, T., Calhoun, V.D.: Effective connectivity analysis of fMRI and MEG data collected under identical paradigms. *Computers in Biology and Medicine* 41(12), 1156–1165 (2011)
19. Ragle-Aure, M., Steinfeld, I., Baumbusch, L.O., Liestøl, K., Lipson, D., Nyberg, S., Naume, B., Sahlberg, K.K., Kristensen, V.N., Børresen-Dale, A.-L., et al.: Identifying In-Trans Process Associated Genes in Breast Cancer by Integrated Analysis of Copy Number and Expression Data. *PLoS ONE* 8(1), e53014 (2013)
20. Rhee, H.S., Pugh, B.F.: Comprehensive Genome-wide Protein-DNA Interactions Detected at Single-Nucleotide Resolution. *Cell* 147(6), 1408–1419 (2011)
21. Al-Shahrouh, F., Díaz-Uriarte, R., Dopazo, J.: FatiGO: a web tool for finding significant associations of Gene Ontology terms with groups of genes. *Bioinformatics* 20(4), 578–580 (2004)
22. Sinha, S.: On counting position weight matrix matches in a sequence, with application to discriminative motif finding. *Bioinformatics* 22(14), e454–e463 (2006)
23. Smeenk, L., van Heeringen, S.J., Koepfel, M., van Driel, M.A., Bartels, S.J.J., Akkers, R.C., Denissov, S., Stunnenberg, H.G., Lohrum, M.: Characterization of genome-wide p53-binding sites upon stress response. *Nucleic Acids Research* 36(11), 3639–3654 (2008)
24. Staden, R.: Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Research* 12(1 Part 2), 505–519 (1984)
25. Steinfeld, I., Navon, R., Ach, R., Yakhini, Z.: miRNA target enrichment analysis reveals directly active miRNAs in health and disease. *Nucleic Acids Research* 41(3), e45–e45 (2013)
26. Steinfeld, I., Navon, R., Ardigò, D., Zavaroni, I., Yakhini, Z.: Clinically driven semi-supervised class discovery in gene expression data. *Bioinformatics* 24(16), i90–i97 (2008)
27. Stormo, G.D., Schneider, T.D., Gold, L.: Quantitative analysis of the relationship between nucleotide sequence and functional activity. *Nucleic Acids Research* 14(16), 6661–6679 (1986)
28. Straussman, R., Nejman, D., Roberts, D., Steinfeld, I., Blum, B., Benvenisty, N., Simon, I., Yakhini, Z., Cedar, H.: Developmental programming of CpG island methylation profiles in the human genome. *Nat. Struct. Mol. Biol.* 16(5), 564–571 (2009)
29. Subramanian, A., Tamayo, P., Mootha, V.K., Mukherjee, S., Ebert, B.L., Gillette, M.A., Paulovich, A., Pomeroy, S.L., Golub, T.R., Lander, E.S., et al.: Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences of the United States of America* 102(43), 15545–15550 (2005)

Probabilistic Approaches to Alignment with Tandem Repeats

Michal Nánási, Tomáš Vinař, and Broňa Brejová

Faculty of Mathematics, Physics, and Informatics, Comenius University,
Mlynská dolina, 842 48 Bratislava, Slovakia

Abstract. We propose a simple tractable pair hidden Markov model for pairwise sequence alignment that accounts for the presence of short tandem repeats. Using the framework of gain functions, we design several optimization criteria for decoding this model and describe the resulting decoding algorithms, ranging from the traditional Viterbi and posterior decoding to block-based decoding algorithms specialized for our model. We compare the accuracy of individual decoding algorithms on simulated data and find our approach superior to the classical three-state pair HMM in simulations.

1 Introduction

In this paper, we explore the use of pair hidden Markov models (pair HMMs, PHMMs) in improving the quality of pairwise sequence alignment in the presence of tandem repeats. We propose a simple tractable model that explicitly accounts for short tandem repeats, and we use the framework of maximum expected gain to explore a variety of decoding optimization criteria for our model.

Pair HMMs have for a long time played a major role in sequence alignment [4]. The traditional Needleman-Wunsch algorithm [19] and its variants can be easily formulated as a special case of alignment with PHMMs (we call this approach Viterbi decoding). The main advantage of PHMMs is that they allow to express the scoring scheme in a principled way in the context of a probabilistic model.

Sequence alignments are a mainstay of comparative genomics. By comparing sequences that evolved from a common ancestor, we can infer their phylogenetic relationships, discover sites under functional constraint, or even shed light on the function of individual sequence elements. However, comparative genomic methods are very sensitive to the quality of underlying alignments, and even slight inaccuracies may lead to artifacts in the results of comparative methods.

It is very difficult to evaluate alignment accuracy, yet even simple statistics can reveal artifacts of present-day algorithms. Lunter et al. [16] demonstrated systematic biases caused by the optimization criteria set by the Needleman-Wunsch approach. They show that by using variants of the posterior decoding instead of the traditional Viterbi algorithm, one can significantly increase the quality of alignments. While the Viterbi decoding seeks one highest scoring alignment, the posterior decoding summarizes information from all alignments of the two sequences. This approach was also found superior by other authors [13, 18, 23].

An algorithm by Hudek [14] is an intermediate between the Viterbi and posterior decoding, summarizing probabilities of alignments within short blocks. The goal is to segment the alignment into blocks, where each block has gaps in only one of the two sequences. The decoding algorithm considers each block as a unit, summing probabilities of all alignments that had the same block structure. Finally, Satija et al. [22] have demonstrated that fixing a particular alignment is not necessary in some comparative genomics applications, instead one can consider all possible alignments weighted by their probability in the PHMM.

In this paper, we concentrate on modeling sequence alignments in the presence of tandem repeats. Short tandem repeats cover more than 2% of the human genome, and occur in many genes and regulatory regions [9]; in fact, majority of recent short insertions in human are due to tandem duplication [17]. Evolution of tandem repeats is dominated by tandem segmental duplications resulting in regions composed of a highly variable number of almost exact copies of a short segment. Such sequences are difficult to align with standard scoring schemes, because it is not clear which copies from the two organisms are orthologous. Misalignments due to the presence of short tandem repeats are usually not limited to the repetitive sequence itself, but may spread into neighbouring areas and impact the overall alignment quality.

Sequence alignment with tandem duplication was first studied by Benson [1]. They propose an extension of the traditional Needleman-Wunsch algorithm that can accommodate tandem repeats in $O(n^4)$ time. They also propose several faster heuristic algorithms. Additional work in this area concentrated on computing variants of edit distance either on whole sequences with tandem arrays or on two tandem arrays using different sets of evolutionary operations [3, 7, 21].

The first probabilistic approach to alignment of tandem duplications was introduced by Hickey and Blanchette [12], who developed a new probabilistic model by combining PHMMs with Tree Adjoining Grammars (TAGs). Their model favors tandem duplications to other insertions, but the approach does not explicitly model whole arrays of tandemly repeated motifs. Moreover, algorithms to train and decode such models are relatively complex.

Protein sequences with repetitive motifs (such as zinc finger proteins) are a special class of proteins and their alignment has many features in common with DNA sequence alignment with tandem repeats. [15] combined profile HMMs (capturing the properties of the repeating motif) and PHMMs (modeling alignments) into a single scoring scheme that can be decoded by a newly proposed algorithm. However, their scoring scheme is no longer a probabilistic model and the method is focused on correctly aligning individual occurrences of a single motif rather than alignment of long sequences interspersed with multiple motifs.

Here, we propose a simple tractable PHMM for sequence alignment with tandem repeats, and we explore various decoding methods for use of this model in sequence alignment. In addition to the classical Viterbi decoding, we define several variants based on the posterior decoding and block-based methods tailored to the specifics of our model. To demonstrate the differences, we have implemented several of these methods and compared their performance.

2 Pair HMM for Alignment with Tandem Repeats

Tandem repeats may arise by a complicated sequence of evolutionary events, including multiple rounds of tandem duplication, deletion, point mutation, gene conversion and other phenomena. Tandem repeat arrays at homologous locations in two related species may have arisen in the common ancestor and thus share part of their evolutionary history, but they could be further modified by independent events occurring after speciation. Models attempting to capture such diverse evolutionary mechanisms usually lead to complex problems in inference and parameter estimation. We propose a tractable model, based on classical PHMMs, which still captures the essence of a tandem repeat array: periodically repeating motif, which may be shared between the two species, or be specific for one species only.

A PHMM defines a probability distribution over alignments of two sequences X and Y . The standard PHMM has three states: match state M generating ungapped columns of the alignment, and two insert states I_X and I_Y , where I_X generates alignment columns with a symbol from X aligned to a gap, and I_Y generates columns with a symbol from Y aligned with a gap [4]. In our work, we will use a more complex PHMM, but standard algorithms for inference in these models are still applicable.

We call our model SFF and its details are shown in Fig.1. The model contains a standard three-state PHMM and two “sunflower” submodels $R_{i,X}$ and $R_{i,Y}$ for each possible repeating motif i . Submodel $R_{i,X}$ generates several (possibly zero) copies of the motif in sequence X and submodel $R_{i,Y}$ generates motif copies in sequence Y . Each copy of the motif is generated independently and the number of copies in X and Y are independent and geometrically distributed.

Each sunflower submodel is a circularized profile HMM emitting copies of the motif in one of the two sequences. For a motif of length p , the submodel contains p match states M_0, \dots, M_{p-1} , each match state emitting one symbol of the motif. Insertion state I_j allows to insert additional characters between symbols emitted by M_j and $M_{(j+1) \bmod p}$. Deletion states D_j and D'_j allow to bypass match state M_j , and thus correspond to deletions with respect to the reference motif sequence. Since the submodel can emit multiple tandem copies of the motif, the states in column $p - 1$ are connected to the states in column 0. To avoid cycles consisting solely of silent states, we use two separate chains of deletion states. Chain D'_0, \dots, D'_{p-2} can be entered only in state D'_0 , and model can stay in this chain for at most $p - 1$ steps. Chain D_1, \dots, D_{p-1} can be entered only after visiting at least one match or insert state in the current copy of the motif. As a result, the model can never pass around the whole circle using delete states. Note that the model prefers integer number of repeats, even though partial repeat occurrences are common in the real data. If desired, this can be addressed by simple changes in the model topology or parameters.

The overall model can have sunflower submodels for an arbitrary number of motifs; we can even define an infinite model, in which every possible finite string serves as the consensus for one pair of sunflowers. In our work, we use $k = 310,091$ motifs chosen as consensus strings of all tandem repeats found by

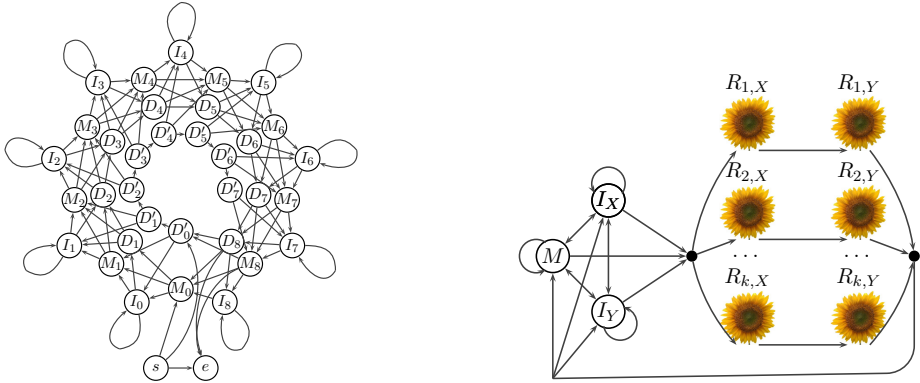


Fig. 1. SFF (sunflower field) model: a pair hidden Markov model for alignment with tandem repeats. Each submodel $R_{i,\alpha}$ (left) is a circular profile hidden Markov model emitting tandem copies of the motif in one sequence. State M_j is the match state generating j th symbol of the motif, state I_j allows insertions between symbols j and $j+1$ of the motif, and states D_j and D'_j allow to skip state M_j . States s and e designate the entry and exit points from the submodel. The full SFF model (right) contains a standard three-state PHMM with states M , I_X and I_Y , and two submodels $R_{i,X}$, $R_{i,Y}$ for each motif i . States and submodels with subscript X and Y generate symbols in the respective sequence X or Y only.

the TRF program [2] run on the human chromosome 15 and its homologous sequences in the dog genome. The probability of choosing a particular motif out of all k possibilities can be uniform or dependent on the motif length or composition. We assign this probability based on the observed frequency of the corresponding consensus pattern in the TRF output. Alternatively, we could use a much smaller model by Frith [8]; however, this model does not easily handle insertions and deletions within repeats.

Likewise, we could use a multiple alignment of real motif occurrences to set individual parameters of the profile HMM. Instead, we use the same set of parameters for all states of all motif submodels. In particular, we set the insert and delete rates to 0.005; the match states allow mutations away from consensus according to the Jukes-Cantor model with parameter $t = 0.05$. Parameters of the three-state PHMM were estimated from the UCSC alignment of the human chromosome 15 and its homologous regions in the dog genome.

Our model also assumes that individual copies of a fixed motif are independent. If they share part of their evolutionary history, this assumption is not valid, but it greatly simplifies the model. We could add some limited dependence by introducing repeat submodels emitting copies in the two sequences simultaneously; we have used such a model in a different setting in our previous work [15].

<p>X: A C - - - - T Y: - C G A A A T</p>	<p>a_i: 1 2 -2 -2 -2 -2 3 b_i: -0 1 2 3 4 5 6 s_i: I_X M I_Y r r r R r_i: 0 0 0 1 1 1 0</p>
---	---

Fig. 2. Example of an alignment represented in our notation, together with its state and repeat annotation. Assuming that submodel $R_{1,Y}$ in the SFF model represents consensus sequence A, state r in the state sequence is a shorthand for the state M_1 within $R_{1,Y}$.

3 Inference Criteria and Algorithms

Given the SFF model introduced in the previous section, and two sequences $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$, we wish to find the alignment of these two sequences best agreeing with the model. We can also annotate this alignment by labeling individual alignment columns with additional information. We start by defining an alignment and its annotation more formally (see Fig.2). An alignment of X and Y is a sequence of pairs $(a_1, b_1), \dots, (a_t, b_t)$, each pair representing one alignment column. Symbol a_i represents either a position in X , or a gap annotated with the position of the nearest non-gap symbol on the left; formally $a_i \in \{1, \dots, n\} \cup \{-0, -1, \dots, -n\}$. To specify a valid alignment, a_1 must be 1 or -0 , a_t must be n or $-n$, and if $a_i \in \{j, -j\}$, a_{i+1} must be $j + 1$ or $-j$. The conditions on symbols b_i representing positions in sequence Y are analogous. The state annotation of an alignment is a sequence of states $s_1 \dots s_t$ such that state s_i generated alignment column (a_i, b_i) . The repeat annotation is a binary sequence $r_1 \dots r_t$, where $r_i = 1$ if the state s_i generating the i -th column is one of the states in the repeat submodels. While the state annotation can be used with any PHMM generating the alignment, the repeat annotation is appropriate only for the SFF model or other PHMMs explicitly modeling repeats.

We will explore several inference criteria for choosing the best alignment. To describe them, we will use the terminology of gain functions [10]; analogous notion of a loss functions is frequently used for example in statistics and machine learning. A gain function $G(A, A_T)$ evaluates similarity between a predicted alignment A and the correct alignment A_T ; higher gain meaning that the prediction is of higher quality. Since the true alignment A_T is not known, we will consider the expected gain $E_{A_T}[G(A, A_T)|X, Y]$ of alignment A , assuming that sequences X and Y were generated by our model

$$E_{A_T}[G(A|A_T)|X, Y] = \sum_{A_T} G(A, A_T) \Pr(A_T|X, Y).$$

In each optimization criterion, we choose a particular gain function and look for alignment A^* maximizing the expected gain $A^* = \arg \max_A E_{A_T}[G(A, A_T)|X, Y]$. Note that the gain function is only a way of defining the optimal solution; the corresponding decoding algorithm needs to be designed on a case-by-case basis.

3.1 Decoding Criteria for the Three-State PHMM

For simplicity, we start with criteria for the three-state PHMM, where the state annotation is uniquely determined by the alignment itself.

Viterbi decoding. Perhaps the simplest gain function assigns gain +1 if the predicted alignment A is identical to the true alignment A_T , and 0 otherwise. To optimize this gain function, we need to find the alignment with the highest overall probability in the model. In the simple three-state PHMM, this alignment can be found by the classical Viterbi algorithm in time $O(nmE)$, where E is the number of non-zero transitions in the model.

Posterior decoding. While the Viterbi decoding assigns gain only if the whole alignment is correctly predicted, posterior decoding assigns gain +1 for each correctly identified alignment column. Recall that the column is a pair (a_i, b_i) , and it is considered correct, if the same column also occurs somewhere in the true alignment. The optimal alignment under this gain function can be found by computing the posterior probability of each alignment column using the forward and backward algorithms for PHMMs, and then finding the alignment as a collection of compatible columns with the highest sum of posterior probabilities. A similar algorithm is considered for example by Lunter et al. [16], except that the column posteriors are multiplied rather than added. The running time of this algorithm is again $O(nmE)$.

Marginalized posterior decoding. Lunter et al. [16] also consider a variant of posterior decoding, where a column $(i, -j)$ is considered correct and receives a gain +1, if the true alignment contains a column $(i, -\ell)$ for any value of ℓ . In other words, when symbol x_i is aligned to a gap, we do not distinguish where is the location of this gap with respect to sequence Y . Columns $(-j, i)$ are treated symmetrically. To optimize this gain function, we again start by computing posteriors of all columns. Then we marginalize the probabilities of gap columns, effectively replacing posterior of column $(i, -j)$ with the sum of posteriors of columns $(i, -\ell)$ for all ℓ . As before, we then find the alignment maximizing the sum of posteriors of its columns. The algorithm runs in $O(nmE)$ time.

3.2 Decoding Criteria for the SFF Model

In more complex models, including ours, one alignment can be generated by several different state paths. Various gain functions can thus take into account also the state or repeat annotation of the alignment.

Viterbi decoding. In more complex models, the classical Viterbi algorithm optimizes a gain function in which the alignment is annotated with the state path generating it, and gain is awarded only when both the alignment and the state path are completely correct.

Posterior and marginalized posterior decoding. We will consider a variant of the posterior decoding, in which alignment columns are annotated by the repeat annotation, and an alignment column gets a gain +1, if the true alignment contains the same column with the same label. The only change in the algorithm is that the forward-backward algorithm produces posterior probabilities of columns annotated with the state, which are then marginalized over all states with the same repeat label. The running time is still $O(nmE)$. Similar modification can be done for marginalized posterior decoding, where we marginalize gap columns based on both state and gap position.

Block decoding. We will consider also a stricter gain function, which requires that repeat regions have correctly identified boundaries. We will split the alignment annotated with repeats into *blocks*, so that each maximal region of consecutive columns labeled as a repeat forms a block. Each column annotated as a non-repeat also forms a separate block. The gain function awards gain +1 for each non-gap symbol in every correctly predicted and labeled block. Correctness of non-repeat columns is defined as in posterior decoding. A repeat block is considered correct, if exactly the same region in X and the same region in Y are also forming a repeat block in the true alignment. Note that the gain for each block is proportional to the number of non-gap symbols in the block to avoid biasing the algorithm towards predicting many short blocks.

To optimize this gain function, we first compute posterior probabilities for all blocks. Note that a block is given by a pair of intervals, one in X and one in Y . Therefore the number of blocks is $O(n^2m^2)$. The expected gain of a block is its posterior probability multiplied by the number of its non-gap symbols. After computing expected gains of individual blocks, we can find the highest scoring combination of blocks by dynamic programming in $O(n^2m^2)$ time.

To compute block posterior probabilities, we transform the SFF model to a generalized PHMM [20], in which all repeat states are replaced by a single generalized state R . In generalized HMMs, emission of a state in one step can be an arbitrary string, rather than a single character. In our case, the new state R generates a pair of sequences from the same distribution as defined by one pass through the repeat portion of the original SFF model. Pair of sequences generated by R represents one block of the resulting alignment. We call this new model the block model. Using the forward-backward algorithm for generalized HMMs, we can compute posterior probabilities of all blocks in $O(n^2m^2f)$ time where f is the time necessary to compute emission probability for one particular block.

If we naively compute each emission separately, we get $f = O(nmE)$. However, we can reduce this time as follows. If the SFF contains only one motif, the emission probability of sequences x and y in the R model is simply

$$\Pr(x, y | R) = \Pr(x | R_{1,X}) \Pr(y | R_{1,Y}),$$

because the model first generates x in the sunflower submodel $R_{1,X}$ and then generates y in the model $R_{1,Y}$. Note that these two models are connected by

a transition with probability 1. In the general case, we sum the probabilities for all k motifs, each multiplied by the transition probability of entering that motif. To compute block emission probabilities fast, we precompute $\Pr(x | R_{i,X})$ and $\Pr(y | R_{i,Y})$ for all substrings x and y of sequences X and Y respectively. This can be done by the forward algorithm in $O((n^2 + m^2)E)$ time. After this preprocessing, the computation of emission probability is $O(k)$, and the overall running time of this algorithm is $O(kn^2m^2 + (n^2 + m^2)E)$.

Block Viterbi decoding. The final gain function we consider is a variant of the Viterbi decoding. The Viterbi decoding assigns gain +1 for a completely correct alignment labeled with a correct state annotation. One alternative is to assign gain +1 if the alignment and its repeat annotation are completely correct. This gain function considers as equivalent all state paths that have the same position of repeat boundaries but use different motifs or different alignments of the sequence to the motif profile HMM.

In the SFF model, location of a repeat block uniquely specifies alignment within the block, because all symbols from sequence X must come first (aligned to gaps), followed by symbols from sequence Y . However, some models may emit repeat bases from the two sequences aligned to each other. We wish to abstract from exact details of repeat alignment, and consider different alignments within a repeat as equivalent. Therefore, we will reformulate the gain function in terms of blocks. The alignment labeled with repeat annotation gets a gain 1, if all its blocks are correct, where block correctness is determined as in the block decoding. This formulation is similar to the one solved by Hudek [14].

To optimize this gain function, we use the Viterbi algorithm for generalized HMMs applied to the block model, which leads to running time $O(kn^2m^2 + (n^2 + m^2)E)$, by similar reasoning as above.

3.3 Practical Considerations

Even the fastest algorithms described above require $O(nmE)$ time, where sequence lengths n and m can be quite high when aligning whole syntenic genomic regions and the size of the model E depends on the sum of the lengths of all repeat motifs, which can be potentially even infinite. However, we can use several heuristic approaches to make the running times reasonable.

First of all, we can use the standard technique of banding, where we restrict the alignment to some window around a guide alignment obtained by a faster algorithm. A simpler form of banding is to split the guide alignment to non-overlapping windows and realign each window separately. These techniques reduce the $O(nm)$ factor.

To restrict the size of the model, we first find tandem repeats in X and Y independently by running the TRF program [2]. Then we include in our model only those motifs which appear at least once in the TRF output. If we process only relatively short windows of the banded alignment, the size of the model will be quite small. Note however, that we keep the transition probabilities entering these models the same as they are in the full SFF model. If TRF finds a consensus

not included in the original SFF model, we add its two submodels with a small probability comparable to the rarest included motifs.

These two heuristics sufficiently speed up algorithms running in $O(nmE)$ time. The block decoding and the block Viterbi decoding need to consider all possible blocks, which is prohibitive even within short alignment windows. Therefore, we limit possible repeat blocks only to intervals discovered as repeats by the TRF program. We allow the generalized repeat state R to generate the block of substrings x and y if each of these substrings is either empty or one of the intervals found by TRF has both its endpoints within 10 bases from the respective endpoints of x or y . Therefore, if TRF finds t_X intervals in X and t_Y intervals in Y , we try at most $(20t_X + n)(20t_Y + m)$ blocks.

The final consideration is that the SFF model does not align tandem repeats at orthologous locations, even if they share a common evolutionary history. This might be impractical for further use. Therefore we postprocess the alignments by realigning all blocks annotated as repeats using the standard three-state PHMM. In this realignment, we also include gaps adjacent to these repeats.

4 Experiments

We have compared decoding methods described in the previous section and several baseline algorithms on simulated data (see Table 1). The data set contained 200 alignments of length at least 200 each generated from the SFF model (the same model parameters were used in the sampling and for the alignments). In generating the dataset, we required that each tandem repeat had at least three copies in both species; otherwise, we would obtain many regions that would be labeled as tandem repeats, but would in fact only have a single copy. The error rate (the first column of the table) measures the fraction of true alignment columns that were not found by a particular algorithm. It was measured only on the alignment columns that were generated from non-repeat states in the simulation, as the SFF model does not give any alignment in repeat regions.

The first observation is that the methods based on the SFF model (the first block of the table) outperform the baseline method (the Viterbi algorithm on the three-state model), reducing the error rate by 10–30%. In general, the methods that score individual alignment columns are more accurate than the block-based or the Viterbi-based methods, which is not surprising, because error rate as a measure of accuracy is closer to the gain function they optimize. We have also compared our approach to the related method of context-sensitive indels [12]. The Context program was trained on a separate set of 200 alignments sampled from our model. However, its error rate is quite high, perhaps due to insufficient training data or some software issues. Finally, we have also run the Muscle aligner with default parameters [5]; we have used the result as a guide alignment for the slower block-based decoding methods (the new alignment was restricted to be within 30 base window from the guide alignment).

The SFF-based algorithms use the tandem repeat motifs predicted by the TRF, as well as approximate repeat intervals (block-based methods). The TRF

Table 1. Accuracy of several decoding methods on simulated data. *: method uses the real consensus motifs. **: method uses the real consensus motifs and intervals from the real repeat blocks.

Algorithm	Alignment	Repeat		Block	
	error	sn.	sp.	sn.	sp.
SFF marginalized	3.37%	95.97%	97.78%	43.07%	44.87%
SFF posterior	3.53%	95.86%	97.87%	42.70%	47.37%
SFF block	3.87%	91.20%	98.04%	36.13%	47.14%
SFF block Viterbi	4.32%	91.28%	97.96%	35.40%	45.97%
SFF Viterbi	4.04%	95.29%	97.85%	42.70%	48.95%
SFF marginalized*	3.02%	98.93%	99.64%	77.01%	76.17%
SFF posterior*	3.42%	98.84%	99.51%	75.91%	80.93%
SFF block**	3.21%	97.70%	99.87%	80.66%	94.44%
SFF block Viterbi**	3.71%	98.12%	99.85%	81.75%	92.18%
SFF Viterbi*	3.94%	98.54%	99.45%	75.55%	83.47%
Context	5.98%				
Muscle	5.62%				
3-state posterior	4.41%				
3-state Viterbi	4.78%				

predictions are not exact and may contribute to the overall error rate. We attempted to quantify this effect by using the real tandem repeat motifs and real boundary positions instead of the TRF predictions (the second block of Table 1). We can see that the use of TRF predictions indeed leaves space for improvement, with the best performing algorithm reducing the error rate by almost 40% compared to the baseline. Block-based methods work significantly better with true intervals than with the TRF intervals, suggesting that further improvements in repeat interval detection are needed.

The decoding methods that use the SFF model produce an alignment and a repeat annotation. Comparing annotation of each base in both sequences with the true repeat annotation sampled from the model (table columns repeat sensitivity and specificity), we note that the marginalized posterior decoding is the most sensitive, and the block decoding the most specific method. Specificity was quite high for all methods, low sensitivity for block-based methods was probably caused by wrong repeat intervals predicted by the TRF, since it improves markedly by using correct intervals.

We also compared the accuracy of predicting repeat block boundaries (table columns block sensitivity and specificity). The number of blocks with correctly predicted boundaries is quite low, most likely because there are usually many high-probability alternatives with slightly shifted boundaries. However, even though more than half of the repeat blocks have some error in the boundary placement, the SFF-based methods improve the alignment accuracy most markedly close to repeat boundaries, as shown in Fig.3. This is expected, because far from repeats the model works similarly to the three-state PHMM.

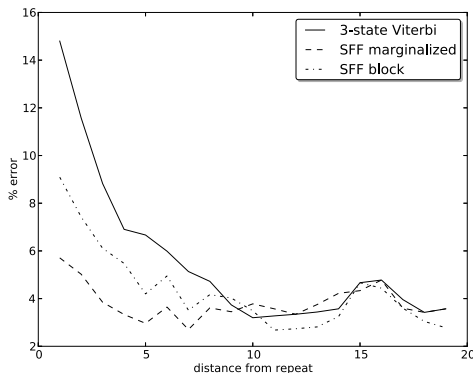


Fig. 3. Alignment error rate of three decoding methods as a function of the distance from the nearest repeat

To illustrate the feasibility of running our methods on real genomic data, we have realigned the 1.8Mb CFTR region on the human chromosome 7 to orthologous portions of the dog genome. We have started with the lastz alignment [11] downloaded from the Ensemble website [6]. We have then run the TRF program on both species and selected alignment windows of length 50–350 that contained at least one repeat. Regions without repeats or with very long repeats were left with the original alignment. Using the block decoding with the SFF model, we have thus realigned windows covering roughly 70% of the original alignment. About 8% of all alignment columns were annotated as repeats.

5 Conclusion

We have designed a new pair hidden Markov model for aligning sequences with tandem repeats and explored a variety of decoding optimization criteria for its use. The new model coupled with appropriate decoding algorithm reduces the error rate on simulated data, especially around boundaries of tandem repeats. With suitable heuristics, our approach can be used to realign long genomic regions.

Our experiments are the first study comparing a variety of different decoding criteria for PHMMs. Our criteria for the SFF model optimize both the alignment and the repeat annotation. Depending on the application, one or the other may be of greater interest, and thus one may want to marginalize over all alignments and optimize the annotation, as in [22], or marginalize over labels and optimize the alignment.

Our model does not take into the account the dependencies between the repeat occurrences in the two species. A tractable model allowing such dependencies would be of great interest. Previously, we have explored the problem of aligning two sequences simultaneously to a profile HMM, but we were not able to design a simple generative model for this purpose [15].

Acknowledgements. This research was funded by VEGA grant 1/1085/12.

References

1. Benson, G.: Sequence alignment with tandem duplication. *Journal of Computational Biology* 4(3), 351–357 (1997)
2. Benson, G.: Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research* 27(2), 573–580 (1999)
3. Bérard, S., Nicolas, F., Buard, J., Gascuel, O., Rivals, E.: A fast and specific alignment method for minisatellite maps. *Evolutionary Bioinformatics Online* 2, 303 (2006)
4. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press (1998)
5. Edgar, R.C.: MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 32(5), 1792–1797 (2004)
6. Flicek, P., et al.: Ensembl 2013. *Nucleic Acids Research* 41(D1), D48–D55 (2013)
7. Freschi, V., Bogliolo, A.: A lossy compression technique enabling duplication-aware sequence alignment. *Evolutionary Bioinformatics Online* 8, 171 (2012)
8. Frith, M.C.: A new repeat-masking method enables specific detection of homologous sequences. *Nucleic Acids Res.* 39(4), e23 (2011)
9. Gemayel, R., Vincés, M.D., Legendre, M., Verstrepen, K.J.: Variable tandem repeats accelerate evolution of coding and regulatory sequences. *Annual Review of Genetics* 44, 445–477 (2010)
10. Hamada, M., Kiryu, H., Sato, K., Mituyama, T., Asai, K.: Prediction of RNA secondary structure using generalized centroid estimators. *Bioinformatics* 25(4), 465–473 (2009)
11. Harris, R.: Improved pairwise alignment of genomic DNA. PhD thesis, Pennsylvania State University (2007)
12. Hickey, G., Blanchette, M.: A probabilistic model for sequence alignment with context-sensitive indels. *Journal of Computational Biology* 18(11), 1449–1464 (2011)
13. Holmes, I., Durbin, R.: Dynamic programming alignment accuracy. *Journal of Computational Biology* 5(3), 493–504 (1998)
14. Hudek, A.K.: Improvements in the Accuracy of Pairwise Genomic Alignment. PhD thesis, University of Waterloo, Canada (2010)
15. Kováč, P., Brejová, B., Vinař, T.: Aligning sequences with repetitive motifs. In: *Information Technologies - Applications and Theory (ITAT)*, pp. 41–48 (2012)
16. Lunter, G., Rocco, A., Mimouni, N., Heger, A., Caldeira, A., Hein, J.: Uncertainty in homology inferences: assessing and improving genomic sequence alignment. *Genome Research* 18(2), 298–309 (2008)
17. Messer, P.W., Arndt, P.F.: The majority of recent short DNA insertions in the human genome are tandem duplications. *Mol. Biol. Evol.* 24(5), 1190–1197 (2007)
18. Miyazawa, S.: A reliable sequence alignment method based on probabilities of residue correspondences. *Protein Engineering* 8(10), 999–1009 (1995)
19. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48(3), 443–443 (1970)

20. Pachter, L., Alexandersson, M., Cawley, S.: Applications of generalized pair hidden Markov models to alignment and gene finding problems. *Journal of Computational Biology* 9(2), 389–399 (2002)
21. Sammeth, M., Stoye, J.: Comparing tandem repeats with duplications and excisions of variable degree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3(4), 395–407 (2006)
22. Satija, R., Hein, J., Lunter, G.A.: Genome-wide functional element detection using pairwise statistical alignment outperforms multiple genome footprinting techniques. *Bioinformatics* 26(17), 2116–2120 (2010)
23. Schwartz, A.S., Pachter, L.: Multiple alignment by sequence annealing. *Bioinformatics*, 23(2), e24–e29 (2007)

Multiscale Identification of Topological Domains in Chromatin

Darya Filippova^{1,2,*}, Rob Patro^{2,*}, Geet Duggal^{1,2,*}, and Carl Kingsford²

¹ Joint Carnegie Mellon University, University of Pittsburgh Ph.D. Program in Computational Biology, Pittsburgh, PA

² Lane Center for Computational Biology, Carnegie Mellon University, Pittsburgh, PA

Abstract. Recent chromosome conformation capture experiments have led to the discovery of dense, contiguous, megabase-sized topological domains that are similar across cell types, are conserved across species. These domains are strongly correlated with a number of chromatin markers and have since been included in a number of analyses. However, functionally relevant domains may exist at multiple length scales. We introduce a new and efficient algorithm that is able to capture persistent domains across various resolutions by adjusting a single scale parameter. The identified novel domains are substantially different from domains reported previously and are highly enriched for insulating factor CTCF binding and histone modifications at the boundaries.

Keywords: chromosome conformation capture, topological domains, weighted interval scheduling.

1 Introduction

Chromatin interactions obtained from a variety of recent experimental techniques in chromosome conformation capture (3C) [3] have resulted in significant advances in our understanding of the geometry of chromatin structure [7], its relation to the regulation of gene expression, nuclear organization, cancer translocations [2], and copy number alterations in cancer [6]. Of these advances, the recent discovery of dense, contiguous regions of chromatin termed *topological domains* [4] has resulted in the incorporation of domains into many subsequent analyses [9,12,14] due to the fact that they are persistent across cell types, conserved across species, and serve as a skeleton for the placement of many functional elements of the genome [1,18].

3C experiments result in matrices of counts that represent the frequency of cross-linking between restriction fragments of DNA that are spatially near one another. The original identification of domains in Dixon et al. [4] employed a Hidden Markov Model (HMM) on these interaction matrices to identify regions initiated by significant downstream chromatin interactions and terminated by

* Contributed equally to this work.

a sequence of significant upstream interactions. A defining characteristic of the domains resulting from their analysis is that higher frequency 3C interactions tend to occur within domains as opposed to across domains. This aspect of domains is also reflected in the block-diagonal structure of 3C interaction matrices as shown in Fig. 1. In this sense, domains can be interpreted as contiguous genomic regions that self-interact frequently and are more spatially compact than their surrounding regions.

However, the single collection of megabase-sized domains may not be the only topologically and functionally relevant collection of domains. On closer inspection of the block-diagonal matrix structure in Fig. 1, it becomes clear that there are alternative contiguous regions of the chromosome that self-interact frequently and are likely more spatially compact than their surrounding regions (dotted lines). Some of these regions appear to be completely nested within others, suggesting a hierarchy of compact regions along the chromosome, while others appear to overlap each other. These observations suggest that functionally-relevant chromosomal domains may exist at multiple scales.

We introduce a new algorithm to efficiently identify topological domains in 3C interaction matrices for a given domain-length scaling factor γ . Our results suggest that there exist a handful of characteristic resolutions across which domains are similar. Based on this finding, we identify a consensus set of domains that persist across various resolutions. We find that domains discovered by our algorithm are dense and cover interactions of higher frequency than inter-domain interactions. Additionally, we show that inter-domain regions within the consensus domain set are highly enriched with insulator factor CTCF and histone modification marks. We argue that our straightforward approach retains the essence of the more complex multi-parameter HMM introduced in [4] while allowing for the flexibility to identify biologically relevant domains at various scales.

2 Problem Definitions

Given the resolution of the 3C experiment (say, 40kbp), the chromosome is broken into n evenly sized fragments. 3C contact maps record interactions between different sections of the chromosome in the form of a weighted adjacency matrix \mathbf{A} where two fragments i and j interact with frequency \mathbf{A}_{ij} .

Problem 1 (Resolution-specific domains). Given a $n \times n$ weighted adjacency matrix \mathbf{A} and a resolution parameter $\gamma \geq 0$, we wish to identify a set of domains D_γ where each domain is represented as an interval $d_i = [a_i, b_i]$, $1 \leq a_i < b_i \leq n$ such that no two d_i and d_j overlap for any $i \neq j$. Additionally, each domain should have a larger interaction frequency within domain than to its surrounding regions.

Here, the parameter γ is inversely related to the average domain size in D_γ : lower γ results in sets of larger domains and higher γ corresponds to sets of smaller domains. We define γ and discuss it in more detail later in the text.

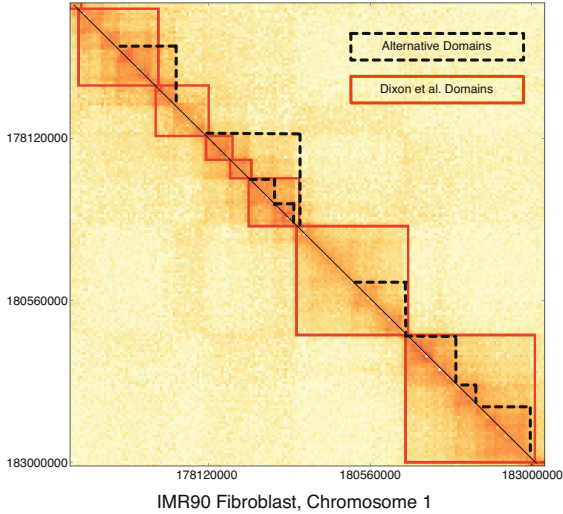


Fig. 1. Interaction matrix for a portion of human chromosome 1 from a recent Hi-C experiment by Dixon et al. [4]. Each axis represents a location on the chromosome (40kbp bins). Densely interacting domains identified by the method of Dixon et al. are shown as red boxes. Alternative domains are shown as dotted black lines on the upper triangular portion of the matrix. Visual inspection of the lower triangular portion suggests domains could be completely nested within another and highly overlapping when compared to Dixon et al.’s domains. This motivates the problem of identifying alternative domains across length scales.

Specifically, we seek to identify a set of non-overlapping domains D_γ that optimize the following objective:

$$\max_{[a_i, b_i] \in D_\gamma} \sum q(a_i, b_i, \gamma), \quad (1)$$

where q is a function that quantifies the quality of a domain $[a_i, b_i]$ at resolution γ . Since domains are required to contain consecutive fragments of the chromosome, this problem differs from the problem of clustering the graph of 3C interactions induced by \mathbf{A} , since such a clustering may place non-contiguous fragments of the chromosome into a single cluster. In fact, this additional requirement allows for an efficient optimal algorithm.

Problem 2 (Consensus domains across resolutions). Given \mathbf{A} and a set of resolutions $\Gamma = \{\gamma_1, \gamma_2, \dots\}$, identify a set of non-overlapping domains D_c that are most persistent across resolutions in Γ :

$$\max_{[a_i, b_i] \in D_c} \sum p(a_i, b_i, \Gamma), \quad (2)$$

where $p(a_i, b_i, \Gamma)$ is the persistence of domain $[a_i, b_i]$ corresponding to how often it appears across resolutions.

3 Algorithms

3.1 Domain Identification at a Particular Resolution

Since each row and corresponding column in a 3C interaction matrix encodes a genomic position on the chromosome, we can write the solution to objective (1) as a dynamic program:

$$\text{OPT}_1(l) = \max_{k < l} \{ \text{OPT}_1(k - 1) + \max\{q(k, l, \gamma), 0\} \}, \tag{3}$$

where $\text{OPT}_1(l)$ is the optimal solution for objective (1) for the sub-matrix defined by the first l positions on the chromosome ($\text{OPT}_1(0) = 0$). The choice of k encodes the size of the domain immediately preceding location l . We define negative-scoring domains as non-domains and, as such, only domains with $q > 0$ in the max term in (3) are retained.

Our quality function q is:

$$q(k, l, \gamma) = s(k, l, \gamma) - \mu_s(l - k), \text{ where} \tag{4}$$

$$s(k, l, \gamma) = \frac{\sum_{g=k}^l \sum_{h=g+1}^l A_{gh}}{(l - k)^\gamma} \tag{5}$$

is a *scaled density* of the subgraph induced by the interactions A_{gh} between genomic loci k and l . Equation (4) is the zero-centered sum of (5), which is the upper-triangular portion of the submatrix defined by the domain in the interval $[k, l]$ divided by the scaled length $(l - k)^\gamma$ of the domain. When $\gamma = 1$, the scaled density is the weighted subgraph density [8] for the subgraph induced by the fragments between k and l . When $\gamma = 2$, the scaled density is half the internal density of a graph cluster [16]. For larger values of γ , the length of a domain in the denominator is amplified, hence, smaller domains would produce larger objective values than bigger domains with similar interaction frequencies. $\mu_s(l - k)$ is the mean value of (5) over all sub-matrices of length $l - k$ along the diagonal of \mathbf{A} , and can it be pre-computed for a given \mathbf{A} . We disallow domains where there are fewer than 100 sub-matrices available to compute the mean. By doing this, we are only excluding domains of size larger than $n - 100$ fragments, which in practice means that we are disallowing domains that are hundreds of megabases long. Values for the numerator in (5) are also pre-computed using an efficient algorithm [5], resulting in an overall run-time of $O(n^2)$ to compute $\text{OPT}_1(n)$.

3.2 Obtaining a Consensus Set of Persistent Domains across Resolutions

For objective (2), we use the procedure in section 3.1 to construct a set $\mathcal{D} = \bigcup_{\gamma \in \Gamma} D_\gamma$. \mathcal{D} is a set of overlapping intervals or domains, each with a quality score

defined by its persistence p across resolutions. To extract a set of highly persistent, non-overlapping domains from \mathcal{D} , we reduce problem 2 to the weighted interval scheduling problem [11], where competing requests to reserve a resource in time are resolved by finding the highest-priority set of non-conflicting requests. To find a consensus set of domains, we map a request associated with an interval of time to a domain and its corresponding interval on the chromosome. The priority of a request maps to a domain's persistence p across length scales.

The algorithm to solve problem 2 is then:

$$\text{OPT}_2(j) = \max\{\text{OPT}_2(j-1), \text{OPT}_2(c(j)) + p(a_j, b_j, \Gamma)\} \quad (6)$$

where $\text{OPT}_2(j)$ is the optimal non-overlapping set of domains for the j th domain in a list of domains sorted by their endpoints ($\text{OPT}_2(0) = 0$), and $c(j)$ is the closest domain before j that does not overlap with j . The first and second terms in (6) correspond to either choosing or not choosing domain j respectively. We pre-compute a domain's persistence p as:

$$p(a_i, b_i, \Gamma) = \sum_{\gamma \in \Gamma} \delta_i \text{ where } \delta_i = \begin{cases} 1 & \text{if } [a_i, b_i] \in D_\gamma \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Equation (7) is therefore a count of how often domain i appears across all resolutions in Γ for domain sets identified by the method in section 3.1. It may be desirable to treat multiple highly overlapping, non-equivalent domains as a single domain, however, we conservatively identify exact repetitions of a domain across resolutions since this setting serves as a lower bound on the persistence of the domain. If $m = |\mathcal{D}|$, then pre-computing persistence takes $O(m|\Gamma|)$ time, and $c(j)$ is precomputed after sorting the intervals by their endpoints. The limiting factor when computing $\text{OPT}_2(m)$ is time to compute $c(j)$, which is $m \log m$. Thus, the overall algorithm runs in $O(m \log m + (n^2 + m)|\Gamma|)$ time taking into account an additional $O(n^2|\Gamma|)$ for computing \mathcal{D} .

4 Results

We used chromatin conformation capture data from Dixon et al. [4] for human fibroblast and mouse embryonic cells. The 3C contact matrices were already aggregated at fragment size 40kbp and were corrected for experimental bias according to [19]. We compared our multiscale domains and consensus sets against the domains generated by Dixon et al. for the corresponding cell type and species. For human fibroblast cells, we used CTCF binding sites from [10]. For mouse embryonic cell CTCF binding sites and chromatin modification marks, we used data by Shen et al. [17].

4.1 Ability to Identify Densely Interacting Domains across Scales

Multiresolution domains successfully capture high frequency interactions and leave interactions of lower mean frequency outside of the domains. We compute

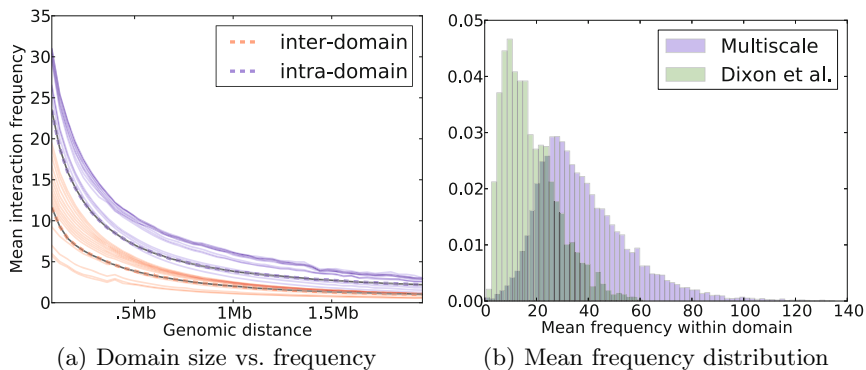


Fig. 2. (a) Our algorithm discovers domains with mean frequency value for inter- and intra-domain interactions (solid lines) at or better than that of Dixon et al. domains (dotted lines). Each solid line represents domains at different resolution γ in human fibroblast cells. (b) Multiscale domains identified in human fibroblast cells by our dynamic program tend to have higher mean frequency than those of Dixon et al. (distributions are plotted after outliers $> \mu + 4\sigma$ were removed).

the mean interaction frequency for all intra- and inter-domain interactions at various genomic lengths and plot the distribution of means for multiple resolutions (Fig. 2(a)). The mean intra-domain interaction frequency (blue) is consistently higher (up to two times) than the mean frequency for interactions that cross domains (red). Compared to the domains reported by Dixon et al., our domains tend to aggregate interactions of higher mean frequency, especially at larger γ . The distribution of mean intra-domain frequencies for Dixon et al. is skewed more to the left than that of the multiscale domains (Fig. 2(b)). This difference can be partially explained by the fact that multiscale domains on average are smaller in size ($\mu = 0.2\text{Mb}$, $\sigma = 1.2\text{Mb}$) than domains reported by Dixon et al. ($\mu = 1.2\text{Mb}$, $\sigma = 0.9\text{Mb}$).

4.2 Domain Persistence across Scales

Domain sets across resolutions share significant similarities, even as the distribution of domains and their sizes begin to change (Fig. 3). The patterns of similarity are particularly obvious if we plot the domains at various resolutions (Fig. 4(a)): many domains identified by our algorithm persist at several resolutions and are aggregated into larger domains at smaller γ , suggesting a hierarchical domain structure. The stability of these domains across resolutions indicates that the underlying chromosomal structure is dense within these domains and that these domains interact with the rest of the chromosome at a much lower frequency.

A pairwise comparison of domain configurations displays regions of stability across multiple resolutions (Fig. 4(b)). We use the variation of information (VI) [15], a metric for comparing two sets of clusters, to compute the distance

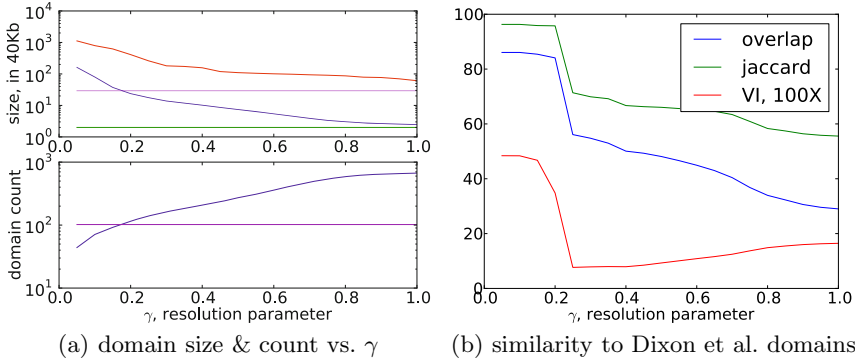


Fig. 3. (a) The domain sizes increase and the domain count decreases as the resolution parameter drops. Above: plotted are maximum (red), average (blue), and minimum (green) domain sizes averaged over all chromosomes for the domains on human fibroblasts. The magenta line shows the average domain size for domains reported by Dixon et al. Below: the number of domains increases for higher values of resolution parameter. The magenta line displays domain count for Dixon et al. (b) According to the Jaccard metric, the similarity between multiresolution domains and domains reported by Dixon et al. increases as the resolution parameter goes to zero.

between two sets of domains. To capture the similarities between two domain sets D and D' and the inter-domain regions induced by the domains, we construct new derivate sets C and C' where C contains all domains $d \in D$ as well as all inter-domain regions (C' is computed similarly). To compute entropy $H(C) = \sum_{c_i \in C} p_i \log p_i$, we define the probability of seeing each interval in C as $p_i = (b_i - a_i)/L$ where L is the number of nucleotides from the start of the leftmost domain to the end of the rightmost domain in the set $D \cup D'$. When computing the mutual information $I(C, C') = \sum_{c_i \in C} \sum_{c'_j \in C'} p_{ij} \log[p_{ij}/(p_i p_j)]$ between two sets of intervals C and C' , we define the joint probability p_{ij} to be $|[a_i, b_i] \cap [a_j, b_j]|/L$. We then compute variation of information on these two new sets: $VI(C, C') = H(C) + H(C') - 2I(C, C')$ where $H(\cdot)$ is entropy and $I(\cdot, \cdot)$ is mutual information. Chromosome 1, for example, has three visually pronounced groups of resolutions within which domain sets tend to be more similar than across ($\gamma = [0.00-0.20]$, $[0.25-0.70]$, and $[0.75-1.00]$ — see Fig. 4(b)).

4.3 Comparison with the Previously Identified Set of Domains in Dixon et al.

At higher resolutions, domains identified by our algorithm are smaller than those reported by Dixon et al. (Fig. 3(a)). As the resolution parameter decreases to 0.0, the average size of the domains increases (see Fig. 3 for results for chromosome 1 on the IMR90 human fibroblast cells). As domains expand to cover more and more of the chromosome, the similarity to the domains identified by Dixon et al. [4] also increases (Fig. 3(b)). We calculate the Jaccard similarity between

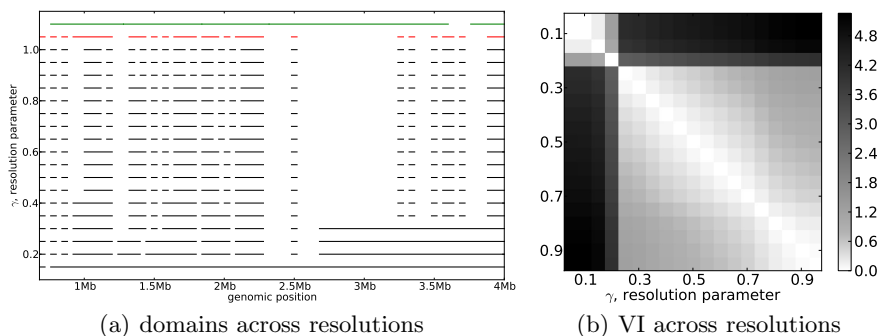


Fig. 4. (a) Domains identified by our algorithm (black) are smaller at higher resolutions and merge to form larger domains at γ close to 0. Visual inspection shows qualitative differences between consensus domains (red) and domains reported by Dixon et al. (green). Data shown for the first 4Mbp of chromosome 1 in human fibroblasts. (b) Variation of information for domains identified by our algorithm across different resolutions for chromosome 1 in human fibroblast cells.

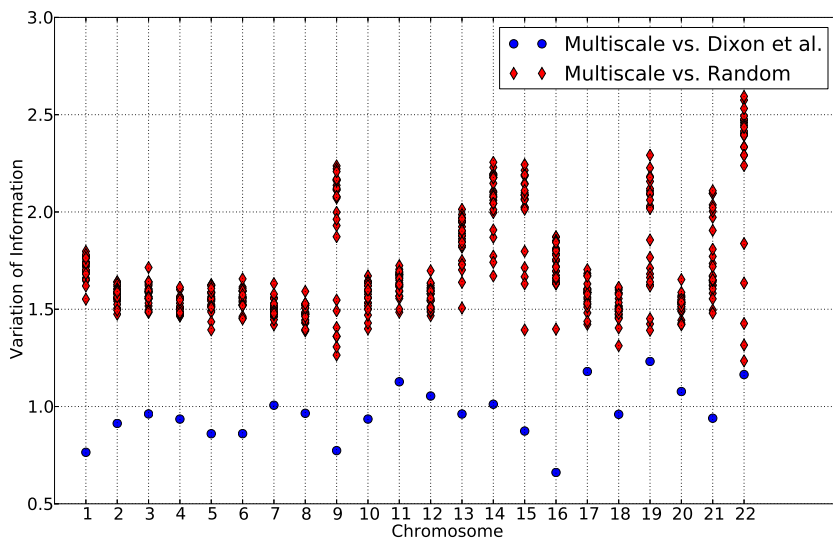


Fig. 5. Comparison of Dixon et al.’s domain set with the multiscale consensus set for chromosomes 1–22 (x -axis). We used the variation of information (VI) (y -axis) to compute distances between domain sets for the multiscale consensus set vs. Dixon et al (blue dots) and the multiscale consensus vs. randomly shuffled domains (red diamonds).

two sets of domains D and D' as $J(D, D') = N_{11}/(N_{11} + N_{01} + N_{10})$ where the quantities N_{11} , N_{01} , and N_{10} are the number of 3C fragments that are in a domain in both sets D and D' , the number of fragments that are in a domain

in D' , but not in D , and the number of fragments that are in a domain in D , but not D' , respectively (light blue in Fig. 3(b)). The composition of the domains, however, is different as is captured by the variation of information (red in Fig. 3(b)). Overall, we identify domains that cover similar regions of the chromosome (Fig. 2), yet differ in their size distribution and genomic positions.

We use the algorithm described in section 3.2 to obtain a consensus set of domains D_c persistent across resolutions. We construct the set Γ by defining the range of our scale parameter to be $[0, \gamma_{\max}]$ and incrementing γ in steps of 0.05. In order to more directly compare with previous results, we set $\gamma_{\max} = 0.5$ for human and 0.25 for mouse since these are the scales at which the maximum domain sizes in Dixon et al.'s sets match the maximum domain sizes in our sets.

Our consensus domain set agrees with the Dixon et al. domains better than with a randomized set of domains adhering to the same domain and non-domain length distributions (Fig. 5). Our primary motivation in comparing to randomized sets of domains is to provide a baseline that we can use to contrast our set of domains with Dixon et al. Comparing to a set of random domains also helps to verify that our observations are due to the observed sequence of domains and not the distribution of domain lengths. To shuffle Dixon's domains, we record the length of every domain and non-domain region, and then shuffle these lengths to obtain a randomized order of domains and non-domains across the chromosome. The fact that variation of information is lower between consensus domains and domains reported by Dixon et al. demonstrates that, though the approaches find substantially different sets of topological domains, they still agree significantly more than one would expect by chance.

4.4 Enrichment of CTCF and Histone Modifications Near Boundaries

We assess the enrichment of transcription factor CTCF and histone modifications H3K4me3 and H3K27AC within the inter-domain regions induced by the consensus domains. These enrichments provide evidence that the boundary regions between topological domains correlate with genomic regions that act as insulators and barriers, suggesting that the topological domains may play a role in controlling transcription in mammalian genomes [4].

Figure 6 illustrates the enrichment of insulator or barrier-like elements in domain boundaries in both the human fibroblast (IMR90) and mouse embryonic stem cell (mESC) lines. Specifically, we observe that the boundaries between consensus domains are significantly enriched for all of the transcription factors and histone marks we consider. In certain cases — specifically in the case of CTCF — we notice that the CTCF binding signals peak more sharply in the boundaries between the domains we discover than in the boundaries between the domains of Dixon et al.

We also observe that, when compared with the domain boundaries predicted by Dixon et al., our boundaries more often contain insulator or barrier-like elements (see Table 1). Specifically, we normalize for the fact that we identify approximately twice as many domains as Dixon et al., and generally observe a

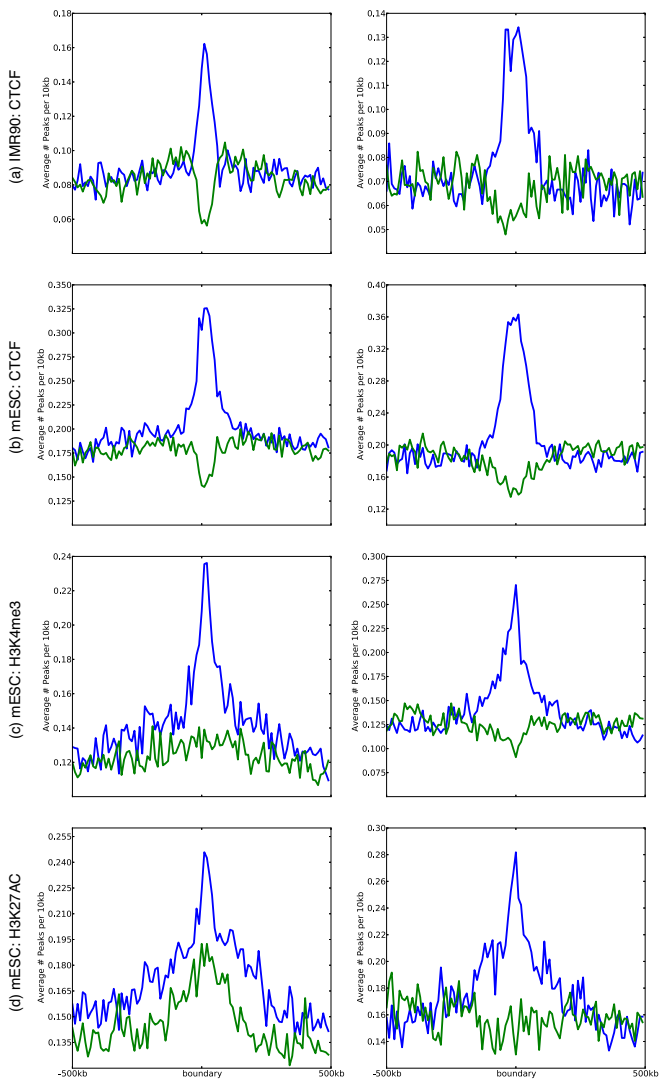


Fig. 6. Enrichment of binding CTCF binding (a) in IMR90 and (b) in mESC and histone modifications (c), (d) in mESC around domain boundaries for our consensus set of persistent domains (left, blue), and for those identified by Dixon et al. (right, blue). Green lines represent the presence of CTCF at the midpoint of the topological domains.

two-fold enrichment in the fraction of boundaries containing peaks for CTCF markers. This suggests that structural boundaries identified by our method are more closely tied to functional sites which serve as barriers to long-range regulation. We also observe a depletion of insulator CTCF elements within our domains when compared to the domains of Dixon et al. This observation is consistent with the assumption that transcriptional regulation is more active within spatially proximate domains since there are fewer elements blocking regulation within these domains. Table 1 also shows similar patterns for histone modifications which suggests that our domain boundaries are enriched for functional markers of gene regulation.

5 Discussion and Conclusions

In this paper, we introduce an algorithm to identify topological domains in chromatin using interaction matrices from recent high-throughput chromosome conformation capture experiments. Our algorithm produces domains that display much higher interaction frequencies within the domains than in-between domains (Fig. 2) and for which the boundaries between these domains exhibit substantial enrichment for several known insulator and barrier-like elements (Fig. 6). To identify these domains, we use a multiscale approach which finds domains at various size scales. We define a consensus set to be a set of domains that persist across multiple resolutions and give an efficient algorithm that finds such a set optimally.

The method for discovering topological domains that we have introduced is practical for existing datasets. our implementation is able to compute the domains for the human fibroblast cell line and extract the consensus set in under 40 minutes when run on a personal computer with 2.3GHz Intel Core i5 processor and 8Gb of RAM.

Our method is particularly appealing in that it requires only a single user-specified parameter γ_{\max} . It uses a score function that encodes the quality of putative domains in an intuitive manner based on their local density of interactions. Variations of the scoring function in (4), for example, by median centering

Table 1. Each table entry is of the form $\frac{e}{t} \approx r$ where e is the number of elements containing ≥ 1 of CTCF and histone modifications, t is the total number of elements and r is the approximate ratio e/t . Our method produces more domains, and hence more boundaries, than that of Dixon et al. [4]. However, relative to Dixon et al., our domains are depleted for peaks of interest, while our boundaries are significantly enriched for such peaks.

Signal	Domains ([4])	Domains (Ours)	Boundaries ([4])	Boundaries (Ours)
CTCF (IMR90)	$\frac{2050}{2234} \approx 0.92$	$\frac{3092}{5365} \approx 0.58$	$\frac{423}{2136} \approx 0.20$	$\frac{2126}{4861} \approx 0.44$
CTCF (mESC)	$\frac{2057}{2066} \approx 1.00$	$\frac{2500}{3578} \approx 0.70$	$\frac{654}{2006} \approx 0.33$	$\frac{2258}{3122} \approx 0.72$
H3K4me3 (mESC)	$\frac{2019}{2066} \approx 0.98$	$\frac{2362}{3578} \approx 0.66$	$\frac{600}{2006} \approx 0.30$	$\frac{1738}{3122} \approx 0.60$
H3K27AC (mESC)	$\frac{1922}{2066} \approx 0.93$	$\frac{2254}{3578} \approx 0.63$	$\frac{458}{2006} \approx 0.23$	$\frac{1342}{3122} \approx 0.43$

rather than mean centering, can be explored to test the robustness of the enrichments described here. For our experiments, the parameter γ_{\max} was set based on the maximum domain sizes observed in Dixon et. al's experiments so that we could easily compare our domains to theirs. This parameter can also be set intrinsically from properties of the Hi-C interaction matrices. For example, we observe similar enrichments in both human and mouse when we set γ_{\max} to be the smallest $\gamma \in I$ such that the median domain size is $>80\text{kbp}$ (two consecutive Hi-C fragments at a resolution of 40kbp). This is a reasonable assumption since domains consisting of just one or two fragments do not capture higher-order spatial relationships (e.g. triad closure) and interaction frequencies between adjacent fragments are likely large by chance [13]. We also compared the fraction of the genome covered by domains identified by Dixon et al. vs. the domains obtained from our method at various resolutions. Dixon et al.'s domains cover 85% of the genome while our sets tend to cover less of the genome ($\approx 65\%$ for a resolution which results in the same number of domains as those of Dixon et al.). The fact that our domain boundaries are more enriched for CTCF sites indicates that our smaller, denser domains may be more desirable from the perspective of genome function.

The dense, functionally-enriched domains discovered by our algorithm provide strong evidence that alternative chromatin domains exist and that a single length scale is insufficient to capture the hierarchical and overlapping domain structure visible in heat maps of 3C interaction matrices. Our method explicitly incorporates the desirable properties of domain density and persistence across scales into objectives that maximize each and uncovers a new view of domain organization in mammalian genomes that warrants further investigation.

Acknowledgments. This work has been partially funded by National Science Foundation (CCF-1256087, CCF-1053918, and EF-0849899) and National Institutes of Health (1R21AI085376). C.K. received support as an Alfred P. Sloan Research Fellow. D.F. is a predoctoral trainee supported by NIH T32 training grant T32 EB009403 as part of the HHMI-NIBIB Interfaces Initiative.

References

1. Bickmore, W.A., van Steensel, B.: Genome Architecture: domain organization of interphase chromosomes. *Cell* 152(6), 1270–1284 (2013)
2. Cavalli, G., Misteli, T.: Functional implications of genome topology. *Nature Structural & Molecular Biology* 20(3), 290–299 (2013)
3. de Wit, E., de Laat, W.: A decade of 3C technologies: insights into nuclear organization.. *Genes & Development* 26(1), 11–24 (2012)
4. Dixon, J.R., Selvaraj, S., Yue, F., Kim, A., Li, Y., Shen, Y., Hu, M., Liu, J.S., Ren, B.: Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature* 485(7398), 376–380 (2012)
5. Filippova, D., Gadani, A., Kingsford, C.: Coral: an integrated suite of visualizations for comparing clusterings. *BMC Bioinformatics* 13(1), 276 (2012)

6. Fudenberg, G., Getz, G., Meyerson, M., Mirny, L.A.: High order chromatin architecture shapes the landscape of chromosomal alterations in cancer. *Nature Biotechnology* 29(12), 1109–1113 (2011)
7. Gibcus, J.H., Dekker, J.: The hierarchy of the 3D genome. *Molecular Cell* 49(5), 773–782 (2013)
8. Goldberg, A.V.: Finding a maximum density subgraph. Computer Science Division. University of California, Berkeley (1984)
9. Hou, C., Li, L., Qin, Z.S., Corces, V.G.: Gene density, transcription, and insulators contribute to the partition of the *Drosophila* genome into physical domains. *Molecular cell* 48(3), 471–484 (2012)
10. Kim, T.H., Abdullaev, Z.K., Smith, A.D., Ching, K.A., Loukinov, D.I., Green, R.D., Zhang, M.Q., Lobanenko, V.V., Ren, B.: Analysis of the vertebrate insulator protein CTCF-binding sites in the human genome. *Cell* 128(6), 1231–1245 (2007)
11. Kleinberg, J., Tardos, É.: *Algorithm Design*. Addison-Wesley, Boston (2005)
12. Kölbl, A.C., Weigl, D., Mulaw, M., Thormeyer, T., Bohlander, S.K., Cremer, T., Dietzel, S.: The radial nuclear positioning of genes correlates with features of megabase-sized chromatin domains. *Chromosome Research* 20(6), 735–752 (2012)
13. Lieberman-Aiden, E., van Berkum, N.L., Williams, L., Imakaev, M., Ragoczy, T., Telling, A., Amit, I., Lajoie, B.R., Sabo, P.J., Dorschner, M.O., et al.: Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science* 326(5950), 289–293 (2009)
14. Lin, Y.C., Benner, C., Mansson, R., Heinz, S., Miyazaki, K., Miyazaki, M., Chandra, V., Bossen, C., Glass, C.K., Murre, C.: Global changes in the nuclear positioning of genes and intra- and interdomain genomic interactions that orchestrate B cell fate. *Nature Immunology* 13(12), 1196–1204 (2012)
15. Meilă, M.: Comparing clusterings by the variation of information. *Learning Theory and Kernel Machines* 2777, 173–187 (2003)
16. Schaeffer, S.E.: Graph clustering. *Computer Science Review* 1(1), 27–64 (2007)
17. Shen, Y., Yue, F., McCleary, D.F., Ye, Z., Edsall, L., Kuan, S., Wagner, U., Dixon, J., Lee, L., Lobanenko, V.V., Ren, B.: A map of the *cis*-regulatory sequences in the mouse genome. *Nature* 488, 116–120 (2012)
18. Tanay, A., Cavalli, G.: Chromosomal domains: epigenetic contexts and functional implications of genomic compartmentalization. *Current Opinion in Genetics & Development* 23(2), 197–203 (2013)
19. Yaffe, E., Tanay, A.: Probabilistic modeling of Hi-C contact maps eliminates systematic biases to characterize global chromosomal architecture. *Nature Genetics* 43(11), 1059–1065 (2011)

Modeling Intratumor Gene Copy Number Heterogeneity Using Fluorescence *in Situ* Hybridization Data*

Charalampos E. Tsourakakis

Carnegie Mellon University, USA
ctsourak@math.cmu.edu

Abstract. Tumorigenesis is an evolutionary process which involves a significant number of genomic rearrangements typically coupled with changes in the gene copy number profiles of numerous cells. Fluorescence *in situ* hybridization (FISH) is a cytogenetic technique which allows counting copy numbers of genes in single cells. The study of cancer progression using FISH data has received considerably less attention compared to other types of cancer datasets.

In this work we focus on inferring likely tumor progression pathways using publicly available FISH data. We model the evolutionary process as a Markov chain in the positive integer cone \mathbb{Z}_+^g where g is the number of genes examined with FISH. Compared to existing work which oversimplifies reality by assuming independence of copy number changes [24,25], our model is able to capture dependencies. We model the probability distribution of a dataset with hierarchical log-linear models, a popular probabilistic model of count data. Our choice provides an attractive trade-off between parsimony and good data fit. We prove a theorem of independent interest which provides necessary and sufficient conditions for reconstructing oncogenetic trees [8]. Using this theorem we are able to capitalize on the wealth of inter-tumor phylogenetic methods. We show how to produce tumor phylogenetic trees which capture the dynamics of cancer progression. We validate our proposed method on a breast tumor dataset.

Keywords: intra-tumor heterogeneity, evolutionary dynamics, cancer phylogenetics, Markov chains, simulation, FISH.

1 Introduction

Tumors are heterogeneous masses which exhibit cellular and genomic differences [13,20,21,22]. Cell-by-cell assay measurements allow us to study the phenomenon of tumor heterogeneity. Fluorescence *in situ* hybridization (FISH) is a cytogenetic technique which allows us to study *gene copy number heterogeneity* within a single tumor. It is used to count the copy number of DNA probes for specific genes or chromosomal regions. Understanding how tumor heterogeneity progresses is

* Topic: Cancer Genomics.

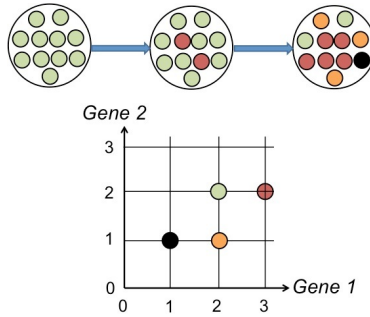


Fig. 1. Copy number genetic diversity within a tumor cell population after three hypothetical stages of tumorigenesis. The four observed states of cells are shown and coded with colors in the positive integer cone \mathbb{Z}_+^2 .

a major problem with significant potential impact on therapeutics. An example which illustrates the importance of heterogeneity in therapeutic resistance is found in chronic myelogenous leukemia (CML). The presence of a subpopulation of leukemic cells of a given type significantly influences the response to therapies based on imatinib mesylate, causing the eventual relapse of the disease [20].

In this work we study the phenomenon of *gene copy number heterogeneity* within a single tumor. An illustration of the phenomenon of gene copy number heterogeneity is shown in Figure 1 which shows a hypothetical cell population of eleven somatic human cells at three stages of tumorigenesis, whose succession is indicated by arrows. We shall describe the state of a cell as a two dimensional point (g_1, g_2) in the positive integer cone \mathbb{Z}_+^2 , with the obvious meaning that the cell has g_1, g_2 copies of gene 1 and gene 2 respectively. Initially, all cells are in the healthy diploid state with respect to their gene copy numbers. At the second stage, nine cells are in state $(2,2)$ and two cells in state $(3,2)$. At the last stage, only 2 cells are in the healthy state $(2,2)$. Five, three and one cell are in states $(3,2)$, $(2,1)$ and $(1,1)$ respectively. The four observed states are shown and coded with colors in \mathbb{Z}_+^2 . Figure 1 illustrates what we observe in many tumors: existence of multiple progression states within a single tumor.

Despite the large amount of research work on modeling tumor progression using different types of tumor datasets, e.g., [12], the study of FISH datasets has received considerably less attention. Specifically, two early studies of FISH datasets were limited to either two [24,25] or three probes [19]. Pennington et al. [24,25] develop novel computational methods for analyzing FISH data. Specifically, they consider a random walk on the positive integer cone \mathbb{Z}_+^2 where at each step a coordinate $i \in \{1, 2\}$ is picked uniformly at random and is modified by $\Delta x \in \{0, 1, -1\}$ with probabilities $1 - p_{i,1} - p_{i,-1}, p_{i,1}, p_{i,-1}$, $i = 1, 2$ respectively. Given this model they optimize a likelihood-based objective over all possible trees and parameters $\{p_{i,1}, p_{i,-1}\}$. Recently, Chowdhury et al. [5] proposed a general procedure which can treat any number of probes. They reduce the study of the progression of FISH probe cell count patterns to the rectilinear minimum spanning tree problem.

Paper contributions and roadmap. In this paper we achieve the following contributions:

- We introduce a novel approach to analyzing FISH datasets. The main features of our approach are its probabilistic nature which provides an attractive trade-off between parsimony and expressiveness of biological complexity and the reduction of the problem to the well-studied inter-tumor phylogeny inference problem. The former allows us to capture complex dependencies between factors while the latter opens the door to a wealth of available and established theoretical methods which exist for the inter-tumor phylogeny inference problem.
- We prove Theorem 1 which provides necessary and sufficient conditions for the unique reconstruction of an oncogenetic tree [8]. Based on the theorem’s conditions, we are able to capitalize on the wealth of inter-tumor phylogenetic methods. However, the result is of independent interest and introduces a set of interesting combinatorial questions.
- We validate our proposed method on a publicly available breast cancer dataset.

The outline of this paper is as follows: Section 2 presents our proposed methods. Section 3 performs an experimental evaluation of our methods on a breast cancer FISH dataset and an extensive biological analysis of the findings. Finally, Section 4 concludes the paper by a discussion and a brief summary.

2 Proposed Method

In Section 2.1 we model the probability distribution of FISH data with hierarchical log-linear models and show how to learn the parameters of the model for a given FISH dataset. In Section 2.2 we prove Theorem 1 which provides necessary and sufficient conditions to uniquely reconstruct an oncogenetic tree [8]. We capitalize on the theorem to harness the wealth of available methods for inter-tumor phylogenetic inference methods [1,2,3,8,9,12,11,14]. Finally in Section 2.3 we present our proposed method.

We will make the same simplifying assumptions with existing work [24,5], namely that only single gene duplication and loss events take place and that the cell population is fixed. In what follows, let $\mathcal{D} = \{x_1, \dots, x_n\}, x_i \in \mathbb{Z}_+^g$ be the input FISH dataset which measures the copy numbers of g genes in n cells taken from the same tumor.

2.1 Model and Fitting

Probabilistic Model: Let X_j be an integer-valued random variable which expresses the copies of the j -th gene with domain $I_j, j = 1, \dots, g$. We model the joint probability distribution of the random vector (X_1, \dots, X_g) as

$$\Pr [x] = \frac{1}{Z} \prod_{A \subseteq [g]} e^{\phi_A(x)} \quad (1)$$

where $x = (x_1, \dots, x_g) \in I = I_1 \times I_2 \times \dots \times I_g$ is a point of the integer positive cone \mathbb{Z}_+^g and Z is a normalizing constant, also known as the partition function, which ensures that the distribution is a proper probability distribution, i.e., $Z = \sum_{x \in I} \prod_{A \subseteq [g]} e^{\phi_A(x)}$. Each potential function ϕ_A depends only on the variables in the subset A and is parameterized by a set of weights w_A . To illustrate this, assume $g = 2$ and $I = \{0, 1\} \times \{0, 1\}$. Then,

$$\begin{aligned} \log \mathbf{Pr} [x] &= w_0 + w_{(1)0} \mathbb{1}\{x_1 = 0\} + w_{(1)1} \mathbb{1}\{x_1 = 1\} + w_{(2)0} \mathbb{1}\{x_2 = 0\} \\ &\quad + w_{(2)1} \mathbb{1}\{x_2 = 1\} + w_{(12)00} \mathbb{1}\{x_1 = 0, x_2 = 0\} + w_{(12)01} \mathbb{1}\{x_1 = 0, x_2 = 1\} \\ &\quad + w_{(12)10} \mathbb{1}\{x_1 = 1, x_2 = 0\} + w_{(12)11} \mathbb{1}\{x_1 = 1, x_2 = 1\} - \log Z, \end{aligned}$$

where w_{Ax} $A \subseteq \{1, 2\}$, $x \in \{0, 1\}^{|A|}$ are the parameters of the model. This probability distribution captures the effects of different factors through parameters w_{Ax} , $A \in \{\{1\}, \{2\}\}$, $x \in \{0, 1\}$ and pairwise interactions through parameters $w_{(12)x}$, $x \in \{00, 01, 10, 11\}$. In general, two variables X_i, X_j are defined to be directly associated if there exists at least one non-zero (or bounded away significantly from zero) parameter including the two variables. We define X_i, X_j to be indirectly associated if there exists a chain of overlapping direct associations that relate X_i, X_j .

We impose the following restriction on the probabilistic model shown in equation (1): If $A \subseteq B$ and $w_A = 0$ then $w_B = 0$. This restriction reduces significantly the size of the parameter space, but allows to express complex dependencies not captured by existing work [24,25]. Since a typical FISH dataset contains detailed measurements for a handful of genes from few hundred cells the combination of these two features is crucial to avoid overfitting and obtain biological insights at the same time. Furthermore, it is worth emphasizing that in terms of biological interpretation the assumption is natural: if a set A of genes does not interact, then any superset of A maintains that property. This class of models are known as hierarchical log-linear models [4].

Learning the Parameters: Learning the parameters w of a hierarchical log-linear model is a well-studied problem, e.g., [4]. An extensive survey of learning methods can be found in [26]. Schmidt et al. [27] propose to maximize a penalized log-likelihood of the dataset \mathcal{D} where the penalty is an overlapping group l_1 -regularization term. Specifically, a spectral projected gradient method is proposed as a sub-routine for solving the following regularization problem:

$$\min_w - \sum_{i=1}^n \log \mathbf{Pr} [x_i | w] + \sum_{A \subseteq S} \lambda_A \|w_A\|_1. \quad (2)$$

2.2 Unique Reconstruction of Oncogenetic Trees

Our main theoretical result in this section is motivated by the following natural sequence of questions:

Can we use any of the existing inter-tumor progression methods [12] on the intra-tumor progression problem? How will the resulting tree capture the evolutionary dynamics of cancer progression, i.e., how do we enforce that state (\dots, i, \dots) is reached either through $(\dots, i + 1, \dots)$ or $(\dots, i - 1, \dots)$ given our single gene duplication and loss event assumption?

An answer to this question is given in Section 2.3. Motivated by our intention to capitalize on inter-tumor phylogenetic methods such as [8,9], we consider a fundamental problem concerning oncogenetic trees [8]. What are the necessary and sufficient conditions to reconstruct them? Theorem 1 is likely to be of independent interest and contributes to the understanding of oncogenetic trees [8]. We briefly review the necessary definitions to state our result. Let $T = (V, E, r)$ be an oncogenetic tree, i.e., a rooted branching¹, on V and let $r \in V$ be the root of T . Given a finite family $\mathcal{F} = \{A_1, \dots, A_q\}$ of sets of vertices, i.e., $A_i \subseteq V(T)$ for $i = 1, \dots, q$, where each A_i is the vertex set of a rooted sub-branching of T , what are the necessary and sufficient conditions, if any, which allow us to uniquely reconstruct T ?

Theorem 1. *The necessary and sufficient conditions to uniquely reconstruct the branching T from the family \mathcal{F} are the following:*

1. For any two distinct vertices $x, y \in V(T)$ such that $(x, y) \in E(T)$, there exists a set $A_i \in \mathcal{F}$ such that $x \in A_i$ and $y \notin A_i$.
2. For any two distinct vertices $x, y \in V(T)$ such that $y \not\prec x$ and $x \not\prec y$ there exist sets $A_i, A_j \in \mathcal{F}$ such that $x \in A_i, y \notin A_i$ and $x \notin A_j$ and $y \in A_j$.

Proof. First we prove the necessity of conditions 1,2 and then their sufficiency to reconstruct T . In the following we shall call a branching T consistent with the family set \mathcal{F} if all sets $A_i \in \mathcal{F}$ are vertices of rooted sub-branchings of T .

Necessity: For the sake of contradiction, assume that Condition 1 does not hold. Then, the two branchings shown in Figure 2(a) are both consistent with \mathcal{F} and therefore we cannot reconstruct T . Similarly, assume that Condition 2 does not hold. Specifically assume that for all j such that $x \in A_j$, then $y \in A_j$ too (for the symmetric case the same argument holds). Then, both branchings in Figure 2(b) are consistent with \mathcal{F} and therefore T is not reconstructable.

Sufficiency: Let $x \in V(T)$ and P_x be the path from the root to x , i.e., $P_x = \{r, \dots, x\}$. Also, define F_x to be the intersection of all sets in the family \mathcal{F} that contain vertex x , i.e., $F_x = \bigcap_{x \in A_i \in \mathcal{F}} A_i$. We prove that $F_x = P_x$. Since by

definition, $P_x \subseteq F_x$ we need to show that $F_x \subseteq P_x$. Assume that the latter does not hold. Then, there exists a vertex $v \in V(T)$ such that $v \notin P_x, v \in F_x$. We consider the following three cases.

- CASE 1 ($x \prec v$): Since every $A_i \in \mathcal{F}$ is the vertex set of a rooted sub-branching, $v \in P_x$ by definition.
- CASE 2 ($v \prec x$): By condition 1 and an easy inductive argument, there exists A_i such that $x \in A_i, v \notin A_i$. Therefore, $v \notin F_x$.

¹ Each vertex has in-degree at most one and there are no cycles.
² We use the notation $u \prec v$ ($u \not\prec v$) to denote that u is (not) a descendant of v in T .

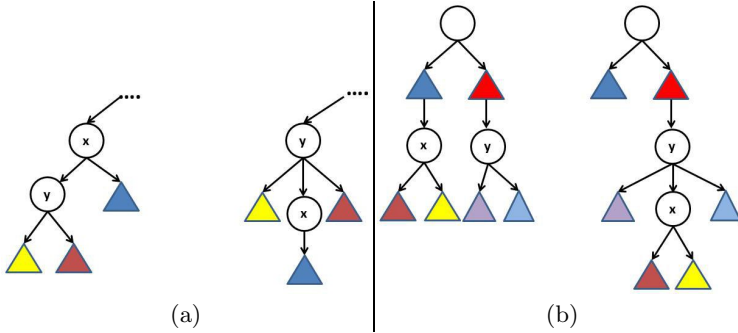


Fig. 2. Illustration of necessity conditions of Theorem 1, (a) condition 1 (b) condition 2

• CASE 3 ($x \not\prec v, v \not\prec x$): By condition 2, there exists A_i such that $x \in A_i$ and $v \notin A_i$. Therefore, in combination with the definition of F_x we obtain $v \notin F_x$.

In all three cases, we obtain a contradiction and therefore $v \in F_x \Rightarrow v \in P_x$, showing that $F_x = P_x$. Given this fact, it is easy to reconstruct the branching T . We sketch the algorithm: compute for each x the set F_x and from F_x reconstruct the ordered version of the path P_x , i.e., $(r \rightarrow v_1 \rightarrow \dots \rightarrow x)$ using sets in \mathcal{F} whose existence is guaranteed by condition 1. \square

A natural question is whether one can extend Theorem 1 to more complex classes of oncogenetic models, such as directed acyclic graphs (DAGs) [12]. The answer is negative. For instance, the oncogenetic tree with edges $1 \rightarrow 2, 1 \rightarrow 3$ and the DAG with edges $1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 3$ are indistinguishable. Another example is shown in Figure 3.

Oncogenetic tree reconstruction algorithms [8] and more generally established inter-tumor cancer progression methods [12] receive as input a family of sets, where each set represents the set of mutations observed in a *single* tumor. Recall, that in the case of intra-tumor cancer progression the typical input is a multiset of points in the positive integer cone where each point is the copy-gene number state of a given cell. For instance, for the tumor shown in the final step of tumorigenesis in Figure 1, the input would be $\mathcal{D} = \{(2, 2) \times 2, (3, 2) \times 5, (2, 1) \times 3, (1, 1) \times 1\}$.

Based on the insights from the proof of Theorem 1, we convert a FISH dataset to a dataset suitable for inter-tumor cancer progression inference. Specifically, we assume we are given a FISH dataset and an algorithm f which infers an evolutionary model of cancer progression from several tumors. Notice that f could be any inter-tumor phylogenetic method, see [12].

For each cell in state $x = (x_1, \dots, x_g)$ we generate a family of sets of “mutations” as follows: for every gene $i \in [g]$ whose number of copies x_i is greater than 2 we generate $x_i - 1$ sets in order to enforce that the gene has $c + 1$ copies if at a previous stage had c copies, $c = 2, \dots, x_i - 1$. For instance if gene i has 4 copies, we generate the sets $\{gene - i - mut - 2\}$, $\{gene - i - mut - 2, gene - i - mut - 3\}$, $\{gene - i - mut - 2, gene - i - mut - 3, gene - i - mut - 4\}$ which show how the gene obtained 2 extra copies. The case $x_i < 2$ is treated in a similar way. Finally,

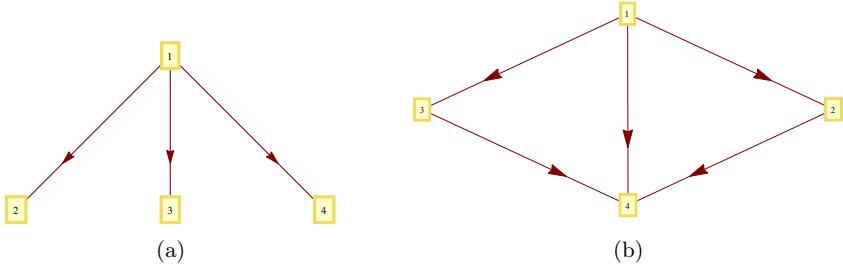


Fig. 3. Extending Theorem 1 to oncogenetic directed acyclic graphs (DAGs) [12] is not possible. For instance, the oncogenetic tree and the DAG are indistinguishable in terms of the generated families of sets.

we generate a set s_f for each cell which contains all mutations that led to state x . For instance, for the state $(0, 3)$ $s_f = \{gene - 1 - mut - 2, gene - 1 - mut - 1, gene - 1 - mut - 0, gene - 2 - mut - 2, gene - 2 - mut - 3\}$. Upon creating the dataset \mathcal{F} we use it as input to $f()$, an existing intra-tumor phylogenetic method, see [12].

Using the conversion above, based on condition 1 of Theorem 1 the output of an inter-tumor phylogenetic method will capture the dynamic nature of the process, which will be consistent with our assumptions of single gene duplication and loss events and Ockham’s razor, e.g., the evolutionary sequence $2 \rightarrow 3 \rightarrow 4$ rather than $2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 4$.

2.3 Progression Inference

Define $\mathcal{B} = [\min_{i \in [n]} x_{i1}, \max_{i \in [n]} x_{i1}] \times \dots \times [\min_{i \in [n]} x_{ig}, \max_{i \in [n]} x_{ig}]$ to be the minimum enclosing box of \mathcal{D} , where x_{ij} is the number of copies of gene j in the i -th cell, $i \in [n], j \in [g]$. Given the observed data we can calculate the empirical probability $\tilde{\pi}(s)$ of any state $s \in \mathcal{B}$ as the fraction $\frac{|\{q: q \in \mathcal{D}, q=s\}|}{n}$. The number of states in \mathcal{B} grows exponentially fast for any typical FISH dataset. We summarize parsimoniously this distribution as described in Section 2.1. Specifically, we learn the parameters w of the hierarchical log-linear model by maximizing the overlapping l_1 penalized log-likelihood of equation (2) as described in [26]. We allow only second-order interactions between factors. It is worth mentioning that k -way interactions, $k \geq 3$ can be embedded in the model as well, see Chapter 6 [26], but we prefer not to avoid overfitting. Alternatively we can allow higher order interactions but then a penalty term for the model complexity (e.g., AIC, BIC) should be taken into account. Let π be the distribution specified by the learned parameters. We define a Metropolis-Hastings chain with stationary distribution π [17]. Initially, all n cells will be in the diploid state $(2, \dots, 2)$. Notice that all we need to compute during the execution of the chain are ratios of the form $\pi(x)/\pi(y)$, which saves us from the computational cost of computing the normalization constant Z . We simulate the chain k times in order to draw

$m \geq 1$ samples from the probability distribution. Finally we use the conversion described in Section 2.2 to infer a tumor phylogeny.

There exists a subtle issue that arises in practice: there exist states of \mathcal{B} which are not observed in the dataset \mathcal{D} . We surpass this problem by adding one fictitious sample to each state $b \in \mathcal{B}$. From a Bayesian point of view this is equivalent to smoothing the data with an appropriately chosen Dirichlet prior.

To summarize, our proposed method consists of the following steps: (1) Given a FISH dataset \mathcal{D} we learn the parameters of a hierarchical log-linear model with pairwise potentials. (2) Given the learned parameters we can compute the probability distribution on Z^g . Let π be the resulting distribution. We define a Metropolis-Hastings chain with stationary distribution π . Initially cells are in the healthy diploid state $(2, \dots, 2)$. (3) Draw $m \geq 1$ samples from the probability distribution by running the Metropolis-Hastings chain simulation m times. (4) Convert the resulting FISH samples to inter-tumor phylogenetic datasets by following the procedure of Section 2.2. (5) Use an inter-tumor phylogenetic method [12] to infer a tumor phylogenetic tree.

Finally, an interesting perspective on our modeling which makes a conceptual connection to [5] is the following: upon learning the parameters of the hierarchical log-linear model, the probability distribution over \mathcal{B} assigns implicitly weights on the edges of the positive integer di-grid \mathbb{Z}^g (each undirected edge of the grid is substituted by two directed edges) according to the Metropolis-Hastings chain. Therefore, both our method and [5] assign weights to the edges of the positive integer grid. This perspective opens two natural research directions which we leave open for future research. First, instead of simulating the Markov chain, one proceed could find an appropriate subgraph of the weighted di-grid, e.g., a maximum weighted branching rooted at the diploid state. Secondly, it is natural to ask whether there exists a natural probabilistic interpretation of the method in [5].

3 Experimental Results

Experimental Setup: In this paper, we show the results of validating our method on a breast cancer dataset from a collection of publicly available FISH datasets which can be found at <ftp://ftp.ncbi.nlm.nih.gov/pub/FISHtrees/data/>.

We used the following third-party publicly available code in our implementations: hierarchical log-linear fitting code³ [26], distance based oncogenetic trees [9], FISH progression trees [5] and *graphviz* for visualization purposes. Our routines are implemented in MATLAB. The number of simulations was set to $m = 10$. We experimented both with smaller values for parameter m ($m \geq 2$) and the choice of log-linear fitting method, see [6], and we found that our results are robust.

Table 2 provides a short description of the six genes that are analyzed in our trees. The breast tumor dataset consists of 187 points in \mathbb{Z}_+^6 .

³ Schmidt's code does not scale well to more than 6-7 variables.

Table 1. Genes are shown in the first column and their cytogenetic positions in the second. The third column describes whether a gene is an oncogene or a tumor suppressor gene.

Gene	Cytogenetic Band	Description
<i>cox-2</i>	1q25.2-q25.3	oncogene
<i>myc</i>	8q24	oncogene
<i>ccnd1</i>	11q13	oncogene
<i>cdh1</i>	16q22.1	tumor suppressor gene
<i>p53</i>	17p13.1	tumor suppressor gene
<i>znf217</i>	20q13.2	oncogene

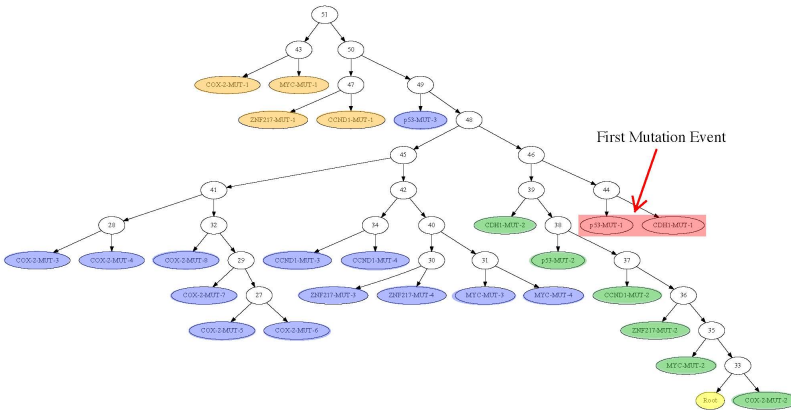


Fig. 4. (best viewed on screen) Cancer phylogenetic tree for a breast cancer tumor obtained by our method. Leaves of the tree are colored in the following way: the root-event leaf is in yellow; the first change in the copy number profile is in red color; euploid states are in green; states with copy number gain/loss are shown in blue/orange. The first changes are losses of one gene copy of genes *p53* and *cdh1*.

Results and Analysis: Figures 4 and 5 show the cancer phylogenetic trees of a ductal carcinoma *in situ* (DCIS) obtained by our method and [5] respectively. Our tree is a distance based phylogenetic tree produced by using our reduction of intra-tumor phylogenetic inference to inter-tumor phylogenetic inference as described in Section 2.3. We observe that the tree of [5] does not explain the 27 different states that appear in the dataset. For instance the state (4, 8, 4, 4, 2, 2, 4) accounts for 0.0053% of the appearing states and is discarded by [5] but is taken into account by our method. Since there is no ground truth available to us it is hard to reach any indisputable conclusion. However, we found that our findings are strongly supported by oncogenetic literature.

The mutational events captured in our phylogenetic tree highlight putative sequential events during progression from ductal carcinoma *in situ* (DCIS) to invasive breast carcinoma. The first mutational events are highlighted in red.

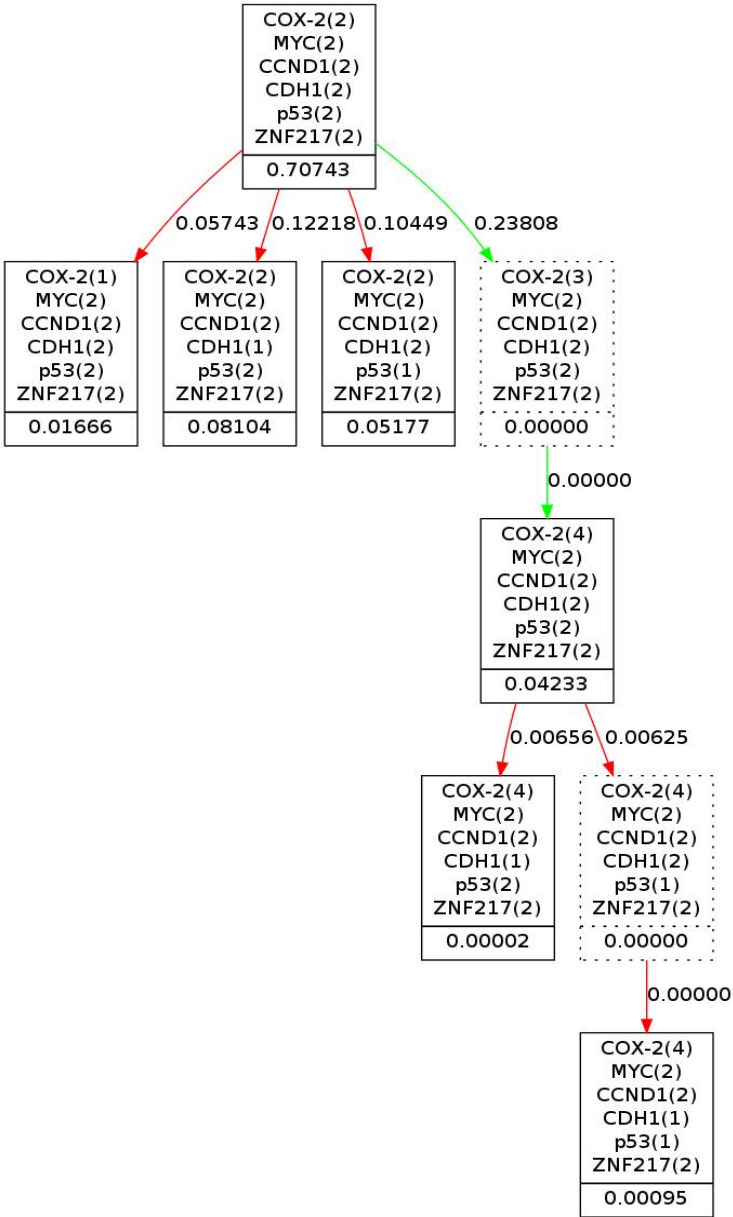


Fig. 5. Cancer phylogenetic tree for a breast cancer tumor obtained by [5]. Nodes with dotted borders represent Steiner nodes, i.e., states that do not appear in the dataset. Green and red edges model gene gain and loss respectively. The weight value on each edge does not have the semantics of probability, but it is the rectilinear distance between the two connected states. See [5] for further details. The weight on each node describes the fraction of cells in the FISH dataset with the particular copy number profile.

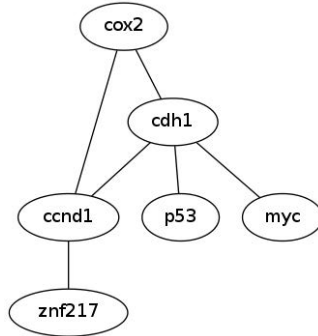


Fig. 6. Direct associations among genes inferred from fitting a hierarchical log-linear model to a ductal carcinoma *in situ* FISH dataset

Initially one allele of *p53* and *cdh1* are lost. Concurrent loss of *cdh1* function and *p53* inactivation act synergistically in the formation, progression and metastasis of breast cancer [7]. Moreover, following the first mutational events, the next changes occur in *ccnd1*, *myc* and *znf217* which are oncogenes participating in cell cycle regulation, proliferation and cancer progression. Specifically, as shown by single invasive ductal carcinoma (IDC) cell analysis copy number loss of *cdh1* is common in DCIS. Furthermore, copy number gains of *myc* are a common feature in the transition from DCIS to IDC [13]. This is consistent with our results. The synergy between *p53* and *cdh1* appears also in the tree of Figure 5 but at the last stages of the progression. Finally, Figure 6 shows the inferred direct associations among genes.

4 Conclusion

In this work we develop a novel approach to studying FISH datasets, a type of dataset which has received considerably less attention to other types of cancer datasets. Compared to prior work we take a probabilistic approach which provides good data fit, avoids overfitting and captures complex dependencies among factors. Motivated by our intention to capitalize on inter-tumor phylogenetic methods we prove a theorem which provides necessary and sufficient conditions for reconstructing oncogenetic trees [8]. Using these conditions, we show one way to perform intra-tumor phylogenetic inference by opening the door to the wealth of established inter-tumor phylogenetic techniques [12]. We model the evolutionary dynamics as a Markov chain in the positive integer cone \mathbb{Z}_+^g where g is the number of genes examined with FISH. Finally, we validate our approach to a breast cancer FISH dataset.

Our work leaves numerous problems open for future research. Improved models need to be developed that remove the simplifying assumption of a fixed cell population and take the clonal evolution model into account [23], namely cancer is initiated once multiple mutations occur in a random single cell which gives

birth to the uncontrolled proliferation of cancerous cells. Secondly, clustering patients and finding consensus FISH progression trees per cluster is another interesting problem. Furthermore, we plan to experiment with (a) other choices of inter-tumor phylogenetic methods and (b) fitting approaches that allow higher order interactions but will also account for the increased complexity of the resulting model. Finally, using features from our inferred trees as features for classification is an interesting question, see [5].

Acknowledgments. The author would like to thank Prof. Russell Schwartz and Maria Tsiarli for helpful discussions on intratumor heterogeneity and the anonymous reviewers for their constructive feedback.

References

1. Beerenwinkel, N., Eriksson, N., Strumfels, B.: Conjunctive bayesian networks. *Bernoulli* 13, 893–909 (2007)
2. Beerenwinkel, N., Rahnenführer, J., Däumer, M., Hoffmann, D., Kaiser, R., Selbig, J., Lengauer, T.: Learning multiple evolutionary pathways from cross-sectional data. *Journal of Computational Biology* 12, 584–598 (2005)
3. Beerenwinkel, N., Sullivant, S.: Markov models for accumulating mutations. *Biometrika* 96, 663–676 (2009)
4. Bishop, Fienberg, S., Holland, P.: *Discrete Multivariate Analysis*. MIT Press (1975)
5. Chowdhury, S.A., et al.: FISHtrees: Modeling tumor progression from fluorescence in situ hybridization (FISH) data from many single cells of solid tumors and their metastases. In: *ISMB 2013* (2013)
6. Dellaportas, P., Forster, J.: Markov Chain Monte Carlo Model Determination for Hierarchical and Graphical Log-linear Models. *Biometrika* (1996)
7. Derksen, P., Liu, X., Saridin, F., et al.: Somatic inactivation of E-cadherin and p53 in mice leads to metastatic lobular mammary carcinoma through induction of anoikis resistance and angiogenesis. *Cancer Cell* 10(5), 437–449 (2006)
8. Desper, R., et al.: Inferring tree models for oncogenesis from comparative genome hybridization data. *Journal of Computational Biology* 6(1), 37–51 (1999)
9. Desper, R., et al.: Distance-based reconstruction of tree models for oncogenesis. *J. Comput. Biol.* 7(6), 789–803 (2000)
10. Gasco, M., Shami, S., Crook, T.: The p53 pathway in breast cancer. *Breast Cancer Res.* 54(2), 70–76 (2002)
11. Gerstung, M., Baudis, M., Moch, H., Beerenwinkel, N.: Quantifying cancer progression with conjunctive bayesian networks. *Bioinformatics* 25, 2809–2815 (2009)
12. Hainke, K., Rahnenführer, J., Fried, R.: Disease progression models: A review and comparison. Dortmund University, Technical Report, <http://tinyurl.com/ceyr9wx>
13. Heselmeyer-Haddad, K., et al.: Single-cell genetic analysis of ductal carcinoma in situ and invasive breast cancer reveals enormous tumor heterogeneity yet conserved genomic imbalances and gain of MYC during progression. *Journal of American Pathology* 181(5), 1807–1822 (2012)
14. Heydebreck, A., Gunawan, B., Füzesi, L.: Maximum likelihood estimation of oncogenetic tree models. *Biostatistics* 5(4), 545–556 (2004)

15. Krig, S.R., Miller, J.K., Frieze, S., et al.: ZNF217, a candidate breast cancer oncogene amplified at 20q13, regulates expression of the ErbB3 receptor tyrosine kinase in breast cancer cells. *Oncogene* 29(40), 5500–5510 (2010)
16. Letouzé, E., Allory, E., Bollet, M., et al.: Analysis of the copy number profiles of several tumor samples from the same patient reveals the successive steps in tumorigenesis. *Genome Biology* 11, R76 (2010)
17. Levin, D., Peres, Y., Wilmer, E.: *Markov Chains and Mixing Times*. American Mathematical Society (2008)
18. Lin, S., Xia, W., Wang, J., et al.: β -catenin, a novel prognostic marker for breast cancer: its roles in cyclin D1 expression and cancer progression. *Proc. Natl. Acad. Sci.* 97(8), 4262–4266 (2000)
19. Martins, F., et al.: Evolutionary pathways in BRCA1-associated breast tumors. *Cancer Discovery* 2(6), 503–511 (2012)
20. Marusyk, A., Polyak, K.: Tumor heterogeneity: causes and consequences. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer* 1805(1), 105–117 (2010)
21. Navin, N., Krasnitz, A., Rodgers, L., et al.: Inferring tumor progression from genomic heterogeneity. *Genome Res.* 20, 68–80 (2010)
22. Navin, N., et al.: Tumour evolution inferred by single-cell sequencing. *Nature* 472, 90–94 (2011)
23. Nowell, P.C.: The clonal evolution of tumor cell populations. *Science* 194, 23–28 (1976)
24. Pennington, G., Smith, C., Shackney, S., Schwartz, R.: Reconstructing tumor phylogenies from heterogeneous single-cell data. *Journal of Bioinformatics and Computational Biology* 5(02A), 407–427 (2007)
25. Pennington, G., Shackney, S., Schwartz, R.: *Cancer Phylogenetics from Single-Cell Assays*. Technical Report CMU-CS-06-103, <http://tinyurl.com/bvjlgch>
26. Schmidt, M.: *Graphical Model Structure Learning with l_1 -Regularization*. Ph.D. Thesis, University of British Columbia (2010)
27. Schmidt, M., Murphy, K.: *Convex Structure Learning in Log-Linear Models: Beyond Pairwise Potentials*. AISTATS (2010)

Phylogenetic Analysis of Cell Types Using Histone Modifications

Nishanth Ulhas Nair^{1,*}, Yu Lin¹, Philipp Bucher^{2,**}, and Bernard M.E. Moret^{1,**}

¹ School of Computer and Communication Sciences

² School of Life Sciences

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

{philipp.bucher, bernard.moret}@epfl.ch

Abstract. In cell differentiation, a cell of a less specialized type becomes one of a more specialized type, even though all cells have the same genome. Transcription factors and epigenetic marks like histone modifications can play a significant role in the differentiation process. In this paper, we present a simple analysis of cell types and differentiation paths using phylogenetic inference based on ChIP-Seq histone modification data. We propose new data representation techniques and new distance measures for ChIP-Seq data and use these together with standard phylogenetic inference methods to build biologically meaningful trees that indicate how diverse types of cells are related. We demonstrate our approach on H3K4me3 and H3K27me3 data for 37 and 13 types of cells respectively, using the dataset to explore various issues surrounding replicate data, variability between cells of the same type, and robustness. The promising results we obtain point the way to a new approach to the study of cell differentiation.

Keywords: cell differentiation, cell type, epigenomics, histone modifications, phylogenetics.

1 Introduction and Background

In developmental biology, the process by which a less specialized cell becomes a more specialized cell type is called cell differentiation. Since all cells in one individual organism have the same genome, epigenetic factors and transcriptional factors play an important role in cell differentiation [8–10]. Thus a study of epigenetic changes among different cell types is necessary to understand cell development.

Histone modifications form one important class of epigenetic marks; such modifications have been found to vary across various cell types and to play a role in gene regulation [3]. Histones are proteins that package DNA into structural units called nucleosomes [14]. These histones are subject to various types of modifications (methylation, acetylation, phosphorylation, and ubiquitination), modifications that alter their interaction with DNA and nuclear proteins. In turn, changes in these interactions influence gene transcription and genomic function. In the last several years a high-throughput, low-cost, sequencing technology called ChIP-Seq has been used in capturing these histone marks

* NUN's project was funded by Swiss National Science Foundation.

** Corresponding authors.

on a genome-wide scale [2, 11]. A study of how histone marks change across various cell types could play an important role in our understanding of developmental biology and how cell differentiation occurs, particularly as the epigenetic state of chromatin is inheritable across cell generations [12].

Since cell differentiation transforms less specialized cell types into more specialized ones and since most specialized cells of one organ cannot be converted into specialized cells of some other organ, the paths of differentiation together form a tree, in many ways similar to the phylogenetic trees used to represent evolutionary histories. In evolution, present-day species have evolved from some ancestral species, while in cell development the more specialized cells have evolved from less specialized cells. Moreover, observed changes in the epigenetic state are inheritable, again much as mutations in the genome are (although, of course, through very different mechanisms and at very different scales); and in further similarity, epigenetic traits are subject to stochastic changes, much as in genetic mutations. (It should be noted that we are interested here in populations of cells of a certain type, not all coming from the same individual, rather than in developmental lineages of cells within one individual.) Finally, one may object that derived and more basic cell types coexist within the body, while phylogenetic analysis places all modern data at the leaves of the tree and typically qualifies internal nodes as “ancestral”. However, species in a phylogenetic tree correspond to paths, not to nodes. In particular, a species that has survived millions of years until today and yet has given rise to daughter species, much like a basic cell type that is observed within the organism, but from which derived cell types have also been produced and observed, is simply a path to a leaf in the tree, a path along which changes are slight enough not to cause a change in identification. (The time scale makes such occurrences unlikely in the case of species phylogenies, but the framework is general enough to include them.)

Therefore it may be possible to use or adapt some of the techniques used in building phylogenetic trees for building *cell-type trees*. There are of course significant differences between a phylogenetic tree and a cell-type tree. Two major differences stand out. The more significant difference is the lack of well established models for changes to histone marks during cell differentiation, as compared to the DNA and amino-acid mutation models in common usage in research in molecular evolution. The other difference is that functional changes in cell differentiation are primarily driven by programmed mutational events rather than by selection—and this of course makes it all the harder to design a good model. In spite of these differences, we felt that phylogenetic approaches could be adapted to the analysis of cell differentiation.

In this paper, we provide evidence that such a scenario is possible. We do this by proposing new data representation techniques and distance measures, then by applying standard phylogenetic methods to produce biologically meaningful results. We used data on two histone modifications (but mostly on H3K4me3) for 37 cell types, including replicate data, to construct cell-type trees—to our knowledge, these are the first such trees produced by computational methods. We show that preprocessing the data is very important: not only are ChIP-Seq data fairly noisy, but the ENCODE data are based on several individuals and thus adds an independent source of noise. We also outline some of the computational challenges in the analysis of cell differentiation,

opening new perspectives that may prove of interest to computer scientists, biologists, and bioinformaticians.

2 Methods

2.1 Model of Differentiation for Histone Marks

We assume that histone marks can be independently gained or lost in regions of the genome as cells differentiate from a less specialized type to a more specialized one. Histones marks are known to disappear from less specialized cell types or to appear in more specialized ones and are often correlated with gene expression, so our assumption is reasonable. The independence assumption simply reflects our lack of knowledge, but it also enormously simplifies computations.

2.2 Data Representation Techniques

The analysis of ChIP-Seq data typically starts with a peak-finding step that defines a set of chromosomal regions enriched in the target molecule. We therefore use peak lists as the raw data for our study. We can decide on the presence or absence of peaks at any given position and treat this as a binary character, matching our model of gain or loss of histone marks. Since all of the cell types have the same genome (subject only to individual SNPs or varying copy numbers), we can compare specific regions across cell types. Therefore we code the data into a matrix in which each row is associated with a different ChIP-Seq library (a different cell type or replicate), while each column is associated with a specific genomic region.

We use two different data representations for the peak data for each cell type. Our first method is a simple windowing (or binning) method. We divide the genome into bins of certain sizes; if the bin contains at least one peak, we code it 1, otherwise we code it 0. The coding of each library is thus independent of that of any other library.

Our second method uses overlap and takes into account all libraries at once. We first find interesting regions in the genome, based on peaks. Denote the i th peak in library n as $P_i^n = [P_{iL}^n, P_{iR}^n]$, where P_{iL}^n and P_{iR}^n are the left and right endpoints (as basepair indices). Consider each peak as an interval on the genome (or on the real line) and build the *interval graph* defined by all peaks in all libraries. An interval graph has one vertex for each interval and an edge between two vertices whenever the two corresponding intervals overlap [6]. We simply want the connected components of the interval graph.

Definition 1. *An interval in the genome is an interesting region iff it corresponds to a connected component of the interval graph.*

Finding these interesting regions is straightforward. Choose a chromosome, let PS be its set of peaks, set $AS = \{\emptyset\}$ and $z = 0$, and enter the following loop:

1. $P_{i^*}^* = \arg \min_{P_i^n \in PS} P_{iL}^n$. Set $a = P_{i^*L}^*$ and $AS = AS \cup \{P_{i^*}^*\}$
2. Set $S = \{P \mid P \cap P_{i^*}^* \neq \emptyset \text{ and } P \in PS\}$ and $AS = AS \cup S$.
3. If S is not empty, then find $P_{i^*}^* = \arg \max_{P_i^n \in PS} P_{iR}^n$ and go to step 2.

4. Let $b = P_{i^*R}^*$ and set $PS = PS - AS$.
5. The interesting region lies between a and b , $IR[a, b]$. Let $D_{IR}^n[z]$ be the data representation for $IR[a, b]$ in library n . Set $z = z + 1$. Set $D_{IR}^n[z] = 1$ if there is a peak in library n that lies in $IR[a, b]$; otherwise set $D_{IR}^n[z] = 0$ ($1 \leq n \leq N$).

Repeat this procedure for all chromosomes in the genome. The algorithm takes time linear in the size of the genome to identify the interesting regions.

For a given collection of libraries, these interesting regions have a unique representation. We assume that it is in these interesting regions that histone marks are lost or gained and we consider that the size of the histone mark (which depends at least in part on the experimental procedures and is typically noisy) does not matter. Our major reason for this choice of representation is noise elimination: since the positioning of peaks and the signal strength both vary from cell to cell as well as from test to test, we gain significant robustness (at the expense of detail) by merging all overlapping peaks into one signal, which we use to decide on the value of a single bit. The loss of information may be illusory (because of the noise), but in any case we do not need a lot of information to build a phylogeny on a few dozen cell types.

2.3 Phylogenetic Analysis

Phylogenetic analysis attempts to infer the evolutionary relationships of modern species or *taxa*—they could also be proteins, binding sites, regulatory networks, etc. The best tools for phylogenetic inference, based on maximum parsimony (MP) or maximum likelihood (ML), use established models of sequence evolution, something for which we have no equivalent in the context of cell differentiation. However, one class of phylogenetic inference methods uses variations on clustering, by computing measures of distance (or similarity) to construct a hierarchical clustering that is assimilated to a phylogenetic tree. This type of method is applicable to our problem, provided we can define a reasonable measure of distance, or similarity, between cell types in terms of our data representations. (We are not implying that models of differentiation do not exist nor that they could not be derived, but simply stating that none exist at present that could plausibly be used for maximum-likelihood phylogenetic inference.) Finally more that, with 0/1 data, we can also use an MP method, in spite of the absence of a valid model of character evolution.

In a cell-type tree, most cell types coexist in the present; thus at least some of them can be found both at leaves and at internal nodes. (We are unlikely to have data for all internal nodes, as we cannot claim to have observed all cell types.) Fortunately, phylogenetic inference still works in such cases: as mentioned earlier, when the same taxon should be associated with both a leaf and an internal node, we should simply observe that each edge on the path from that internal node to that leaf is extremely short, since that distance between the two nodes should be zero (within noise limits). The tree inferred will have the correct shape; however, should we desire to reconstruct the basic cell types, then we would have to *lift* some of the leaf data by copying them to some internal nodes.

From among the distance-based methods, we chose to use the most commonly used one, Neighbor-Joining (NJ) [15]. While faster and possibly better distance-based methods exist, such as FastME [4], it was not clear that their advantages would still obtain

in this new domain; and, while very simple, the NJ method has the advantage of not assuming a constant rate of evolution across lineages. In each of the two data representation approaches, we compute pairwise distance between two libraries as the Hamming distance of their representations. (The Hamming distance between two strings of equal length is the number of positions at which corresponding symbols differ.) We thus obtain a distance matrix between all pairs of histone modification libraries; running NJ on this matrix yields an unrooted tree. For MP, we used the TNT software [7].

2.4 On the Inference of Ancestral Nodes

We mentioned that lifting some of the leaf data into internal nodes is the natural next step after tree inference. However, in general, not all internal nodes can be labelled in this way, due mostly to sampling issues: we may be missing the type that should be associated with a particular internal node, or we may be missing enough fully differentiated types that some internal tree nodes do not correspond to any real cell type. Thus we are faced with a problem of ancestral reconstruction and, more specifically, with three distinct questions:

- For a given internal node, is there a natural lifting from a leaf?
- If there is no suitable lifting, is the node nevertheless a natural ancestor—i.e., does it correspond to a valid cell type?
- If the node has no suitable lifting and does correspond to a valid cell type, can we infer its data representation?

These are hard questions, in terms of both modelling and computational complexity; they are further complicated by the noisy nature of the data. Such questions remain poorly solved in standard phylogenetic analysis; in the case of cell-type trees, we judged it best not to address these problems until the tree inference part is better understood and more data are analyzed.

3 Experimental Design

The histone modification ChIP-Seq data were taken from the ENCODE project database (UW ENCODE group) for human (hg19) data [5]. We carried out experiments on both H3K4me3 and H3K27me3 histone mark data. H3K4me3 is a well studied histone mark usually associated with gene activation, while the less well studied H3K27me3 is usually associated with gene repression [13]. We used data for cell types classified as “normal” and for embryonic stem cells—we did not retain cancerous or EBV cells as their differentiation processes might be completely distinct from those of normal cells. The ENCODE project provides peaks of ChIP-Seq data for each replicate of each cell type. We therefore used their peaks as the raw input data for our work. For the windowing representation, we used bins of 200 bp: this is a good size for histone marks, because 147 bp of DNA wrap around the histone and linker DNA of about 80 bp connect two histones, so that each bin represents approximately the absence or presence of just one histone modification. We programmed our procedures in *R* and used the NJ implementation from the *ape* library in *R*.

Table 1. Cell names, short description, and general group for H3K4me3 data. For details see the ENCODE website [1].

Cell Name	Short Description	Group
AG04449	fetal buttock/thigh fibroblast	Fibroblast
AG04450	fetal lung fibroblast	Fibroblast
AG09319	gum tissue fibroblasts	Fibroblast
AoAF	aortic adventitial fibroblast cells	Fibroblast
BJ	skin fibroblast	Fibroblast
CD14	Monocytes-CD14+ from human leukapheresis production	Blood
CD20(1)	B cells replicate, African American	Blood
CD20(2) and CD20(3)	B cells replicates, Caucasian	Blood
hESC	undifferentiated embryonic stem cells	hESC
HAc	astrocytes-cerebellar	Astrocytes
HAsp	astrocytes spinal cord	Astrocytes
HBMEC	brain microvascular endothelial cells	Endothelial
HCFaa	cardiac fibroblasts- adult atrial	Fibroblast
HCF	cardiac fibroblasts	Fibroblast
HCM	cardiac myocytes	Myocytes
HCPEpiC	choroid plexus epithelial cells	Epithelial
HEEpiC	esophageal epithelial cells	Epithelial
HFF	foreskin fibroblast	Fibroblast
HFF MyC	foreskin fibroblast cells expressing canine cMyc	Fibroblast
HMEC	mammary epithelial cells	Epithelial
HPAF	pulmonary artery fibroblasts	Fibroblast
HPF	pulmonary fibroblasts isolated from lung tissue	Fibroblast
HRE	renal epithelial cells	Epithelial
HRPEpiC	retinal pigment epithelial cells	Epithelial
HUVEC	umbilical vein endothelial cells	Endothelial
HVMF	villous mesenchymal fibroblast cells	Fibroblast
NHDF Neo	neonatal dermal fibroblasts	Fibroblast
NHEK	epidermal keratinocytes	Epithelial
NHLF	lung fibroblasts	Fibroblast
RPTEC	renal proximal tubule epithelial cells	Epithelial
SAEC	small airway epithelial cells	Epithelial
SKMC	skeletal muscle cells	Skeletal Muscle
WI 38	embryonic lung fibroblast cells	Fibroblast

Table 1 show the list of the 37 cell types used for H3K4me3 data, giving for each an abbreviation and a short description. In addition, the cells are classified into various groups whose names are based on their cell type. Keratinocytes (NHEK) is included in the Epithelial group. We have two replicates for most cell types, but only one replicate for types HCFaa, HFF, and CD14, and three replicates for CD20. (CD20(1) is a B-cell from an African-American individual while CD20(2) and CD20(3) are from a Caucasian individual). The replicates are biological replicates, i.e., the data come from two independent samples. For human Embryonic Stem Cells (hESC) we have data for different days of the cell culture, so we shall use hESC D2 to mean data for hESC cells on day 2. For each cell type, we shall mention the replicate number in brackets, unless the cell type has only one replicate.

4 Results/Discussion

4.1 H3K4me3 Data on Individual Replicates

We report on our analyses using peak data from the ENCODE database for H3K4me3 histone modifications. We carried out the same analyses using H3K27me3 data, but results were very similar and so are not detailed here—we simply give one tree for comparison purposes. The similarity of results between the two datasets reinforces our contention that phylogenetic analyses yield biologically meaningful results on such data. We color-code trees to reflect the major groupings listed in Table 1.

Fig. 1 shows the trees constructed using only one replicate for each cell type using both windowing and overlap representations. The color-coding shows that embryonic stem cells and blood cells are in well separated clades of their own, while fibroblasts and epithelial cells fall in just two clades each. Even within the hESC group we see that day 0 is far off from day 14 compared to its distance from day 2. Thus epigenetic data such as histone marks do contain a lot of information about cell differentiation history.

In order to quantify the quality of the groupings, we compute the total number of cells in a subtree that belong to one group. Since our groups are based on cell type only, there could be many subdivisions possible within each group. Therefore we choose the two largest such subtrees available for each group such that each subtree contains only the leaf nodes of that group. The results are shown in Table 2: most of the cell types in each group do cluster together in the tree. Fig. 1 shows long edges between (most) leaf nodes and their parents—a disquieting feature, as it casts doubt as to the robustness of the tree, parts of which could be assimilated to stars. To quantify this observation, we measured the SR ratio, defined as $SR = \frac{\sum_{e \in I} l(e)}{\sum_{e \in E} l(e)}$, where I is the set of all edges connecting leaf nodes to their parents, E is the set of all edges in the tree, and $l(e)$ is the length of edge e . If this ratio SR is close to 1, then the tree looks star-shaped with long branches to the leaves. This ratio was 0.93 using the windowing representation;

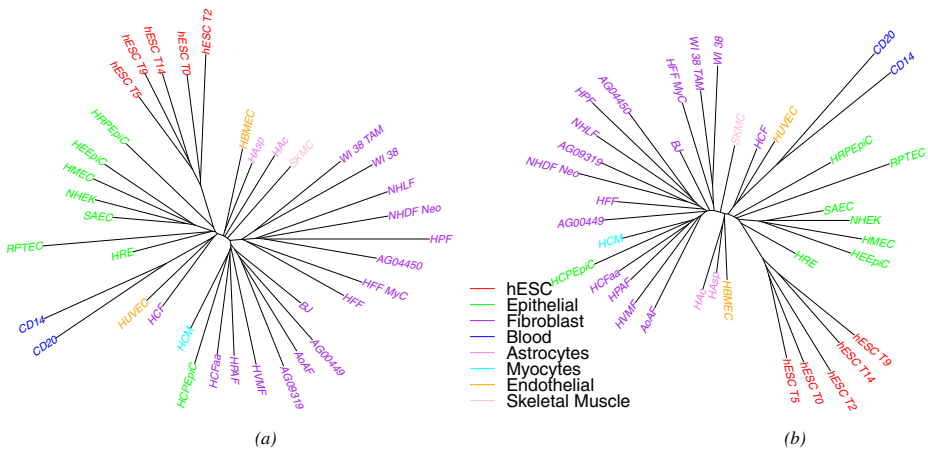


Fig. 1. Cell-type tree on H3K4me3 data using only one replicate: (a) windowing representation, (b) overlap representation.

Table 2. Statistics for cell-type trees on H3K4me3 data. 2nd to 9th columns show the number of cells (of the same type) belonging to the largest and second-largest clades; the total number of cells of that type is in the top row. Rows correspond to various methods (WM: windowing; OP: overlap; TP: top peaks). The last column contains the percent deviation (PD) of the distances between the leaves found using the NJ tree from the Hamming distance between the leaves.

	hESC (5)	Epithelial (8)	Fibroblast (16)	Blood (2)	Astrocytes (2)	Myocytes (1)	Endothelial (2)	Skeletal Muscle (1)	SR	PD (%)
WM (one replicate)	5,0	6,1	8,4	2,0	1,1	1,0	1,1	1,0	0.93	3.20
OM (one replicate)	5,0	4,1	6,3	2,0	2,0	1,0	1,1	1,0	0.92	3.94
WM (all replicates)	5,0	6,1	11,2	2,0	1,1	1,0	1,1	1,0	0.84	3.30
OM (all replicates)	5,0	4,2	9,4	2,0	2,0	1,0	1,1	1,0	0.78	3.88
WM (all replicates)-TP	5,0	6,1	7,4	2,0	1,1	1,0	1,1	1,0	0.81	3.73
OM (all replicates)-TP	5,0	4,3	8,5	2,0	2,0	1,0	1,1	1,0	0.74	3.98

using the overlap representation reduced it very slightly to 0.92. These long branches are due in part to the very high level of noise in the data, explaining why the overlap representation provided a slight improvement.

As a final entry in the table, we added another measure on the tree and the data. The NJ algorithm is known to return the “correct” tree when the distance matrix is ultrametric; the technical definition does not matter so much here as the consequence: if the matrix is ultrametric, then the sum of the length of the edges on the path between two leaves always equals the pairwise distance between those two leaves in the matrix. Thus one way to estimate how far the distance matrix deviates from this ideal is to compare its distances to the length of the leaf-to-leaf paths in the tree:

$$PD = \frac{\sum_{i,j} |NJ(i,j) - M(i,j)|}{\sum_{i,j} NJ(i,j)}$$

where i and j are leaf nodes, $NJ(i,j)$ is the tree distance between i and j , and $M(i,j)$ is the matrix distance between i and j . A high value of PD indicates that the data representations and measures do not fit well to any tree. We get very low values (of less than 4% for both windowing and overlap representations), suggesting that the distances we compute are in fact representative of a tree and thus offering confirmation of the validity of the inference.

4.2 H3K4me3 Data with All Replicates

By bringing replicates into the analysis, we can expect to see a stronger phylogenetic signal as each replicate adds to the characterization of its cell type. In particular, whenever we have two or more replicates, they should form a tight subtree of their own. We thus used our replicate data (two replicates for 33 of the 37 cell types, and three for one type, for a total of 72 libraries) in the same analysis pipeline. Fig. 2 shows the differentiation trees obtained using windowing and overlap representations. For completeness, we include the same study (in overlap representation only) on H3K27me3 data in Fig. 3. (Finally, the trees obtained using TNT are very similar and not shown.) As expected, almost all replicates are grouped; since we usually have two replicates, we get a collection of “cherries” (pairs of leaves) where we had a single leaf before. In most cases,

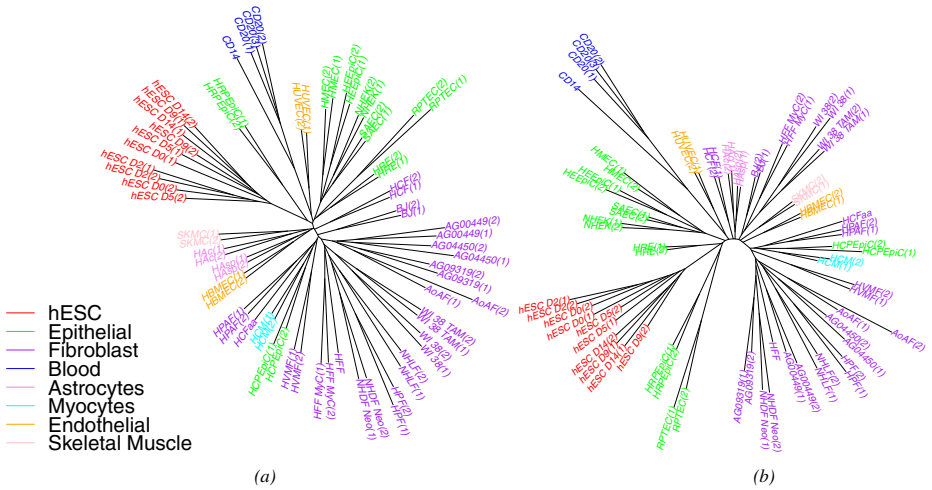


Fig. 2. Cell-type tree on H3K4me3 data (using all replicates): (a) windowing representation, (b) overlap representation.

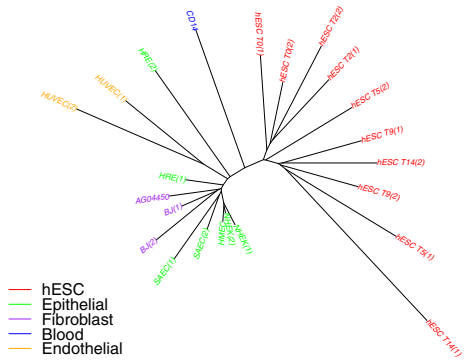


Fig. 3. Cell-type tree on H3K27me3 data, using all replicates and overlap representation.

it is now the distance from each leaf in a cherry to their common parent that is large, indicating that the distance between the two replicates is quite large—as we can also verify from the distance matrix. This suggests much noise in the data. This noise could be at the level of raw ChIP-Seq data, but also due to the bias of peak-finding methods used—one expects a general-purpose peak finder to be biased against false negatives and more tolerant of false positives, but for our application we would be better served by the inverse bias. Another reason for the large distance is the nature of the data: these are biological replicates, grown in separate cultures, so that many random losses or gains of histone marks could happen once the cell is differentiated. Thus it may be that only a few of the mutations in the data are correlated with cell differentiation. Identifying these few mutations would be of high interest, but with just two replicates we are unlikely to pinpoint them with any accuracy.

Looking again at Table 2, we see that, using the windowing representation, the value of SR for the full set of replicates is 0.84 and that here the overlap representation, which is more effective at noise filtering, yields an SR value of 0.78. This is a significant reduction and indicates that the long edges are indeed due to noise. The PD percentage values remain very low for both representations, so the trees we obtained do represent the data well. Note that the groupings appear (in the color-coding in the figure) somewhat better than when we used only one replicate, and the values in columns 2 through 9 of Table 2 confirm this impression.

4.3 Using Top Peaks and Masking Regions

In order to study the nature of the noise, we removed some of the less robust peaks. The ENCODE dataset gives a p-value for each peak listed; we kept only peaks with (negative) log p-values larger than 10. We kept all replicates and ran the analysis again, with the results depicted in Fig. 4. The PD percentage values are again very low, so the trees once again fit the data well. The improvement looks superficially minor, but we obtained some more biologically meaningful clusters with this approach. For example, in the fibroblast group, the top two subtrees in Table 2 changed from (9,4) to (8,5) when we used only top peaks in the overlap method. This change occurred because cell HFF moved from the larger group to the smaller group forming a subtree with HFF-Myc (which makes more sense as both are foreskin fibroblast cells). Such a change could be due to particularly noisy data for the HFF cells having obscured the relationship before we removed noisy peaks. Overall, removing noisy peaks further reduced the SR ratio from 0.78 to 0.74 for the overlap representation and from 0.84 to 0.81 for the windowing representation.

Another typical noise-reduction procedure, much used in sequence analysis, is to remove regions that appear to carry little information or to produce confounding

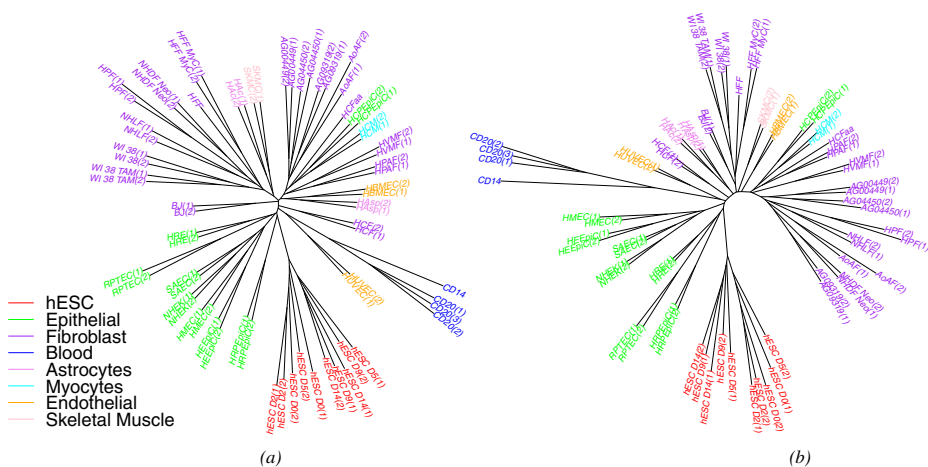


Fig. 4. Cell-type tree on H3K4me3 data (using all replicates) on peaks with negative log p-value ≥ 10 : (a) windowing representation, (b) overlap representation.

indications—a procedure known as masking. We devised a very simplified version of masking for our problem, for use only with replicate data, by removing any region within which at most one library gave a different result (1 instead of 0 or vice versa) from the others. In such regions, the presence of absence of peaks is perfectly conserved across all but one replicate, indicating the one differing replicate has probably been called wrong. After removing such regions, we have somewhat shorter representations, but follow the same procedure. The trees returned have exactly the same topology and so are not shown; the length of edges changed very slightly, as the *SR* value decreased from 0.74 down to 0.70 using top peaks in the overlap representation.

4.4 A Better Looking Tree

Barring the addition of many replicates, the *SR* ratio of 0.70 appears difficult to reduce and yet remains high. However, the cherries of replicate pairs by themselves give an indication of the amount of “noise” (variation among individual cells as well as real noise) present in the data. We can take that noise out directly by replacing each cherry with its parent, which is a better representative of the population of this particular cell type than either of the two leaves. We carried out this removal on the tree of Fig. 2(b) and obtained the tree shown in Fig. 5. Since hESC cells do not form clear pairs, we replaced the entire clade of hESC cells by their last common ancestor. The leaves with remaining long edges are those for which we did not have a replicate (CD14, HCFaa, and HFF).

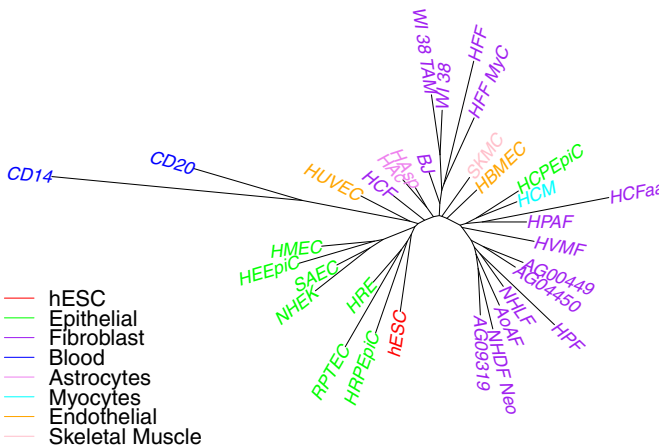


Fig. 5. H3K4me3 data, overlap representation on peaks with negative log p-value ≥ 10 . Replicate leaves are removed and replaced by their parent.

5 Conclusions

We addressed the novel problem of inferring cell-type trees from histone modification data. We defined methods for representing the peaks as 0/1 vectors and used these vectors to infer trees. We obtained very good trees, conforming closely to expectations

and biologically plausible, in spite of the high level of noise in the data and the very limited number of samples per cell type. Our results confirm that histone modification data contain much information about the history of cell differentiation. We carried out a number of experiments to understand the source of the noise, using replicate data where available, but also devising various noise filters. Our results show that larger replicate populations are needed to infer ancestral nodes, an important step in understanding the process of differentiation. Refining models will enable the use of likelihood-based methods and thus lead to better trees. Since many histone marks appear independent of cell differentiation, identifying which marks are connected with the differentiation process is of significant interest. Finally, once such marks have been identified, reconstructing their state in ancestral nodes will enable us to identify which regions of the genome play an active role in which steps of cell differentiation.

References

1. <http://genome.ucsc.edu/cgi-bin/hgFileUi?db=hg19&g=wgEncodeUwHistone>
2. Barski, A., et al.: High-resolution profiling of histone methylations in the human genome. *Cell* 129(4), 823–837 (2007)
3. Berger, S.L.: Histone modifications in transcriptional regulation. *Current Opinion in Genetics & Development* 12(2), 142–148 (2002)
4. Desper, R., Gascuel, O.: Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. In: Guigó, R., Gusfield, D. (eds.) *WABI 2002*. LNCS, vol. 2452, pp. 357–374. Springer, Heidelberg (2002)
5. Project Consortium ENCODE: A user's guide to the encyclopedia of DNA elements (ENCODE). *PLoS Biol.* 9(4), e1001046 (2011)
6. Fishburn, P.C.: *Interval orders and interval graphs: A study of partially ordered sets*. Wiley New York (1985)
7. Goloboff, P.A.: Analyzing large data sets in reasonable times: solutions for composite optima. *Cladistics* 15(4), 415–428 (1999)
8. Lee, J.-H., Hart, S.R., Skalnik, D.G.: Histone deacetylase activity is required for embryonic stem cell differentiation. *Genesis* 38(1), 32–38 (2004)
9. Lister, R., et al.: Hotspots of aberrant epigenomic reprogramming in human induced pluripotent stem cells. *Nature* 471(7336), 68–73 (2011)
10. Lobe, C.G.: Transcription factors and mammalian development. *Current Topics in Developmental Biology* 27, 351–351 (1992)
11. Mardis, E.R., et al.: ChIP-seq: welcome to the new frontier. *Nature Methods* 4(8), 613–613 (2007)
12. Martin, C., Zhang, Y.: Mechanisms of epigenetic inheritance. *Current Opinions Cell Biology* 3(19), 266–272 (2007)
13. Mikkelsen, T.S., et al.: Genome-wide maps of chromatin state in pluripotent and lineage-committed cells. *Nature* 448(7153), 553–560 (2007)
14. Nelson, D.L., Cox, M.M.: *Lehninger principles of biochemistry*. W.H. Freeman (2010)
15. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4(4), 406–425 (1987)

Detecting Superbubbles in Assembly Graphs

Taku Onodera¹, Kunihiro Sadakane², and Tetsuo Shibuya¹

¹ Human Genome Center, Institute of Medical Science, University of Tokyo 4-6-1
Shirokanedai, Minato-ku, Tokyo 108-8639, Japan

{tk-ono,tshibuya}@hgc.jp

² National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku,
Tokyo 101-8430, Japan

sada@nii.ac.jp

Abstract. We introduce a new concept of a subgraph class called a superbubble for analyzing assembly graphs, and propose an efficient algorithm for detecting it. Most assembly algorithms utilize assembly graphs like the de Bruijn graph or the overlap graph constructed from reads. From these graphs, many assembly algorithms first detect simple local graph structures (motifs), such as tips and bubbles, mainly to find sequencing errors. These motifs are easy to detect, but they are sometimes too simple to deal with more complex errors. The superbubble is an extension of the bubble, which is also important for analyzing assembly graphs. Though superbubbles are much more complex than ordinary bubbles, we show that they can be efficiently enumerated. We propose an average-case linear time algorithm (*i.e.*, $O(n+m)$ for a graph with n vertices and m edges) for graphs with a reasonable model, though the worst-case time complexity of our algorithm is quadratic (*i.e.*, $O(n(n+m))$). Moreover, the algorithm is practically very fast: Our experiments show that our algorithm runs in reasonable time with a single CPU core even against a very large graph of a whole human genome.

1 Introduction

The sequencing technologies have evolved dramatically in the past 25 years, and nowadays many next-generation sequencers (NGSs) can sequence a human genome-size genome in only a few hours with very small costs. But still there is no sequencing technology that can sequence the entire genome at a time without breaking the genome into millions or billions of short reads. Thus assembling these reads into a whole genome has been one of the most important computational problems in molecular biology, and quite a few algorithms have been proposed for the problem [5, 9, 14] despite the computational difficulty of the problem [10].

Most assembly algorithms construct some graph in their first stage. They are categorized into two types depending on the types of the graph. Many old-time assemblers utilize a graph called the *overlap graph*, in which a vertex corresponds to a read and an edge corresponds to a pair of reads that have an enough-length overlap [1, 3, 11]. More recent algorithms often utilize a graph called the *de*

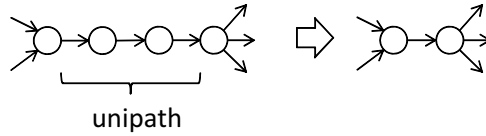


Fig. 1. Construction of a unipath graph

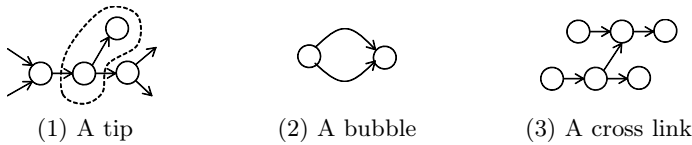


Fig. 2. Assembly graph simple motifs

Bruijn graph, in which an edge corresponds to a k -mer that exists in reads and a vertex corresponds to the shared $(k - 1)$ -mer between the adjacent k -mers [4, 6, 8, 13, 15–17]. The de Bruijn graph is said to be more suitable for NGS short reads of large depth.

The next step of most sequencing algorithms after constructed the graph is to simplify the obtained graph by decomposing a maximal unbranched sequence of edges (which is called a *unipath*) into one single edge [4, 8, 15] (Fig. 1). The obtained graph is called a *unipath graph*. After obtained the unipath graph, many sequencing algorithms next detect simple typical motif structures caused by errors to detect errors: The most common motifs are tips, bubbles, and cross links [4, 6, 15, 17] (Fig. 2).

A tip (Fig. 2 (1)) is a low-frequency edge whose end (or start) vertex has no outgoing (resp. incoming) edges, which goes out from (resp. comes into) a high-frequency vertex¹. This motif often appears in case there are some error(s) around the end of a read. A bubble (Fig. 2 (2)) consists of multiple edges (with the same direction) between a pair of vertices, which is often caused by error(s) somewhere in the middle of a read. A cross link (Fig. 2 (3)) is a low-frequency edge that lies between high-frequency vertices. This appears when a substring of a read accidentally becomes (by error) the same substring that appears in a different region. All of these motifs are easy to find (obviously in linear time) due to their simplicity.

But we should consider much more complex structures if input reads are erroneous (as in the case of the third generation sequencers), have many repeats (as in many large-scale genomes/meta-genomes), or have many mutations (as in cancer genomes). Fig. 3 shows an example of a subgraph of a unipath graph obtained from actual whole human genome reads (the same set of reads used in

¹ We say 'low/high'-frequency vertices/edges for vertices/edges that correspond to few/many reads.

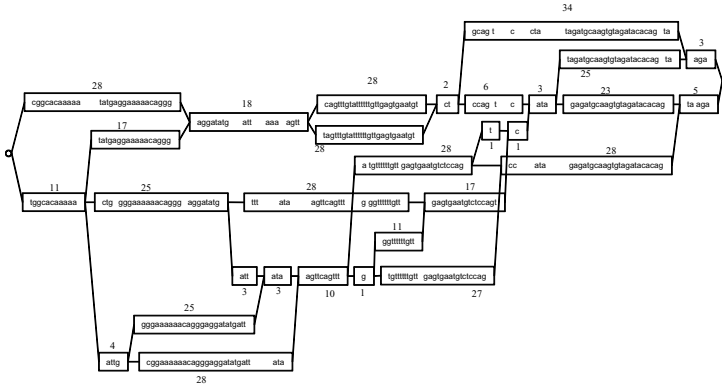


Fig. 3. A superbubble: A very complicated structure caused by errors or repeats. All the edges are labeled with sequences (vertices are not shown). The gaps in the labels are inserted manually in the figure to show alignment between edge labels that start at different offsets from the entrance of the superbubble.

the experiments in section 4). In this subgraph, paths from the leftmost vertex branch to many paths but they converge into the rightmost single vertex in the end, and there are no cycles in this subgraph, *i.e.*, the subgraph forms a directed acyclic graph (DAG). The vertices between the leftmost vertex and the rightmost vertex has no outgoing/incoming edges to/from external vertices (*i.e.*, vertices not in this subgraph). An important point is that all the paths have similar labels with similar lengths.² We call this kind of a subgraph a *superbubble*, as it can be considered as an extension of an ordinary simple bubble (more detailed definition of superbubbles will be given in section 2). Superbubbles are complicated, but it is apparent that many of them are formed as a result of errors, inexact repeats, diploid/polyploid genomes, or frequent mutations. Thus detection of superbubbles should be very important, and it should be useful if we can detect them efficiently. For example, further time-consuming complicated algorithms (e.g., optimal alignment, paired-end read analyses, etc) are applicable against the superbubbles, even if they are too complicated to use against the entire graph.

In the followings, we will give detailed definition of the superbubbles in section 2, and show an efficient algorithm for finding superbubbles in section 3. We will show that the algorithm runs in average-case linear time against graphs with a reasonable model, though the worst-case time complexity is quadratic. In section 4, we will show that the superbubbles can be efficiently enumerated in reasonable time with a small machine, through large-scale experiments against reads from a whole human genome.

² The experiments in section 4 will show that the path label lengths of a superbubble are only at most 5% different in more than 85% of the detected superbubbles.

2 Preliminaries

2.1 Superbubble

Here, we formally define superbubbles and show some properties of them which are necessary in the rest of the paper.

Definition 1. Let $G = (V, E)$ be a directed graph. If an ordered pair of distinct vertices (s, t) satisfies the following:

reachability t is reachable from s ;

matching the set of vertices reachable from s without passing³ through t is equal to the set of vertices from which t is reachable without passing through s ;

acyclicity the subgraph induced by U is acyclic where U is the set of vertices in the above condition;

minimality no vertex in U other than t forms a pair with s that satisfies the conditions above,

then we say that the subgraph in the description of the acyclicity condition is a **superbubble** and s , t and $U \setminus \{s, t\}$ are this superbubble's **entrance**, **exit** and **interior** respectively. For any pair of vertices (s, t) that satisfies the above conditions, we denote the superbubble as $\langle s, t \rangle$.

To take full advantage of the notation $\langle s, t \rangle$, we first need to confirm that if $(s_1, t_1) \neq (s_2, t_2)$ then $\langle s_1, t_1 \rangle \neq \langle s_2, t_2 \rangle$. The following remark ensures it.

Remark 1. There is a one-to-one correspondence between the vertex pairs satisfying the conditions in Definition 1 and superbubbles.

Proof. Because of the acyclicity condition, the vertices of a superbubble can be topologically sorted, *i.e.*, they can be ordered in such a way that if v is reachable from u then $u < v$. Due to the matching condition, s (resp. t) is the minimum (resp. maximum) ordered vertex.

Now we observe a proposition which clarifies the situation and motivates linear time enumeration of superbubbles.

Proposition 1. Any vertex can be the entrance (resp. exit) of at most one superbubble.

Note that this proposition does not exclude the possibility that a vertex is the entrance of a superbubble and the exit of another superbubble.

Proof. We prove the proposition by *reductio ad absurdum*. Suppose $\langle s, t_1 \rangle$ and $\langle s, t_2 \rangle$ are distinct superbubbles. If t_2 is a vertex in $\langle s, t_1 \rangle$, then t_2 is in the interior of $\langle s, t_1 \rangle$ but this contradicts to the minimality condition for $\langle s, t_1 \rangle$. Similarly, t_1 being a vertex in $\langle s, t_2 \rangle$ also results in a contradiction.

³ Passing through a vertex means that visiting and then leaving it, not just visiting or leaving alone.

Suppose, on the other hand, that t_2 is not a vertex in $\langle s, t_1 \rangle$. There is a path from s to t_2 . By removing cycles from t_2 to t_2 if necessary, this path can be taken in such a way that t_2 appears only at the last step and at this time, all vertices in the path are in $\langle s, t_2 \rangle$. On the other hand, the vertex just before the first vertex on the path that is not in $\langle s, t_1 \rangle$ is t_1 . In particular this means that t_1 is in $\langle s, t_2 \rangle$ but this leads to contradiction by the first half of the argument.

Corollary 1. *There are $O(n)$ superbubbles in a graph with n vertices.*

Before closing this subsection, let us point out, without proof, yet another property of superbubbles that is not directly necessary for this work but worth mentioning to grasp the picture.

Claim. If two distinct superbubbles share a vertex, either one's exit is the other's entrance or one is included in the other's interior.

2.2 Construction of a Unipath Graph

Given a set \mathcal{R} of reads, we first construct the de Bruijn graph [13]. Let $T = T[1, m]$ be a read of length m in \mathcal{R} . The k -mers of T are length- k substrings of T , that is, $T[i, i+k-1]$ for $i = 1, 2, \dots, m-k+1$. Let K denote the multiset of k -mers of all reads in \mathcal{R} , and K_d denote the set of (distinct) k -mers that appear at least d times in K . A k -mer in K_d is called a *solid k -mer*.

The de Bruijn graph $G = (V, E)$ of \mathcal{R} is defined as follows. The vertex set V is the set of $(k-1)$ -mers defined as $V = \{T[1, k-1] \mid T[1, k] \in K_d\} \cup \{T[2, k] \mid T[1, k] \in K_d\}$. The edge set E is defined as $\{(u, v) \mid \exists T[1, k] \in K_d, u = T[1, k-1], v = T[2, k]\}$. The edge label of (u, v) is $T[k]$ if $u = T[1, k-1], v = T[2, k]$. Typical values of k and d are $k = 28, d = 3$.

We use the succinct de Bruijn graph [2], which is a compressed representation of the de Bruijn graph of \mathcal{R} . For a set of m solid k -mers, the succinct de Bruijn graph uses $4m + o(m)$ bits to encode the graph, and supports the following operations.

- $outdeg(v)/indeg(v)$ returns the number of outgoing/incoming edges from/to vertex v in $O(1)$ time, respectively.
- $outgoing(v, c)$ returns the vertex w pointed to by the outgoing edge of vertex v with edge label c in $O(1)$ time. If no such vertex exists, it returns -1 .
- $incoming(v, c)$ returns the vertex $w = T[1, k-1]$ such that there is an edge from w and v and $T[1] = c$ in $O(k)$ time. If no such vertex exists, it returns -1 .

From a de Bruijn graph $G = (V, E)$, we construct a unipath graph $G' = (V', E')$ as follows. The vertex set V' is a subset of V such that any vertex in V' has more than one outgoing edges or more than one incoming edges. The edge set E' is the multiset of all pairs (u, v) such that $u, v \in V'$ and there is a path $u, x_1, x_2, \dots, x_\ell, v$ in G and outdegrees and indegrees of x_1, x_2, \dots, x_ℓ are all one. The edge label of (u, v) is the concatenation of edge labels of $(u, x_1), (x_1, x_2), \dots, (x_{\ell-1}, x_\ell), (x_\ell, v)$ in G . The length of the edge label is $\ell + 1$.

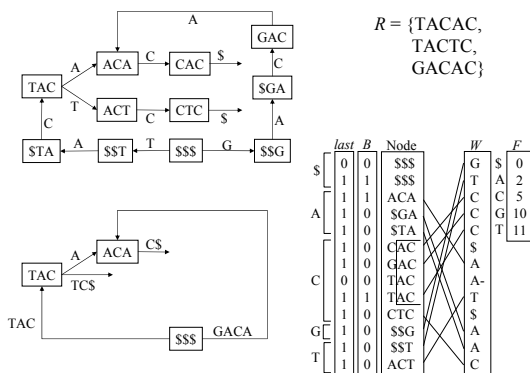


Fig. 4. Top right: The input set \mathcal{R} , top left: The de Bruijn graph of \mathcal{R} with $k = 3, d = 1$, bottom left: the unipath graph, bottom right: the succinct de Bruijn graph and the unipath graph. Non-branching nodes are removed. We store only *last*, *B*, *W* and *F*.

In addition to the data structure of the succinct de Bruijn graph, we use a bit vector $B[1, m]$ where $m = |E|$ is the number of edges in G to represent the unipath graph G' . We set $B[v] = 1$ if and only if the vertex v of G is also a vertex of G' . The outdegree and the indegree of v in G' is equal to those of v in G . To find the vertex $outgoing(v, c)$ in G' , we first compute $w = outgoing(v, c)$ in G . Then we repeatedly traverse the unique outgoing edge of w until $B[w] = 1$. The resulting vertex is the answer. The unipath graph is constructed in linear time from the succinct de Bruijn graph because each of the *outdeg*, *indeg*, and *outgoing* operations takes constant time. Figure 4 shows an example.

3 Algorithm

Here, we explain how to enumerate all superbubbles in a given graph. As we have seen in subsection 2.1, each vertex can be the entrance of at most one superbubble. Therefore, once we have a way to check if a vertex s has another vertex t s.t. (s, t) is the entrance/exit pair, then we can find all superbubbles just by iterating this procedure for all $s \in V$. Below, we focus our attention on this reduced problem.

Description. The algorithm is based on the standard topological sorting. It takes a directed graph $G = (V, E)$ and $s \in V$ as inputs, and returns $t \in V$ s.t. (s, t) is an entrance/exit pair of a superbubble if any. It proceeds by visiting vertices one by one maintaining the dynamic set S of vertices it can visit the next time. Initially, S is set to be $\{s\}$. It also maintains a label for each vertex. The label *visited* means that the vertex has already been visited. The label *seen* means that the vertex has at least one visited parent. At each step, the algorithm picks out an arbitrary vertex v from S labeling it as *visited* and label each child as

seen. If all the parents of a child are visited, it pushes the child into S . In visiting vertices, the algorithm aborts anytime when it finds a vertex with no child, which means a tip, or a parent of s , which means a cycle because any vertex visited is a descendent of s . After visiting a vertex, the algorithm tests if it is going to visit the exit at the next step as follows. First it checks if S consists of one vertex, say t , and no vertex other than t is labelled as seen. If not, the test is negative. Otherwise, the algorithm further checks if the edge (t, s) exists or not. If it does, the algorithm aborts because it just found a path from s to s , a cycle. Otherwise, the algorithm returns t . The algorithm aborts if S runs out.

Require: directed graph $G = (V, E)$, $s \in V$

Ensure: returns t s.t. (s, t) is an entrance/exit pair of a superbubble if any

```

1: push  $s$  into  $S$ 
2: repeat
3:   pick out an arbitrary  $v \in S$ 
4:   label  $v$  as visited
5:   if  $v$  does not have a child then
6:     abort // tip
7:   for  $u$  in  $v$ 's children do
8:     if  $u = s$  then
9:       abort // cycle including  $s$ 
10:    label  $u$  as seen
11:    if all of  $u$ 's parents are visited then
12:      push  $u$  into  $S$ 
13:    if only one vertex  $t$  is left in  $S$  and no other vertex is seen then
14:      if edge  $(t, s)$  does not exist then
15:        return  $t$ 
16:      else
17:        abort // cycle including  $s$ 
18: until  $|S| = 0$ 

```

Fig. 5. Pseudocode of an algorithm to find the corresponding exit of an potential entrance

Correctness. A vertex can be pushed into S at most once because it happens when all its parents are visited and once visited a vertex never cease to being so. Thus, the algorithm can pick out a vertex from S at most n times and in particular it halts. Below, we prove the correctness of the returned value, which reduces to the followings: a) if the input vertex is the entrance of some superbubble, then the algorithm returns the corresponding exit; b) if the algorithm returns a vertex, it is the exit of a superbubble and the input vertex is the corresponding entrance.

First, we observe an invariant. Let V_{seen} be the set of vertices labelled as seen and V_{visited} be the set of vertices labelled as visited. Let V_{to} be the set of vertices that are reachable from s without passing through any element of V_{seen} and let V_{from} be the set of vertices from which at least one element of $V_{\text{visited}} \cup S$ can be reachable without passing through s .

Lemma 1. *After the algorithm visits a vertex, i.e., after the line 12 of the pseudocode in Figure 5 is executed, $V_{to} = V_{visited} \cup V_{seen}$ and $V_{from} = V_{visited} \cup S$. In particular, if the algorithm returns t , then (s, t) satisfies the matching condition.*

Proof. We prove the first half by mathematical induction. After the first visit, $V_{visited}$, V_{seen} and S consist of s , s 's children and s 's children with indegree 1 respectively and the lemma holds. Suppose the lemma holds up to the visit to some vertex. During the visit to the next vertex, say v ,

1. v is removed from S and its label is changed from seen to visited;
2. all children of v are labelled as seen;
3. the children of v whose parents are all visited are added to S .

Consequently, both V_{to} and $V_{visited} \cup V_{seen}$ acquire the vertices reachable from v without passing through any element of V_{seen} , i.e., the children of v . Therefore, $V_{to} = V_{visited} \cup V_{seen}$ still holds. On the other hand, $V_{visited} \cup S$ acquires the vertices newly added to S , i.e., the children of v whose parents are all labelled as visited. Now these vertices are also in V_{from} because $V_{from} \supseteq V_{visited} \cup S$ by definition. Furthermore, they are the only vertices V_{from} acquires because the parents of them were already in V_{from} after the previous visit by the induction hypothesis. Therefore, $V_{from} = V_{visited} \cup S$ also stays true.

Next, we prove the last half. If the algorithm returns t , after the last visit, $V_{to} = V_{from}$ because $S = V_{seen}$ due to the first half. On the other hand, at this time, V_{to} consists of the vertices reachable from s without passing through t because $V_{seen} = \{t\}$. Therefore, it suffices to show that V_{from} consists of the vertices from which t is reachable without passing through s . This is true because after every visit, from any vertex in $V_{visited}$ at least one vertex in V_{seen} is reachable without passing through s , a fact which can be proven easily by mathematical induction again.

Next, we prove a). Let t be the exit corresponding to s . Because of the matching condition of (s, t) , the algorithm never aborts due to a tip or running out of S at least up to the point when t is pushed into S , no matter if t is pushed into S at all. Similarly, the algorithm never aborts due to a cycle up to the same point because of the acyclicity condition of (s, t) . On the other hand, if t is indeed pushed into S , then t must be the only vertex seen and all other vertices of $\langle s, t \rangle$ must be visited due to the matching condition of (s, t) and the lemma. Therefore, the only possibilities left are that the algorithm outputs t or some other vertex in $\langle s, t \rangle$. But the second case never happens because a vertex, say v , other than t in $\langle s, t \rangle$ is output, then the pair (s, v) satisfies the reachability, matching (due to the lemma) and acyclicity conditions, which contradicts to the minimality condition of (s, t) .

Last, we prove b). Suppose the algorithm returns a vertex t . Obviously, t is reachable from s . The matching condition holds because of the lemma. The alleged superbubble does not contain cycles including s because otherwise the algorithm must have aborted. And it does not contain cycles not including s because otherwise the first vertex visited in the cycle has a parent in the cycle.

Table 1. Histogram of the size of superbubbles

size	3-9	10-19	20-29	30-39	40-49	50-59	60-
#S.B.	71663	4295	347	69	21	8	3

This means the parent has been visited earlier, which contradicts the way the child was chosen. Thus, the acyclicity condition holds. The minimality condition holds because otherwise, there is a vertex v s.t. (s, v) is an entrance/exit pair and because of a) the algorithm must have returned v , instead of t .

Analysis. In the worst case, each execution of the algorithm takes $\Theta(n + m)$ -time and in total the calculation of all superbubbles takes $\Theta(n(n + m))$ -time. Below, we show that, under a reasonable model, the algorithm takes constant time on average and thus all superbubbles can be found in $\Theta(n)$ -time in total.

As we will see in the next section, although there are tens of thousands of superbubbles in practical unipath graphs, the entire graph is so large that its size is orders of magnitude greater than the total size of superbubbles. Thus, most of the time spent in the iterated executions of the algorithm is dedicated for traversing regions that are far away from any superbubbles. Therefore, it is reasonable to reduce the analysis of the algorithm to the evaluation of the time spent until the traversal of a non-superbubble region is aborted. In such a case, if a vertex is not pushed into S when it is labelled as *seen*, then it is very unlikely to be visited afterwards. In other words, once the algorithm comes across a vertex of indegree greater than 1, then it almost never proceeds to traverse its descendants. With these observations in mind, we model the way the tree of visited vertices grows in the algorithm by the following probabilistic tree generation process. It starts from the root. Each vertex is good with probability p . A good vertex corresponds to a vertex of indegree 1. If a vertex is good, it spawns i children with probability p_i . The theory of Galton-Watson branching processes [7] tells that the expected number of vertices of depth i is $\Theta(r^i)$ where $r := p \sum_i i p_i$, i.e., the expected number of children of each vertex. Therefore, if $r < 1$ the expected size of the tree is $\Theta(\frac{1}{1-r})$, a constant. For the unipath graph we constructed from human genome data, r was about 0.77 where p and p_i were determined as the proportion of vertices with particular in/out-degree within all vertices.

4 Experiment

Procedures. We first constructed the succinct de Bruijn graph with parameter $k = 27$ and $d = 3$ for the read set SRX016231, which was derived by sequencing a human individual by an Illumina sequencer. The length of each read is 100bp and the coverage is about 40. Next, we constructed the unipath graph as described in subsection 2.2. The resulting unipath graph consists of 107,154,751 vertices and 210,207,840 edges. Last, we found all superbubbles in the unipath graph by the algorithm in section 3.

Results. Table 1 is the histogram of the size of superbubbles where the size of a superbubble means the number of vertices in it. The superbubbles of size 2 are omitted because they are ordinary bubbles. The superbubble of Fig. 3 is of size 20 and this histogram tells, among other things, that there are hundreds of equally or more complex superbubbles. On the other hand, what matters the most for the application to genome assembly problem is whether superbubbles really capture erroneous or repeat/mutation abundant regions, which topological complexity alone does not necessarily suggest. One way to assess the relevance of a superbubble in this regard is to compare the length of paths in it where length of an edge is the length of the sequence represented by the edge. Note that topologically close paths can have a variety of lengths because each edge can be originated from a unipath. But among 23,078 superbubbles of size equal to or greater than 5 we found, 19,926 (86.3%) of them have the longest/shortest path length ratio smaller than 1.05. Therefore, superbubbles like that of Fig. 3 are indeed typical.

In terms of the computation time, it took 742.1 seconds for a Xeon 3.0GHz CPU to enumerate all superbubbles including ordinary bubbles. The number of vertices visited was 126,537,254.

5 Concluding Remarks

We introduced the concept of superbubbles in assembly graphs, and proposed an efficient algorithm for detecting them. But many tasks remain as future work. It is an open problem whether it is possible to detect superbubbles in worst-case linear time. Developing methods for categorizing the detected superbubbles (e.g., errors, repeats, mutations, and polyploids), and methods for fixing errors in superbubbles are important future tasks. It is also interesting to extend our algorithm for other bubble-like structures (e.g. the bulge structure [12]).

Acknowledgments. KS and TS are supported in part by KAKENHI 23240002. This research was supported by JST, ERATO, Kawarabayashi Large Graph Project. The super-computing resource was provided in part by Human Genome Center, the Institute of Medical Science, the University of Tokyo.

References

1. Batzoglou, S., Jaffe, D.B., Stanley, K., Butler, J., Gnerre, S., Mauceli, E., Berger, B., Mesirov, J.P., Lander, E.S.: Arachne: a whole-genome shotgun assembler. *Genome Research* 12, 177–189 (2002)
2. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de bruijn graphs. In: Raphael, B., Tang, J. (eds.) *WABI 2012*. LNCS, vol. 7534, pp. 225–235. Springer, Heidelberg (2012)
3. Huang, X., Yang, S.P.: Generating a genome assembly with pcap. *Current Protocols in Bioinformatics*, Unit 11.3 (2005)

4. Jackson, B., Regennitter, M., Yang, X., Schnable, P.S., Aluru, S.: Parallel de novo assembly of large genomes from high-throughput short reads. In: Proc. 24th International Parallel and Distributed Processing Symposium (IPDPS), pp. 1–10 (2010)
5. Kasahara, M., Morishita, S.: Large-Scale Genome Sequence Processing. Imperial College Press (2006)
6. Li, R., Zhu, H., Ruan, J., Qjan, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Yang, H., Wang, J.: De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research* 20, 265–272 (2010)
7. Lyons, R., Peres, Y.: Probability on Trees and Networks. Cambridge University Press (2012) (in preparation), Current version available at <http://mypage.iu.edu/string~rdlyons/>
8. MacCallum, I., Przybylski, D., Gnerre, S., Burton, J., Shlyakhter, I., Gnirke, A., Malek, J., McKernan, K., Ranade, S., Shea, T.P., Williams, L., Young, S., Nusbaum, C., Jaffe, D.B.: Allpaths 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology* 10(R103) (2009)
9. Miller, J.R., Koren, S., Sutton, G.: Assembly algorithms for next-generation sequencing data. *Genomics* 95, 315–327 (2010)
10. Myers, E.W.: Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology* 2, 275–290 (1995)
11. Myers, E.W., Sutton, G.G., Delcher, A.L., Dew, I.M., Fasulo, D.P., Flanigan, M.J., Kravitz, S.A., Mobarry, C.M., Reinert, K.H.J., Remington, K.A., Anson, E.L., Bolanos, R.A., Chou, H., Jordan, C.M., Halpern, A.L., Lonardi, S., Beasley, E.M., Brandon, R.C., Chen, L., Dunn, P.J., Lai, Z., Liang, Y., Nusskern, D.R., Zhan, M., Zhang, Q., Zheng, X., Rubin, G.M., Adams, M.D., Venter, J.C.: A whole-genome assembly of drosophila. *Science* 287, 2196–2204 (2000)
12. Nurk, S., et al.: Assembling genomes and mini-metagenomes from highly chimeric reads. In: Deng, M., Jiang, R., Sun, F., Zhang, X. (eds.) RECOMB 2013. LNCS, vol. 7821, pp. 158–170. Springer, Heidelberg (2013)
13. Pevzner, P.A., Tang, H., Waterman, M.S.: An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences* 98, 9748–9753 (2001)
14. Pop, M.: Genome assembly reborn: recent computational challenges. *Briefings in Bioinformatics* 10(4), 354–366 (2009)
15. Sahli, M., Shibuya, T.: Arapan-s: a fast and highly accurate whole-genome assembly software for viruses and small genomes. *BMC Research Notes* 5(243) (2012)
16. Simpson, J.T., Wong, K., Jackman, S.D., Schein, J.E., Jones, S.J.: Abyss: a parallel assembler for short read sequence data. *Genome Research* 19, 1117–1123 (2009)
17. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research* 18, 821–829 (2008)

Cerulean: A Hybrid Assembly Using High Throughput Short and Long Reads

Viraj Deshpande¹, Eric D.K. Fung², Son Pham¹, and Vineet Bafna¹

¹ Department of Computer Science & Engineering, University of California, San Diego, CA, USA
vbfna@eng.ucsd.edu

² Bioinformatics Undergraduate Program, Department of Bioengineering, University of California, San Diego, CA, USA

Abstract. Genome assembly using high throughput data with short reads, arguably, remains an unresolvable task in repetitive genomes, since when the length of a repeat exceeds the read length, it becomes difficult to unambiguously connect the flanking regions. The emergence of third generation sequencing (Pacific Biosciences) with long reads enables the opportunity to resolve complicated repeats that could not be resolved by the short read data. However, these long reads have high error rate and it is an uphill task to assemble the genome without using additional high quality short reads. Recently, Koren et al. 2012 [1] proposed an approach to use high quality short reads data to correct these long reads and, thus, make the assembly from long reads possible. However, due to the large size of both dataset (short and long reads), error-correction of these long reads requires excessively high computational resources, even on small bacterial genomes. In this work, instead of error correction of long reads, we first assemble the short reads and later map these long reads on the assembly graph to resolve repeats.

Contribution: We present a hybrid assembly approach that is both computationally effective and produces high quality assemblies. Our algorithm first operates with a simplified version of the assembly graph consisting only of long contigs and gradually improves the assembly by adding smaller contigs in each iteration. In contrast to the state-of-the-art long reads error correction technique, which requires high computational resources and long running time on a supercomputer even for bacterial genome datasets, our software can produce comparable assembly using only a standard desktop in a short running time.

1 Introduction

The advent of high throughput sequencing technologies has generated a lot of interest from the computational perspective of de novo assembly of genomic sequences. A major breakthrough in the massively parallel high-throughput sequencing technologies includes the second generation sequencing platforms including those from Illumina and Life Technologies. These platforms generate

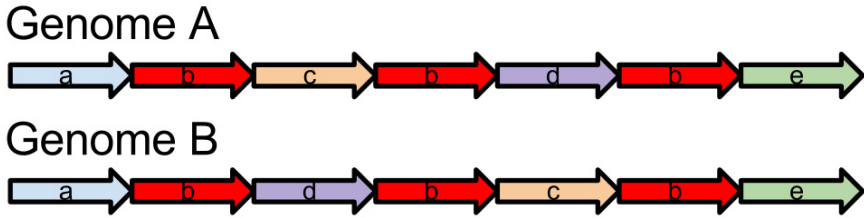


Fig. 1. Two possible genomic architectures, Genome A and Genome B each with a large repeat b. Both putative genomes can generate the same set of paired-end reads. With the information from paired-end reads, the assembler can only identify shorter contigs a, b, c, d and e but not the entire true genome.

paired-end reads with length of the order of 100 or 250 base pairs. The mean end-to-end distance between the paired-end reads, called the *insert size*, is around 300 to 500 base-pairs. The paired-end reads can be sequenced with high accuracy and high depth of coverage. Two approaches are broadly used by assembly tools to assemble the paired-end reads into complete genomes of genomic contigs: (i) Overlap-layout-consensus assembly was introduced by Staden [2], which was later improvised by the introduction of string graph by Myers [3] (Celera Assembler [4], SGA [5]); (ii) de Bruijn graph-based assembly was originally proposed by Idury and Waterman [6] and extended by Pevzner et al [7] (Euler-SR [8], ABySS [9], Velvet [10]). These de novo assembly approaches can generate high quality assemblies using the high quality paired-end reads. However, these paired-end reads are unable to span large repeats and as a result the assembled contigs are often short. Figure 1 shows an example where the assembler is unable to identify the true genome architecture using only short read.

Newer high throughput sequencing platforms [11] target the length limitation of second generation short reads by generating libraries with a large span. The prominent technologies include: (i) jumping libraries which generate small mate-pair reads of around 150 base pairs and variable span of 5 kbp or more; (ii) long reads from Pacific Biosciences with variable length ranging from 1 kbp to 20 kbp and (iii) genomic fragments amplified by Moleculo technology (size: 1.5 kbp to 15 kbp [12]) and then sequenced using short read sequencing technologies. In this work we focus on PacBio long reads generated by directly sequencing entire genomic fragments. In the rest of this article we will use the term *short reads* to describe the paired-end reads from Illumina and *long reads* to describe reads generated using the Pacific Biosciences RS platform.

Long reads generated from PacBio RS can easily span most repeats and have the potential to produce very large assembled contigs. Unfortunately, these reads can have a very high error rate with mean error rate as high as 16%. Hence they are difficult to assemble by themselves and require very high coverage; the assembly quality falls rapidly with smaller coverage [13]. However, combining the high quality of second generation short reads and the large length of the

long reads, these datasets can be processed simultaneously to produce very long genomic contigs that otherwise required costly low-throughput techniques.

Recent efforts (PacbioToCA [1], LSC [14]) have focused on mapping short reads to the erroneous long reads to correct the long reads using aligners like NovoAlign [15] and GMAP [16] which can allow large edit distance for the mapping. These corrected long reads are then used to generate an assembly with significantly longer contigs. However, such mapping from all short reads to all long reads with large edit-distance is computationally expensive and requires a large running time even for small bacterial datasets. Furthermore, if there are two or more similar regions in the genome, the short reads from one region can still map to long reads from the other region given the high edit-distance. Reads corrected in such fashion may create spurious adjacencies leading to misassemblies.

An alternative approach is to first assemble the high coverage short read dataset to produce high quality contigs and use long reads for scaffolding. Previous tools that use this approach include AHA scaffolder [17] and ALLPATHS-LG [18]. However, these approaches are specialized to perform hybrid assembly in the presence additional libraries including Roche 454 reads for AHA scaffolder and jumping libraries of mate-pair reads for ALLPATHS-LG. Following this approach, a hybrid assembler will essentially take as input: the assembler should find the correct traversal of genome on the graph using the support information from the mapping of long reads.

While the alignments of long reads on the complete genome can be easily identified using BLASR [19], alignments to shorter contigs can be spurious. This is because we have to allow very short alignments and cannot conclusively say if these are true alignments or accidental alignments due to short repeats and similar-looking regions. Such alignments generate ambiguous adjacencies between contigs and stop us from making high confidence calls while scaffolding. Furthermore, as we see in Figure 2, the longer assembled contigs tend to be unique in the genome whereas the shorter contigs tend to repeat. If such short contigs occur adjacent to each other in the genome, then using long reads to determine the exact layout of all the contigs in the presence of spurious alignments becomes a difficult problem.

Contribution: In this work, we present *Cerulean*, a completely automated hybrid assembly approach to produce a high quality scaffolds using Illumina paired-end reads and PacBio long reads. Cerulean does not use the short reads directly; instead it works with an *assembly graph* structure generated from short read data using existing assemblers. Assembly graphs are graphs where nodes correspond to contigs of assembled short reads and edges represent putative adjacencies of the contigs, but not confined to overlapping contigs. Such assembly graph are commonly built using overlap-layout-consensus and more recently with the de Bruijn graph paradigm. The input to Cerulean includes: (i) the *assembly graph* generated by ABySS paired-end read assembler [9] constructed from short reads (and it can be applied to graphs generated by overlap graph or de Bruijn graph based assemblers as long as the graph has the desired format) and (ii) the mapping of long reads to the assembled contigs. The output of Cerulean is an

simplified representation of the unentangled assembly graph. The non-branching paths in this simplified graph correspond to the scaffolds in the genome.

We recognize that multiple spurious alignments of long reads to short contigs makes it a difficult problem to unentangle the assembly graph, especially when the short contigs form densely connected substructures. The Cerulean algorithm addresses this problem through an iterative framework to identify and extend high confidence genomic paths. Cerulean initially operates with a simplified representation of the assembly graph, which we call the *skeleton graph* (Figure 3(a)), consisting only of long contigs. We then gradually improve the assembly by adding smaller contigs to the skeleton graph in each iteration.

Our software produced higher quality assembly than the state-of-the-art software for hybrid assembly (PacbioToCA, AHA) in much shorter running time and lower memory usage. While assembly of Cerulean was significantly better than AHA scaffolder; PacbioToCA required 8 hours to run on a supercomputer with 24 threads for a bacterial genome dataset with a total memory usage of 55GB and temporary files of 300GB. In contrast, Cerulean finished within few minutes on a single thread on a regular desktop with memory usage of 100MB and pre-processing (ABYSS, BLASR) taking less than an hour on a desktop computer. Starting with N75 of 60 kbp of contigs generated by ABYSS, scaffolds generates scaffolds with N75 of 503 kbp as compared to 247 kbp for PacbioToCA and 106 kbp for AHA scaffolder.

2 Methods

Inputs: The inputs to Cerulean include (i) the assembly graph and contig sequences from short read assembly (using ABYSS or other assemblers) and (ii) alignments of long reads to contigs from the assembly graph (using BLASR). The assembly graph consists of one vertex for each contig and a conjugate vertex for its reverse complement. The length of a vertex corresponds to the length of the contig. A directed edge between two vertices indicates a putative adjacency between the two vertices. For every directed edge, the conjugate edge is a directed edge from the conjugate of its sink to the conjugate of its source. For each edge, we define the length to be the offset between the end of source contig and start of the sink contig. Thus, if the two contigs overlap, then the length is a negative number; in case of a gap this length is a positive number. The size of the overlap or the gap may depend on the short read assembler, e.g., if the assembler just produces the de Bruijn graph, then the overlap is directly determined by k -mer size, but many short read assemblers also implement preliminary analysis of the graph structure and so the assembled contigs may even overlap by a few thousand base pairs even though the paired-end insert size is only few hundred based pairs. Henceforth, *contigs* refer to DNA sequences assembled by the short read assembler and *scaffolds* refer chain of contigs glued together using alignments of long reads to contigs.

Pipeline: The contigs generated by the short paired-end read assembly have a large distribution of lengths and some of these contigs repeat multiple times in

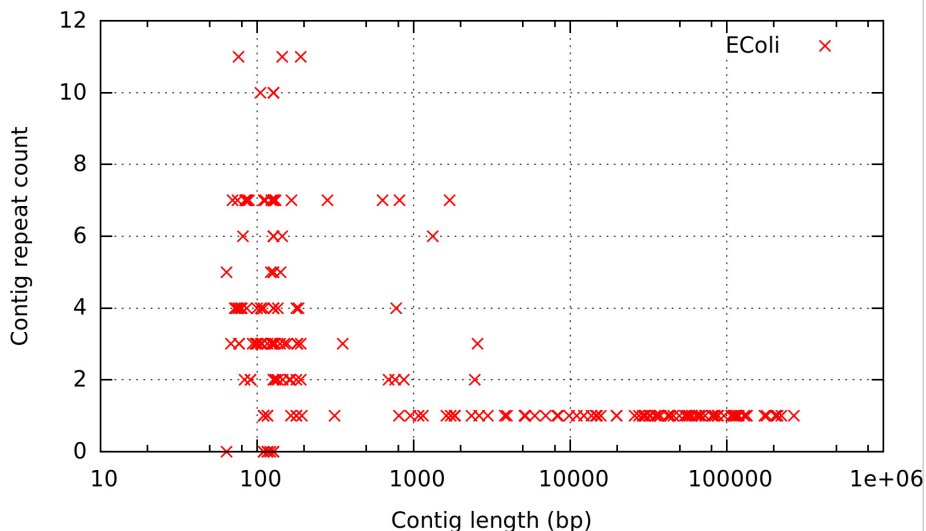


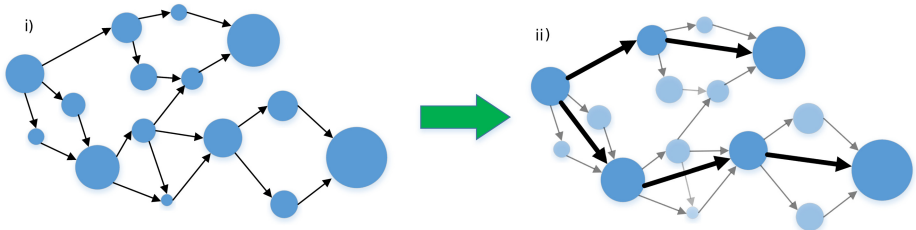
Fig. 2. Distribution of repeat count of contigs assembled from short reads

the reference. Most repeats tend to be short and there are very few long contigs which occur multiple times in the reference as shown in Figure 2 for E. Coli dataset. Resolution of the assembly graph in the presence of these short and repetitive contigs is difficult since they create noise in mapping (spurious alignments) and may form dense structures in the graph which is a major obstacle for the repeat resolution procedure.

Our algorithm relies on the construction of a *skeleton graph* (Figure 3(a)) which is a simplified representation of the assembly graph containing only long contigs. The edges in the skeleton graph represent the putative genomic connections of these contigs. *As we shall see below, we include an edge in the skeleton graph only if there is sufficient number of long reads that indicate the corresponding adjacency.* Since the skeleton graph has a simple structure consisting only of long contigs, mapping long reads to the skeleton graph has less noise and repeat resolution is simpler. Our approach gradually improves the assembly by adding smaller contigs in every iteration.

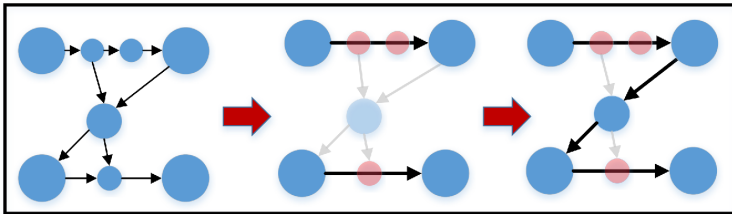
In each iteration, the algorithm goes through three components (Figure 3(b)): I) Skeleton graph construction/extension; II) Repeat Resolution ; III) Gap bridging.

Skeleton Graph Construction: Given the assembly graph $G(V, E)$, genome S , a length threshold L , the skeleton graph $SG(G, S, L) = G(V', E')$ has vertex set V' containing only vertices corresponding to contigs longer than L in V . An edge $(v'_i, v'_j) \in E'$ depicts a putative adjacent layout between the corresponding contigs in the genome. The skeleton graph represents a simplification of the original assembly graph by ignoring all intermediate short contigs in the assembly graph. The short contigs that occur between consecutive long contigs in the genome are implicitly included by annotating the relevant edges.



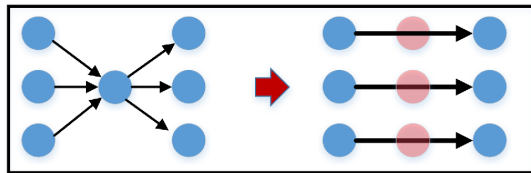
(a) (i) Assembly graph (ii) Skeleton graph retains only big vertices (long contigs)

i) Iterative Skeleton Graph

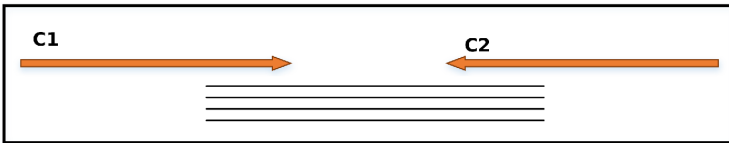


Vertex Length Threshold

ii) Repeat Resolution



iii) Gap Filling



(b) Iterative Pipeline for Cerulean: (i) Skeleton graph reconstruction (Circle size proportional to contig length) (ii) Repeat resolution (iii) Gap filling.

Fig. 3. Skeleton graph representation and iterative construction of approximate skeleton graph

Since the skeleton graph is a simpler graph with long vertices, unambiguously mapping the long reads to long contigs and resolving repeats in the skeleton graph is more favorable than in the assembly graph. However, since the genome is unknown, the skeleton graph can not always be constructed in its entirety. Below, we construct an approximate version of skeleton graph using the information from all long read alignments.

The first iteration of the skeleton graph is constructed by using only long contigs from the assembly graph as vertices. Alignments of pairs of long contigs to a long read imply a certain distance (overlap or gap) between the contigs if these were true alignments. A directed edge is added between 2 vertices if there exists a path (or edge) in the assembly graph that certifies the implied distance within certain tolerance. The length of the edge is defined as the distance inferred by following the path (in the assembly graph) rather than the distance inferred from erroneous long reads. In case the adjacent contigs overlap in the assembly graph, then the long reads need to span the entire overlap to include a putative edge between the two contigs in the skeleton graph.

We further refine this approximate graph by the following steps:

Read Count Threshold: We should only keep those edges which have a significant long read support. The length of the long reads is variable. So if we want to resolve a long repeat or fill a large gap, then we will expect a small number of long reads to span the entire repeat or gap. Thus, the expected number of long reads that connect two contigs will depend on the coverage as well as the distance between the two contigs and the read length distribution of the long reads. We thus evaluate the significance of the number of long reads supporting an edge in comparison to support for other competing edges. Our criteria for adding new edges consists of three parts: (i) if the number of long reads supporting an edge is greater than a high confidence threshold that edge is certainly retained; (ii) if the number of supporting long reads is less than a certain low confidence threshold, then such an edge is discarded; (iii) if long read count is between these thresholds, an edge is included either if it is the only outgoing/incoming edge for the source/sink, or if the read count for the edge is significantly higher than other edges incident on source or sink.

Length-sensitive Transitive Edge Reduction: We can identify the high confidence scaffolds by looking at non-branching paths in the skeleton graph. However, some chains of contigs which ideally should form non-branching paths in the graph may get connected to vertices beyond their immediate neighbours. Such cases can happen when some long reads do not align to the intermediate vertex due to errors. For identification of non-branching paths, we remove the transitive edges which create the false impression of a branch (Figure 4). An edge is defined to be transitive if there is an alternative path from source to sink implying the same distance offset.

Repeat Resolution: The repeat resolution procedure is illustrated in Figure 3(b) (ii). When we remove the transitive edges to identify simple paths, we may lose valuable information from long reads that span repeats appearing as branching

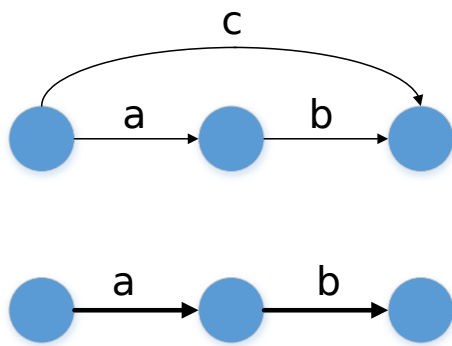


Fig. 4. Length-sensitive transitive edge reduction along a non-branching path

nodes in the graph. A branching node will have multiple incoming edges on one side on multiple outgoing edges on the other. The corresponding incoming and outgoing vertices form a bipartite structure as seen in Figure 3(b)(ii). We resolve such repeats by matching pairs of vertices (not necessarily all pairs) in this bipartite structure by looking at reads spanning the repeat. Matched pairs which can be joined unambiguously are then connected by a direct edge (with annotation) and the edges to the intermediate vertex are removed. We do repeat resolution in its simplest form of spanning only one contig at a time. This works if the repeating contig is a maximal repeat. It is the common case for most contigs in our procedure to be maximal repeats since we deal with only long contigs initially. But as we scale to larger genomes with more complex repeat structures of shorter contigs, we need to implement more powerful strategies for spanning repeats.

Gap Bridging: The gap bridging procedure is illustrated in Figure 3(b)(iii). After using all information from connections in the graph to identify the scaffolds, we observe that certain paths terminate as there are no further edges to extend these paths. At this point, we relax the constraint that edges in the skeleton graph should correspond to existing paths of the assembly graph. We can identify possible ways to extend a scaffold by looking at long reads that align from the end of that scaffold to the end of another scaffold. If this is the only possible way to extend either of these scaffolds, then we use these long read alignments to unambiguously bridge the gap between these paths (scaffolds) by adding a new edge between their terminal vertices.

Iteration on Vertex Length: After we have inferred all possible scaffolds from the long contig skeleton, we have the list of all simple paths from this skeleton. Now we can decrease the vertex length threshold according to a vertex length schedule. We add the new short contig vertices (longer than the new lower threshold) either if they are connected to the end vertices of the simple paths from the previous vertex length iteration, or if they are connected to other short vertices. Thus non-terminal non-branching long contigs in scaffolds from one iteration are untouched in the next iteration. Then we iteratively go through the above steps of transitive edge reduction, repeat resolution and gap bridging.

Final Assembly: After the completion of all iterations through the vertex length schedule we have our final approximation of the skeleton graph. The simple paths represent our final scaffolds. The edges of these simple paths can either be directed edges in the original assembly graph, or they can be annotated with either path traversals in the assembly graph by the set of long reads bridging a gap. Thus, the final scaffolds can be inferred from these annotated paths on the corresponding sequence of reads that are used for gap filling. Those vertices which are not included in the scaffolds can be inferred as independent contigs.

3 Results

We tested our software for the *Escherichia Coli* bacterial genome (strain: K12, isolate: MG1665). The short [20] and long [21] read datasets were obtained from the samples provided by Illumina and Pacific Biosciences respectively as described in Table 1.

Short read assembly: We assembled the short read contigs using the ABySS paired-end assembler with k -mer size of 64 base pairs. The computational resources and assembly results are mentioned in Table 2 and Table 3 respectively.

Mapping long reads to contigs: We mapped the long reads to the ABySS assembled contigs using BLASR with minimum percentage identity of 70%.

Filtering spurious alignments: There are many short alignments from long reads to contigs due to short repeats contained within contigs as shown in Figure 5. Reads mapping to multiple contigs do not necessarily imply adjacency and need to be filtered. We classify an alignment as long alignment if the unaligned overhang (i.e. length of unaligned portion of the read which ideally should have mapped in case of a true alignment of unerroneous read) is less than 30% of the ideal alignment length (i.e. sum of unaligned overhang and aligned portion). Henceforth, when we refer to alignment of a long read it means it satisfies the criteria for long alignment.

Cerulean scaffolding: We used the ABySS assembly graph and filtered BLASR alignments to generate the scaffolds. The vertex length schedule we used is 2048 bp, 1024 bp, 512 bp, 256 bp and 0 bp.

PacbioToCA: We compare our results to the assembly generated by the alternative approach of assembling error-corrected long reads using PacbioToCA.

AHA scaffold: We also tested the results of the AHA scaffold using the ABySS assembled contigs and PacBio long reads as inputs.

ALLPATHS-LG: We did not test ALLPATHS-LG since it requires jumping libraries.

Table 1. Details of sequencing data used for assembly

Platform	Illumina Hiseq	PacBio RS
Coverage	400X	30x
Read Length	151bpX2 (insert size 300bp)	N50: 5900, Largest:19416
Number of reads	11 million	75152

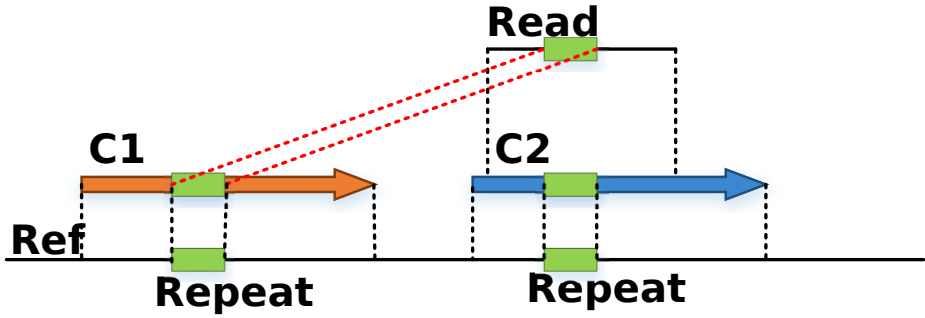


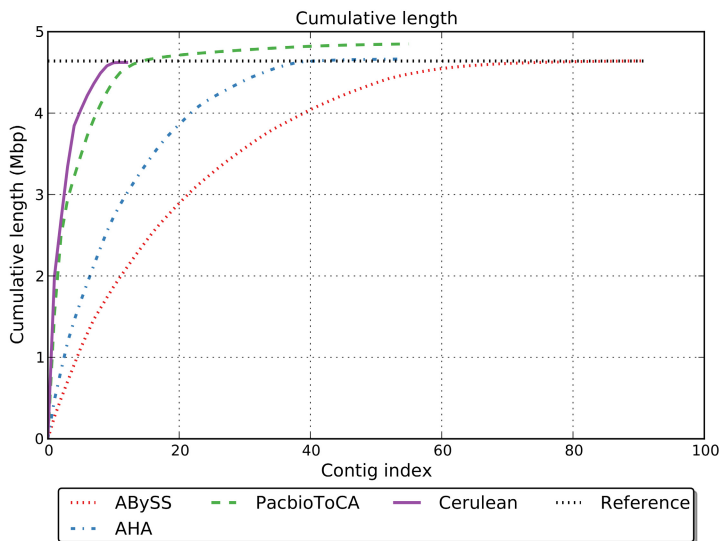
Fig. 5. Spurious alignments of long reads due to short repeats are filtered.

The length distribution comparison of the assembled contigs is displayed in Figures 3 and 6(b). We can see that the N50 values for the PacbioToCA assembly is determined solely by the first 2 contigs, but the contig length drops drastically after that giving a very low N75 value of 273 Kbp as compared to N50 of 957 Kbp. The length of the scaffolds generated by Cerulean falls much slower giving a significantly better N75 of 503 Kbp comparable to the N50 length 694 Kbp. Figure 3 also shows that the total assembled length for PacbioToCA is significantly larger than the genome length.

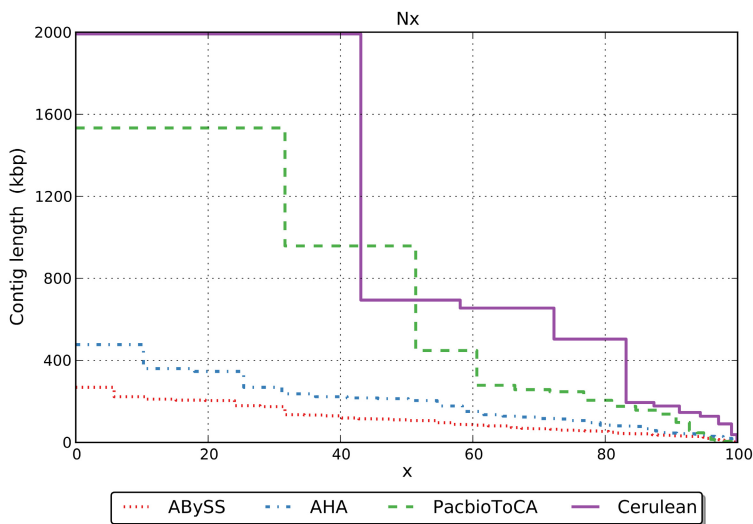
Table 2. Computational resources for hybrid assembly

Software	ABYSS	BLASR	Cerulean	PacbioToCA	AHA
Number of threads	1	8	1	24	4
Peak memory usage	<4 GB	300 MB	100 MB	55 GB	300 MB
Runtime	<30 mins	<30 mins	2 mins	12 hours	2 hours
Temporary files	75 MB	8 MB	5 MB	300 GB	500 MB

Analysis of the final set of contigs indicated that all the 11 long contigs were essentially separated by just 2 non-exact repeats in the reference of lengths (2500 bp and 4300 bp). If we are more aggressive in resolving these repeats, then this approach has the potential to retrieve the entire genome as a single contig. Our conservative adjacency calls did not resolve these repeats in order to retain the accuracy of the assembly. We chose not to make more aggressive decisions in repeat resolution in the dataset because in the case of just 2 repeats, it is easy to implement a scheme that will overfit the data and not scale to other genomes when running in a fully automated setting.



(a) Cumulative length distribution of all contigs/scaffolds arranged in decreasing order of length



(b) Nx length of all contigs/scaffolds of scaffolds arranged in decreasing order of length

Fig. 6. Comparison of lengths of contigs/scaffolds generated by various approaches

Table 3. Assembly statistics for hybrid assembly analyzed using QuAsT [22]. We have separately validated the sequential order and offsets of all long contigs forming the scaffold by mapping them to the reference. We also confirmed that all 4 reported misassemblies for ABySS + Cerulean were actually local misassemblies of a small length ($< 1Kbp$) and 1 fake misassembly due to circular genome. However, a significant number of misassemblies in PacbioToCA and AHA involved relocations/inversions of significant number of contigs longer than 1Kbp and as large as 30 Kbp and 19 Kbp respectively.

Software	Reference	ABySS	Cerulean	PacbioToCA	AHA
# contigs	1	199	21	55	54
# contigs > 1000bp	1	83	11	55	48
N50	4639675	110Kbp	694Kbp	950Kbp	213 Kbp
N75	4639675	64Kbp	507Kbp	247 Kbp	107 Kbp
Largest contig length	4639675	268969	1991897	1533073	477080
Total length	4639675	4849724	4625935	4641287	4663300
#misassemblies	-	3	4	22	11

4 Discussions

Cerulean has a very low resource usage and high accuracy of assembled scaffolds. This makes a very strong case for scaling this approach to larger genomes. The algorithm in its current state focuses on making decisions based on very simple building blocks one at a time. This makes it possible for us to make low risk decisions towards a high accuracy assembly for simple bacterial genomes. However, when analyzing datasets from larger complex genome, we have no prior knowledge of the structure of the repeats and the layout of the contigs generated by short read assemblers. So there are cases where the scaffolding algorithm may not be able to distinguish between a true adjacency signal and a false adjacency signal. In most cases, this will simply stop the algorithm from extending a scaffold due to branching. However, we cannot conclusively rule out the possibility of producing other side effects for every decision made by the algorithm. We also need to acknowledge the fact that we are currently dealing with small bacterial genomes for which we can easily obtain high and more or less uniform coverage for short reads. So far we rely completely on the short read assembler to generate the initial contigs. However, for larger genomes, variable coverage caused due to sequencing bias combined with decisions made by the short read assembler can cause misassembled contigs to start with. In this case, the scaffolder will benefit from not assuming the assembled contigs as ground truth, but actually testing for misassemblies by the short read assembler.

However, the framework of gradual inclusion of complexity does provide us with the opportunity to tackle even more complex genomes in a systematic fashion. Here we discuss a few of these cases where this framework is useful. When extending the scaffolds consisting of large contigs, we aim to extend them unambiguously with the inherent assumption that the larger contigs are usually

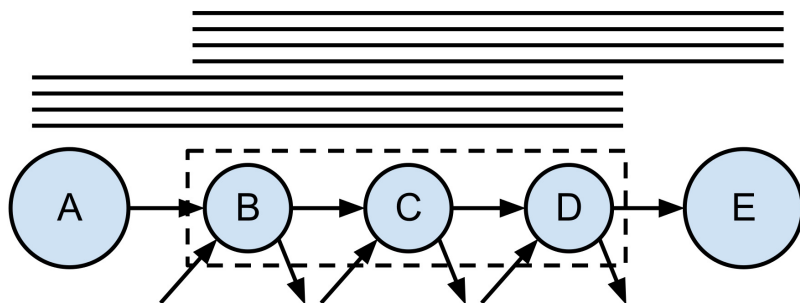


Fig. 7. Example for resolving multiple consecutive repeats. Contigs A and E are not connected directly by long reads. The intermediate vertices B, C and D all have branches however, the triplet $B \rightarrow C \rightarrow D$ only occurs at a unique location. Thus reads which span all 3 vertices B, C and D simultaneously can be treated as aligning to one large combined contig (3-gram) and used to extend the scaffold from A to E.

unique in the reference. We can increase the confidence in this assumption by considering the coverage information of contigs to estimate the repeat counts of contigs and make our decision for extension based on this assumption. Furthermore, if in one iteration of the vertex length schedule, we find an unambiguous way of extending a scaffold, it does not necessarily imply that it is the only way to extend and there can possibly be other ways to extend which require looking at shorter vertices for bridging. One way to address problem is by using the vertex length threshold as a soft cutoff rather than a hard cutoff to allow shorter contigs to compete with the larger contigs if they have very good support from the reads and graph structure.

In Cerulean, we bypass the problems of complex repetitive structures in graph, by only looking at uniquely occurring long vertices. We resolve only one branching vertex at a time under the inherent assumption that most long repeats are maximal repeats isolated from other repeats of comparable length. Larger genomes certainly violate this assumption to a large extent. Thus we may have to connect scaffolds that are separated by multiple branching vertices of comparable lengths. There is also the opportunity to exploit the structure of the graph in extending the scaffolds as shown in Figure 7. A path tracing approach can help us extend scaffolds across such multiple branching vertices. Path tracing is a non-trivial problem in big graphs, but we can use path tracing in the context of our incremental framework to solve specific small problems.

Finally there are techniques we can use to improve our ability to extend scaffolds or add more edges to the skeleton graph. An edge in the skeleton graph can correspond to a walk in the assembly graph. The input mapping from long reads to contigs may miss some alignments from the long reads to intermediate contigs due to high error rate. In such a case, we can perform a more informed search for such a walk by looking at local alignments of reads along neighboring contigs. If we can identify true chains of small contigs, then we can rerun the

mapping the reads to the concatenated sequences of these chains and use these mappings while extending existing scaffolds. PBJelly [23] has displayed that gaps in scaffolds can be filled with high accuracy using the mapping reads. Thus while retrieving the the intermediate sequences of scaffolds, we can use a combination of long reads and assembled contigs to bridge the long contigs. After we have finished making the most conservative calls using the assembled short read contigs and filled in all gaps using the pairwise alignments of the long reads, we can be more ambitious and bridge the gaps between the scaffolds with targeted assembly of long reads.

In conclusion, we present a hybrid assembly approach that is both computationally effective and produces high quality assemblies. Our algorithm first operates with a simplified version of the assembly graph consisting only of long contigs and gradually improve the assembly by adding smaller contigs in each iteration. In contrast to the state-of-the-art long reads error correction technique, which requires high computational resources and long running time on a supercomputer even for bacterial genome datasets, our software can produce comparable assembly using only a standard desktop in a short running time.

Acknowledgments. The authors will like to sincerely thank Pavel Pevzner and Glenn Tesler for their insightful comments. V.D. and V.B. were supported in part by NIH grants 5RO1-HG004962 and U54 HL108460, and by the NSF grant NSF-CCF-1115206. S.P. was supported in part by NIH grant 3P41RR024851-02S1.

Disclosure Statement

No competing financial interests exist.

References

1. Koren, S., Schatz, M.C., Walenz, B.P., Martin, J., Howard, J.T., Ganapathy, G., Wang, Z., Rasko, D.A., McCombie, W.R., Jarvis, E.D., et al.: Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology* 30(7), 693–700 (2012)
2. Staden, R.: A strategy of dna sequencing employing computer programs. *Nucleic Acids Research* 6(7), 2601–2610 (1979)
3. Myers, E.W.: The fragment assembly string graph. *Bioinformatics* 21(suppl. 2), ii79–ii85 (2005)
4. Myers, E.W., Sutton, G.G., Delcher, A.L., Dew, I.M., Fasulo, D.P., Flanigan, M.J., Kravitz, S.A., Mobarry, C.M., Reinert, K.H., Remington, K.A., et al.: A whole-genome assembly of drosophila. *Science* 287(5461), 2196–2204 (2000)
5. Simpson, J.T., Durbin, R.: Efficient de novo assembly of large genomes using compressed data structures. *Genome Research* 22(3), 549–556 (2012)
6. Idury, R.M., Waterman, M.S.: A new algorithm for dna sequence assembly. *Journal of Computational Biology* 2(2), 291–306 (1995)

7. Pevzner, P.A., Tang, H., Waterman, M.S.: An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences* 98(17), 9748–9753 (2001)
8. Chaisson, M.J., Pevzner, P.A.: Short read fragment assembly of bacterial genomes. *Genome Research* 18(2), 324–330 (2008)
9. Simpson, J.T., Wong, K., Jackman, S.D., Schein, J.E., Jones, S.J., Birol, Ī.: Abyss: a parallel assembler for short read sequence data. *Genome Research* 19(6), 1117–1123 (2009)
10. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research* 18(5), 821–829 (2008)
11. Eisenstein, M.: Companies' going long generate buzz at marco island. *Nature Biotechnology* 31(4), 265–266 (2013)
12. Waldbieser, G.: Production of long (1.5 kb–15.0 kb), accurate, dna sequencing reads using an illumina hiseq2000 to support de novo assembly of the blue catfish genome. In: *Plant and Animal Genome XXI Conference, Plant and Animal Genome* (2013)
13. Chin, C.S., Alexander, D.H., Marks, P., Klammer, A.A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E.E., et al.: Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature Methods* (2013)
14. Au, K.F., Underwood, J.G., Lee, L., Wong, W.H.: Improving pacbio long read accuracy by short read alignment. *PLoS One* 7(10), e46679 (2012)
15. Hercus, C.: Novocraft short read alignment package (2009), <http://www.novocraft.com>
16. Wu, T.D., Watanabe, C.K.: Gmap: a genomic mapping and alignment program for mrna and est sequences. *Bioinformatics* 21(9), 1859–1875 (2005)
17. Bashir, A., Klammer, A.A., Robins, W.P., Chin, C.S., Webster, D., Paxinos, E., Hsu, D., Ashby, M., Wang, S., Peluso, P., et al.: A hybrid approach for the automated finishing of bacterial genomes. *Nature Biotechnology* (2012)
18. Ribeiro, F.J., Przybylski, D., Yin, S., Sharpe, T., Gnerre, S., Abouelleil, A., Berlin, A.M., Montmayeur, A., Shea, T.P., Walker, B.J., et al.: Finished bacterial genomes from shotgun sequence data. *Genome Research* 22(11), 2270–2277 (2012)
19. Chaisson, M.J., Tesler, G.: Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC Bioinformatics* 13(1), 238 (2012)
20. E.Coli MG1655 Illumina HiSeq2000 sequencing dataset, ftp://webdata.webdata@ussd-ftp.illumina.com/Data/SequencingRuns/MG1655/MiSeq_Ecoli_MG1655_110721_PF.bam (2013) (online; accessed June 24, 2013)
21. E.Coli K12 MG1655 Pacbio RS sequencing dataset (2013), <http://files.pacb.com/datasets/primary-analysis/e-coli-k12/1.3.0/e-coli-k12-mg1655-raw-reads-1.3.0.tgz> (online; accessed June 24, 2013)
22. Schmutz, J., Wheeler, J., Grimwood, J., Dickson, M., Yang, J., Caoile, C., Bajorek, E., Black, S., Chan, Y.M., Denys, M., et al.: Quality assessment of the human genome sequence. *Nature* 429(6990), 365–368 (2004)
23. English, A.C., Richards, S., Han, Y., Wang, M., Vee, V., Qu, J., Qin, X., Muzny, D.M., Reid, J.G., Worley, K.C., et al.: Mind the gap: Upgrading genomes with pacific biosciences rs long-read sequencing technology. *PLoS One* 7(11), e47768 (2012)

Using Cascading Bloom Filters to Improve the Memory Usage for de Bruijn Graphs

Kamil Salikhov¹, Gustavo Sacomoto^{2,3}, and Gregory Kucherov^{4,5}

¹ Lomonosov Moscow State University, Moscow, Russia
salikhov.kamil@gmail.com

² INRIA Grenoble Rhône-Alpes, France
gustavo.sacomoto@inria.fr

³ Laboratoire Biométrie et Biologie Evolutive, Université Lyon 1, Lyon, France

⁴ Department of Computer Science, Ben-Gurion University of the Negev,
Be'er Sheva, Israel

⁵ Laboratoire d'Informatique Gaspard Monge, Université Paris-Est & CNRS,
Marne-la-Vallée, Paris, France
Gregory.Kucherov@univ-mlv.fr

Abstract. De Bruijn graphs are widely used in bioinformatics for processing next-generation sequencing (NGS) data. Due to the very large size of NGS datasets, it is essential to represent de Bruijn graphs compactly, and several approaches to this problem have been proposed recently. In this work, we show how to reduce the memory required by the algorithm of Chikhi and Rizk (WABI, 2012) that represents de Bruijn graphs using Bloom filters. Our method requires 30% to 40% less memory with respect to their method, with insignificant impact to construction time. At the same time, our experiments showed a better query time compared to their method. This is, to our knowledge, the best *practical* representation for de Bruijn graphs.

1 Introduction

Modern next-generation sequencing (NGS) technologies generate huge volumes of short nucleotide sequences (*reads*) drawn from the DNA sample under study. The length of a read varies from 35 to about 400 base pairs (letters) and the number of reads may be hundreds of millions, thus the total volume of data may reach tens or even hundreds of Gb.

Many computational tools dealing with NGS data, especially those devoted to *genome assembly*, are based on the concept of a *de Bruijn graph*, see e.g. [8]. The nodes of the de Bruijn graph¹ are all distinct *k-mers* occurring in the reads, and two *k-mers* are linked by an arc if there is a suffix-prefix overlap of size $k - 1$. The value of k is an open parameter, that in practice is chosen between 20 and 64. The idea of using de Bruijn graph for genome assembly goes back to

¹ Note that this actually a *subgraph* of the de Bruijn graph under its classical combinatorial definition. However, we still call it de Bruijn graph to follow the terminology common to the bioinformatics literature.

the “pre-NGS era” [11]. Note, however, that *de novo* genome assembly is not the only application of those graphs when dealing with NGS data. There are several others, including: *de novo* transcriptome assembly [5] and *de novo* alternative splicing calling [14] from transcriptomic NGS data (RNA-seq); metagenome assembly [10] from metagenomic NGS data; and genomic variant detection [6] from genomic NGS data using a reference genome.

Due to the very large size of NGS datasets, it is essential to represent de Bruijn graphs as compactly as possible. This has been a very active line of research. Recently, several papers have been published that propose different approaches to compressing de Bruijn graphs [4,15,3,2,9].

Conway and Bromage [4] proposed a method based on classical succinct data structures, i.e. bitmaps with efficient rank/select operations. On the same direction, Bowe *et al.* [2] proposed a very interesting succinct representation that, assuming only one string (read) is present, uses only $4m$ bits, where m is the number of arcs in the graph. The more realistic case, where there are M reads, can be easily reduced to the one string case by concatenating all M reads using a special separator character. However, in this case the size of the structure is $4m + O(M \log m)$ bits ([2], Theorem 1). Since the multiplicative constant of the second term is hidden by the asymptotic notation, it is hard to know precisely what would be the size of this structure in practice.

Ye *et al.* [15] proposed a different method based on a sparse representation of de Bruijn graphs, where only a subset of k -mers present in the dataset are stored. Pell *et al.* [9] proposed a method to represent it approximately, the so called *probabilistic de Bruijn graph*. In their representation a node have a small probability to be a false positive, i.e. the k -mer is not present in the dataset. Finally, Chikhi and Rizk [3] improved Pell’s scheme in order to obtain an exact representation of the de Bruijn graph. This was, to our knowledge, the best *practical* representation of an exact de Bruijn graph.

In this work, we focus on the method proposed in [3] which is based on Bloom filters. They were first used in [9] to provide a very space-efficient representation of a subset of a given set (in our case, a subset of k -mers), at the price of allowing *one-sided errors*, namely *false positives*. The method of [3] is based on the following idea: if all queried nodes (k -mers) are only those which are reachable from some node known to belong to the graph, then only a fraction of all false positives can actually occur. Storing these false positives explicitly leads to an exact (false positive free) and space-efficient representation of the de Bruijn graph.

Our contribution is an improvement of this scheme by changing the representation of the set of false positives. We achieve this by iteratively applying a Bloom filter to represent the set of false positives, then the set of “false false positives” etc. We show analytically that this cascade of Bloom filters allows for a considerable further economy of memory, improving the method of [3]. Depending on the value of k , our method requires 30% to 40% less memory with respect to the method of [3]. Moreover, with our method, the memory grows very little as k grows. Finally, we implemented our method and tested it against

[3] on real datasets. The tests confirm the theoretical predictions for the size of structure and show a 20% to 30% *improvement* in query times.

2 Preliminaries

A *Bloom filter* is a space-efficient data structure for representing a given subset of elements $T \subseteq U$, with support for efficient membership queries with one-sided error. That is, if a query for an element $x \in U$ returns *no* then $x \notin T$, but if it returns *yes* then x may or not belong to T , i.e. with small probability $x \notin T$ (false positive). It consists of a bitmap (array of bits) B with size m and a set of p distinct hash functions $\{h_1, \dots, h_p\}$, where $h_i : U \mapsto \{0, \dots, m-1\}$. Initially, all bits of B are set to 0. An insertion of an element $x \in T$ is done by setting the elements of B with indices $h_1(x), \dots, h_p(x)$ to 1, i.e. $B[h_i(x)] = 1$ for all $i \in [1, p]$. The membership queries are done symmetrically, returning *yes* if all $B[h_i(x)]$ are equal 1 and *no* otherwise. As shown in [7], when considering hash functions that yield equally likely positions in the bit array, and for large enough array size m and number of inserted elements n , the false positive rate \mathcal{F} is

$$\mathcal{F} \approx (1 - e^{-pn/m})^p = (1 - e^{-p/r})^p \quad (1)$$

where $r = m/n$ is the number of bits (of the bitmap B) per element (of T represented). It is not hard to see that this expression is minimized when $p = r \ln 2$, giving a false positive rate of

$$\mathcal{F} \approx (1 - e^{-p/r})^p = (1/2)^p \approx 0.6185^r. \quad (2)$$

A *de Bruijn graph*, for a given parameter k , of a set of reads (strings) $\mathcal{R} \subseteq \Sigma^* = \{A, C, T, G\}^*$ is entirely defined by the set $T \subseteq U = \Sigma^k$ of k -mers present in \mathcal{R} . The nodes of the graph are precisely the k -mers of T and for any two vertices $u, v \in T$, there is an arc from u to v if the suffix of u of size $k-1$ is equal to the prefix of v of the same size. Thus, given a set $T \subseteq U$ of k -mers we can represent its de Bruijn graph using a Bloom filter B . This approach has the disadvantage of having false positive nodes, as direct consequence of the false positive queries in the Bloom filter, which can create false connections in the graph (see [9] for the influence of false positive nodes on the topology of the graph). The naive way to remove those false positives nodes, by explicitly storing (e.g. using a hash table) the set of all false positives of B , is clearly inefficient, as the expected number of elements to be explicitly stored is $|U|\mathcal{F} = 4^k\mathcal{F}$.

The key idea of [3] is to explicitly store only a subset of all false positives of B , the so-called *critical false positives*. This is possible because in order to perform an exact (without false positive nodes) graph traversal, only potential neighbors of nodes in T are queried. In other words, the set of critical false positives consists of the potential neighbors of T that are false positives of B , i.e. the k -mers from U that overlap the k -mers from T by $k-1$ letters and are false positives of B . Thus, the size of the set of critical false positives is bounded by $8|T|$, since each node of T has at most $2|\Sigma| = 8$ neighbors (for each node, there are $|\Sigma|$ k -mers overlapping the $k-1$ suffix and $|\Sigma|$ overlapping the $k-1$ prefix). Therefore, the expected number of critical false positives can be upper-estimated by $8|T|\mathcal{F}$.

3 Cascading Bloom Filter

Let \mathcal{R} be a set of reads and T_0 be the set of occurring k -mers (nodes of the de Bruijn graph) that we want to store. As stated in Section 2, the method of [3] stores T_0 via a bitmap B_1 using a Bloom filter, together with the set T_1 of critical false positives. T_1 consists of those k -mers which have a $k - 1$ overlap with k -mers from T_0 but which are stored in B_1 “by mistake”, i.e. belong² to B_1 but not to T_0 . B_1 and T_1 are sufficient to represent the graph provided that the only queried k -mers are those which are potential neighbors of k -mers of T_0 .

The idea we introduce in this work is to use this structure recursively and represent the set T_1 by a new bitmap B_2 and a new set T_2 , then represent T_2 by B_3 and T_3 , and so on. More formally, starting from B_1 and T_1 defined as above, we define a series of bitmaps B_1, B_2, \dots and a series of sets T_1, T_2, \dots as follows. B_2 stores the set of false positives T_1 using another Bloom filter, and the set T_2 contains the critical false positives of B_2 , i.e. “true nodes” from T_0 that are stored in B_2 “by mistake” (we call them **false**² positives). B_3 and T_3 , and, generally, B_i and T_i are defined similarly: B_i stores k -mers of T_{i-1} using a Bloom filter, and T_i contains k -mers stored in B_i “by mistake”, i.e. those k -mers that do not belong to T_{i-1} but belong to T_{i-2} (we call them **false** ^{i} positives). Observe that $T_0 \cap T_1 = \emptyset$, $T_0 \supseteq T_2 \supseteq T_4 \dots$ and $T_1 \supseteq T_3 \supseteq T_5 \dots$

The following lemma shows that the construction is correct, that is it allows one to verify whether or not a given k -mer belongs to the set T_0 .

Lemma 1. *Given a k -mer (node) K , consider the smallest i such that $K \notin B_{i+1}$ (if $K \notin B_1$, we define $i = 0$). Then, if i is odd, then $K \in T_0$, and if i is even (including 0), then $K \notin T_0$.*

Proof. Observe that $K \notin B_{i+1}$ implies $K \notin T_i$ by the basic property of Bloom filters that membership queries have one-sided error, i.e. there are no false negatives. We first check the Lemma for $i = 0, 1$.

For $i = 0$, we have $K \notin B_1$, and then $K \notin T_0$.

For $i = 1$, we have $K \in B_1$ but $K \notin B_2$. The latter implies that $K \notin T_1$, and then K must be a false² positive, that is $K \in T_0$. Note that here we use the fact that the only queried k -mers K are either nodes of T_0 or their neighbors in the graph (see [3]), and therefore if $K \in B_1$ and $K \notin T_0$ then $K \in T_1$.

For the general case $i \geq 2$, we show by induction that $K \in T_{i-1}$. Indeed, $K \in B_1 \cap \dots \cap B_i$ implies $K \in T_{i-1} \cup T_i$ (which, again, is easily seen by induction), and $K \notin B_{i+1}$ implies $K \notin T_i$.

Since $T_{i-1} \subseteq T_0$ for odd i , and $T_{i-1} \subseteq T_1$ for even i (for $T_0 \cap T_1 = \emptyset$), the lemma follows.

Naturally, the lemma provides an algorithm to check if a given k -mer K belongs to the graph: it suffices to check successively if it belongs to B_1, B_2, \dots until we encounter the first B_{i+1} which does not contain K . Then, the answer will simply depend on whether i is even or odd: K belongs to the graph if and only if i is odd.

² By a slight abuse of language, we say that “an element belongs to B_j ” if it is accepted by the corresponding Bloom filter.

In our reasoning so far, we assumed an infinite number of bitmaps B_i . Of course, in practice we cannot store infinitely many (and even simply many) bitmaps. Therefore, we “truncate” the construction at some step t and store a finite set of bitmaps B_1, B_2, \dots, B_t together with an explicit representation of T_t . The procedure of Lemma 1 is extended in the obvious way: if for all $1 \leq i \leq t$, $K \in B_i$, then the answer is determined by directly checking $K \in T_t$.

4 Memory and Time Usage

First, we estimate the memory needed by our data structure, under the assumption of an infinite number of bitmaps. Let N be the number of “true positives”, i.e. nodes of T_0 . As stated in Section 2, if T_0 has to be stored via a bitmap B_1 of size rN , the false positive rate can be estimated as c^r , where $c = 0.6185$. And, the expected number of critical false positive nodes (set T_1) has been estimated in [3] to be $8Nc^r$, as every node has eight extensions, i.e. potential neighbors in the graph. We slightly refine this estimation to $6Nc^r$ by noticing that for most of the graph nodes, two out of these eight extensions belong to T_0 (are real nodes) and thus only six are potential false positives. Furthermore, to store these $6Nc^r$ critical false positive nodes, we use a bitmap B_2 of size $6rNc^r$. Bitmap B_3 is used for storing nodes of T_0 which are stored in B_2 “by mistake” (set T_2). We estimate the number of these nodes as the fraction c^r (false positive rate of filter B_2) of N (size of T_0), that is Nc^r . Similarly, the number of nodes we need to put to B_4 is $6Nc^r$ multiplied by c^r , i.e. $6Nc^{2r}$. Keeping counting in this way, the memory needed for the whole structure is $rN + 6rNc^r + rNc^r + 6rNc^{2r} + rNc^{2r} + \dots$ bits. The number of bits per k -mer is then

$$r + 6rc^r + rc^r + 6rc^{2r} + \dots = (r + 6rc^r)(1 + c^r + c^{2r} + \dots) = (1 + 6c^r) \frac{r}{1 - c^r}. \quad (3)$$

A simple calculation shows that the minimum of this expression is achieved when $r = 5.464$, and then the minimum memory used per k -mer is 8.45 bits.

As mentioned earlier, in practice we store only a finite number of bitmaps B_1, \dots, B_t together with an explicit representation (such as array or hash table) of T_t . In this case, the memory taken by the bitmaps is a truncated sum $rN + 6rNc^r + rNc^r + \dots$, and a data structure storing T_t takes either $2k \cdot Nc^{\lceil \frac{t}{2} \rceil r}$ or $2k \cdot 6Nc^{\lceil \frac{t}{2} \rceil r}$ bits, depending on whether t is even or odd. The latter follows from the observations that we need to store $Nc^{\lceil \frac{t}{2} \rceil r}$ (or $6rNc^{\lceil \frac{t}{2} \rceil r}$) k -mers, each taking $2k$ bits of memory. Consequently, we have to adjust the optimal value of r minimizing the total space, and re-estimate the resulting space spent on one k -mer.

Table 1 shows estimations for optimal values of r and the corresponding space per k -mer for $t = 4$ and $t = 6$, and several values of k . The data demonstrates that even such small values of t lead to considerable memory savings. It appears that the space per k -mer is very close to the “optimal” space (8.45 bits) obtained for the infinite number of filters. Table 1 reveals another advantage of our improvement: the number of bits per stored k -mer remains almost constant for different values of k .

Table 1. 1st column: k -mer size; 2nd and 4th columns: optimal value of r for Bloom filters (bitmap size per number of stored elements) for $t = 4$ and $t = 6$ respectively; 3rd and 5th columns: the resulting space per k -mer (for $t = 4$ and $t = 6$); 6th column: space per k -mer for the method of [3] ($t = 1$)

k	optimal r for $t = 4$	bits per k -mer for $t = 4$	optimal r for $t = 6$	bits per k -mer for $t = 6$	bits per k -mer for $t = 1$ ([3])
16	5.777	8.556	5.506	8.459	12.078
32	6.049	8.664	5.556	8.47	13.518
64	6.399	8.824	5.641	8.49	14.958
128	6.819	9.045	5.772	8.524	16.398

The last column of Table 1 shows the memory usage of the original method of [3], obtained using the estimation $(1.44 \log_2(\frac{16k}{2.08}) + 2.08)$ the authors provided. Note that according to that estimation, doubling the value of k results in a memory increment by 1.44 bits, whereas in our method the increment is of 0.11 to 0.22 bits.

Let us now estimate preprocessing and query times for our scheme. If the value of t is small (such as $t = 4$, as in Table 1), the preprocessing time grows insignificantly in comparison to the original method of [3]. To construct each B_i , we need to store T_{i-2} (possibly on disk, if we want to save on the internal memory used by the algorithm) in order to compute those k -mers which are stored in B_{i-1} “by mistake”. The preprocessing time increases little in comparison to the original method of [3], as the size of B_i decreases exponentially and then the time spent to construct the whole structure is linear on the size of T_0 .

The query time can be split in two parts: the time spent on querying t Bloom filters and the time spent on querying T_t . Clearly, using t Bloom filters instead of a single one introduces a multiplicative factor of t to the first part of the query time. On the other hand, the set T_t is generally much smaller than T_1 , due to the above-mentioned exponential decrease. Depending on the data structure for storing T_t , the time saving in querying T_t vs. T_1 may even dominate the time loss in querying multiple Bloom filters. Our experimental results (Section 5.1 below) confirm that this situation does indeed occur in practice. Note that even in the case when querying T_t weakly depends on its size (e.g. when T_t is implemented by a hash table), the query time will not increase much, due to our choice of a small value for t , as discussed earlier.

4.1 Using Different Values of r for Different Filters

In the previous section, we assumed that each of our Bloom filters uses the same value of r , the ratio of bitmap size to the number of stored k -mers. However, formula (3) for the number of bits per k -mer shows a difference for odd and even filter indices. This suggests that using different parameters r for different

filters, rather than the same for all filters, may reduce the space even further. If r_i denotes the corresponding ratio for filter B_i , then (3) should be rewritten to

$$r_1 + 6r_2c^{r_1} + r_3c^{r_2} + 6r_4c^{r_1+r_3} + \dots, \quad (4)$$

and the minimum value of this expression becomes 7.93 (this value is achieved with $r_1 = 4.41; r_i = 1.44, i > 1$).

In the same way, we can use different values of r_i in the truncated case. This leads to a small 2% to 4% improvement in comparison with case of unique value of r . Table 2 shows results for the case $t = 4$ for different values of k .

Table 2. Estimated memory occupation for the case of different values of r vs. single value of r , for 4 Bloom filters ($t = 4$). Numbers in the second column represent values of r_i on which the minimum is achieved. For the case of single r , its value is shown in Table 1.

k	r_1, r_2, r_3, r_4	bits per k -mer different values of r	bits per k -mer single value of r
16	5.254, 3.541, 4.981, 8.653	8.336	8.556
32	5.383, 3.899, 5.318, 9.108	8.404	8.664
64	5.572, 4.452, 5.681, 9.108	8.512	8.824
128	5.786, 5.108, 6.109, 9.109	8.669	9.045

5 Experimental Results

5.1 Implementation and Experimental Setup

We implemented our method using the MINIA software [3] and ran comparative tests for 2 and 4 Bloom filters ($t = 2, 4$). Note that since the only modified part of MINIA was the construction step and the k -mer membership queries, this allows us to precisely evaluate our method against the one of [3].

The first step of the implementation is to retrieve the list of k -mers that appear more than d times using DSK [13] – a constant memory streaming algorithm to count k -mers. Each k -mer appearing more than d times (set T_0) is inserted into B_1 . Next, all possible extensions of each k -mer in T_0 are queried against B_1 , and those which return true are written to the disk. Then, this set is traversed and only the k -mers absent from T_0 are kept. This results in the set T_1 of critical false positives, which is also kept on disk. Up to this point, the procedure is identical to that of [3].

Next, we insert all k -mers from T_1 into B_2 and to obtain T_2 , we check for each k -mer in T_0 if a query to B_2 returns true. This results in the set T_2 . Thus, at this point we have B_1, B_2 and T_2 , a complete representation for $t = 2$. In order

to build the data structure for $t = 4$, we continue this process, by inserting T_2 in B_3 and retrieving T_3 from T_1 (stored on disk). It should be noted that to obtain T_i we need T_{i-2} , and by always storing it on disk we guarantee not to use more memory than the size of the final structure. The set T_i (that is, T_1 , T_2 or T_4 in our experiments) is stored as a sorted array and is searched by a binary search. We found this implementation more efficient than a hash table.

Assessing the query time is done through the procedure of graph traversal, as it is implemented in [3]. Since the procedure is identical and independent on the data structure, the time spent on graph traversal is a faithful estimator of the query time.

We compare three versions: $t = 1$ (i.e. the version of [3]), $t = 2$ and $t = 4$. For convenience, we define 1 Bloom, 2 Bloom and 4 Bloom as the versions with $t = 1, 2$ and 4 , respectively.

5.2 *E.coli* Dataset, Varying k

In this set of tests, our main goal was to evaluate the influence of the k -mer size on principal parameters: size of the whole data structure, size of the set T_t , graph traversal time, and time of construction of the data structure. We retrieved 10M *E. coli* reads of 100bp from the *Short Read Archive* (ERX008638) without read pairing information and extracted all k -mers occurring at least two times. The total number of k -mers considered varied, depending on the value of k , from 6,967,781 ($k = 15$) to 5,923,501 ($k = 63$). We ran each version, 1 Bloom ([3]), 2 Bloom and 4 Bloom, for values of k ranging from 16 to 64. The results are shown in Fig. 1.

The total size of the structures in bits per stored k -mer, i.e. the size of B_1 and T_1 (respectively, B_1, B_2, T_2 or B_1, B_2, B_3, B_4, T_4) is shown in Fig. 1(a). As expected, the space for 4 Bloom filters is the smallest for all values of k considered, showing a considerable improvement, ranging from 32% to 39%, over the version of [3]. Even the version with just 2 Bloom filters shows an improvement of at least 20% over [3], for all values of k . Regarding the influence of the k -mer size on the structure size, we observe that for 4 Bloom filters the structure size is almost constant, the minimum value is 8.60 and the largest is 8.89, an increase of only 3%. For 1 and 2 Bloom the same pattern is seen: a plateau from $k = 16$ to 32, a jump for $k = 33$ and another plateau from $k = 33$ to 64. The jump at $k = 32$ is due to switching from 64-bit to 128-bit representation of k -mers in the table T_t .

The traversal times for each version is shown in Fig. 1(c). The fastest version is 4 Bloom, showing an improvement over [3] of 18% to 30%, followed by 2 Bloom. This result is surprising and may seem counter-intuitive, as we have four filters to apply to the queried k -mer rather than a single filter as in [3]. However, the size of T_4 (or even T_2) is much smaller than T_1 , as the size of T_i 's decreases exponentially. As T_t is stored in an array, the time economy in searching T_4 (or T_2) compared to T_1 dominates the time lost on querying additional Bloom filters, which explains the overall gain in query time.

As far as the construction time is concerned (Fig. 1(d)), our versions yielded also a faster construction, with the 4 Bloom version being 5% to 22% faster than that of [3]. The gain is explained by the time required for sorting the array

storing T_t , which is much higher for T_0 than for T_2 or T_4 . However, the gain is less significant here, and, on the other hand, was not observed for bigger datasets (see Section 5.4).

5.3 *E. coli* Dataset, Varying Coverage

From the complete *E. coli* dataset (≈ 44 M reads) from the previous section, we selected several samples ranging from 5M to 40M reads in order to assess the impact of the coverage on the size of the data structures. This strain *E. coli* (K-12 MG1655) is estimated to have a genome of 4.6M bp [1], implying that a sample of 5M reads (of 100bp) corresponds to ≈ 100 X coverage. We set $d = 3$ and $k = 27$. The results are shown in Fig. 2. As expected, the memory consumption per k -mer remains almost constant for increasing coverage, with a slight decrease for 2 and 4 Bloom. The best results are obtained with the 4 Bloom version, an improvement of 33% over the 1 Bloom version of [3]. On the other hand, the number of distinct k -mers increases markedly (around 10% for each 5M reads) with increasing coverage, see Fig. 2(b). This is due to sequencing errors: an increase in coverage implies more errors with higher coverage, which are not removed by our cutoff $d = 3$. This suggests that the value of d should be chosen according to the coverage of the sample. Moreover, in the case where read qualities are available, a quality control pre-processing step may help to reduce the number of sequencing errors.

5.4 Human Dataset

We also compared 2 and 4 Bloom versions with the 1 Bloom version of [3] on a large dataset. For that, we retrieved 564M Human reads of 100bp (SRA: SRX016231) without pairing information and discarded the reads occurring less than 3 times. The dataset corresponds to ≈ 17 X coverage. A total of 2,455,753,508 k -mers were indexed. We ran each version, 1 Bloom ([3]), 2 Bloom and 4 Bloom with $k = 23$. The results are shown in Table 3.

The results are in general consistent with the previous tests on *E. coli* datasets. There is an improvement of 34% (21%) for the 4 Bloom (2 Bloom) in the size of the structure. The graph traversal is also 26% faster in the 4 Bloom version. However, in contrast to the previous results, the graph construction time increased by 10% and 7% for 4 and 2 Bloom versions respectively, when compared to the 1 Bloom version. This is due to the fact that disk writing/reading operations now dominate the time for the graph construction, and 2 and 4 Bloom versions generate more disk accesses than 1 Bloom. As stated in Section 5.1, when constructing the 1 Bloom structure, the only part written on the disk is T_1 and it is read only once to fill an array in memory. For 4 Bloom, T_1 and T_2 are written to the disk, and T_0 and T_1 are read at least one time each to build B_2 and B_3 . Moreover, since the size coefficient of B_1 reduces, from $r = 11.10$ in 1 Bloom to $r = 5.97$ in 4 Bloom, the number of false positives in T_1 increases.

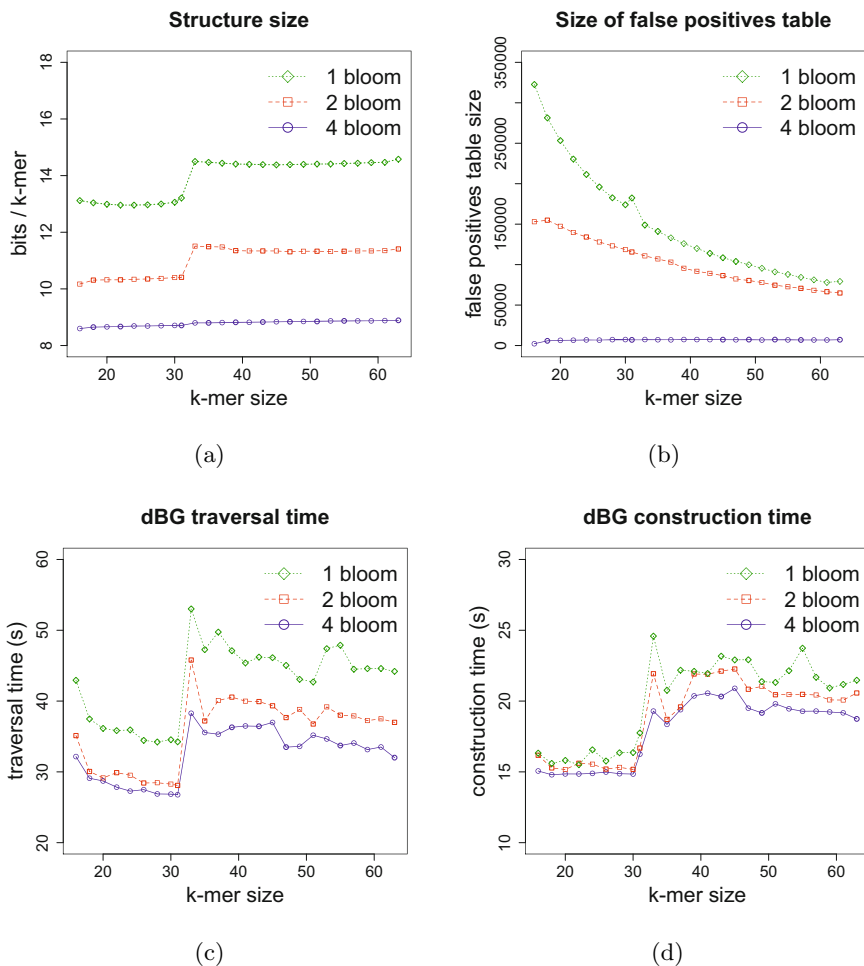


Fig. 1. Results for 10M E.coli reads of 100bp using several values of k . The *1 Bloom* version corresponds to the one presented in [3]. (a) Size of the structure in bits used per k -mer stored. (b) Number of false positives stored in T_1 , T_2 or T_4 for 1, 2 or 4 Bloom filters, respectively. (c) De Bruijn graph construction time, excluding k -mer counting step. (d) De Bruijn graph traversal time, including branching k -mer indexing.

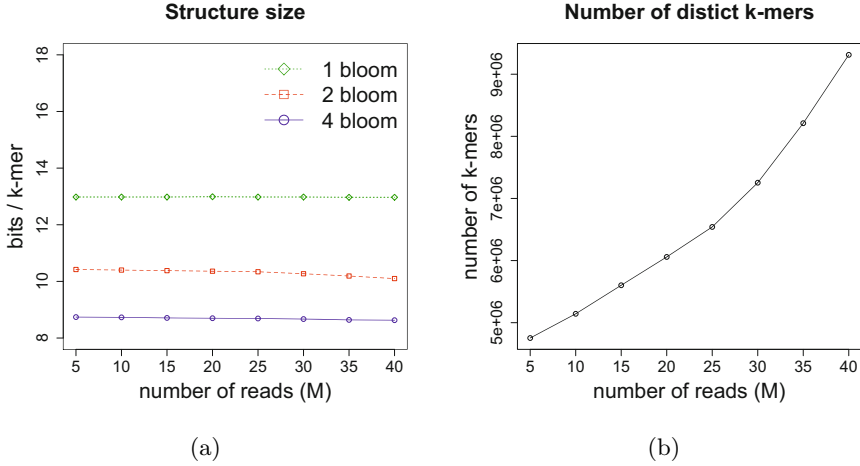


Fig. 2. Results for *E. coli* reads of 100bp using $k = 27$. The 1 Bloom version corresponds to the one presented in [3]. (a) Size of the structure in bits used per k -mer stored. (b) Number of distinct k -mers.

Table 3. Results of 1, 2 and 4 Bloom filters version for 564M Human reads of 100bp using $k = 23$. The 1 Bloom version corresponds to the one presented in [3].

Method	1 Bloom	2 Bloom	4 Bloom
Construction time (s)	40160.7	43362.8	44300.7
Traversal time (s)	46596.5	35909.3	34177.2
r coefficient	11.10	7.80	5.97
Bloom filters size (MB)	$B_1 = 3250.95$	$B_1 = 2283.64$ $B_2 = 323.08$	$B_1 = 1749.04$ $B_2 = 591.57$ $B_3 = 100.56$ $B_4 = 34.01$
False positive table size (MB)	$T_1 = 545.94$	$T_2 = 425.74$	$T_4 = 36.62$
Total size (MB)	3796.89	3032.46	2511.8
Size (bits/k-mer)	12.96	10.35	8.58

6 Discussion and Conclusions

Using cascading Bloom filters for storing de Bruijn graphs brings a clear advantage over the single-filter method of [3]. In terms of memory consumption, which is the main parameter here, we obtained an improvement of around 30%-40%

in all our experiments. Our data structure takes 8.5 to 9 bits per stored k -mer, compared to 13 to 15 bits by the method of [3]. This confirms our analytical estimations. The above results were obtained using only four filters and are very close to the estimated optimum (around 8.4 bits/ k -mer) produced by the infinite number of filters. An interesting characteristic of our method is that the memory grows insignificantly with the growth of k , even slower than with the method of [3]. Somewhat surprisingly, we also obtained a significant decrease, of order 20%-30%, of query time. The construction time of the data structure varied from being 10% slower (for the human dataset) to 22% faster (for the bacterial dataset).

As stated previously, another compact encoding of de Bruijn graphs has been proposed in [2], however no implementation of the method was made available. For this reason, we could not experimentally compare our method with the one of [2]. We remark, however, that the space bound of [2] heavily depends on the number of reads (i.e. coverage), while in our case, the data structure size is almost invariant with respect to the coverage (Section 5.3).

An interesting prospect for further possible improvements of our method is offered by work [12], where an efficient replacement to Bloom filter was introduced. The results of [12] suggest that we could hope to reduce the memory to about 5 bits per k -mer. However, there exist obstacles on this way: an implementation of such a structure would probably result in a significant construction and query time increase.

Acknowledgements. Part of this work has been done during the visit of KS to LIGM in France, supported by the CNRS French-Russian exchange program in Computer Science. GK has been partly supported by the ABS2NGS grant of the French gouvernement (program *Investissement d'Avenir*) as well as by a Marie-Curie Intra-European Fellowship for Carrier Development. GS was supported by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. [247073]10

References

1. Blattner, F.R., Plunkett, G., Bloch, C.A., et al.: The complete genome sequence of *Escherichia coli* k-12. *Science* 277(5331), 1453–1462 (1997)
2. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de Bruijn graphs. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS, vol. 7534, pp. 225–235. Springer, Heidelberg (2012)
3. Chikhi, R., Rizk, G.: Space-efficient and exact de bruijn graph representation based on a bloom filter. In: Raphael, B., Tang, J. (eds.) WABI 2012. LNCS, vol. 7534, pp. 236–248. Springer, Heidelberg (2012)
4. Conway, T.C., Bromage, A.J.: Succinct data structures for assembling large genomes. *Bioinformatics* 27(4), 479–486 (2011)
5. Grabherr, M.G., Haas, B.J., Yassour, M., Levin, J.Z., et al.: Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nat. Biotech.* 29(7), 644–652 (2011)

6. Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G.: De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nat. Genet.* 44(2), 226–232 (2012)
7. Kirsch, A., Mitzenmacher, M.: Less hashing, same performance: Building a better bloom filter. *Random Struct. Algorithms* 33(2), 187–218 (2008)
8. Miller, J.R., Koren, S., Sutton, G.: Assembly algorithms for next-generation sequencing data. *Genomics* 95(6), 315–327 (2010)
9. Pell, J., Hintze, A., Canino-Koning, R., Howe, A., Tiedje, J.M., Brown, C.T.: Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proc. Natl. Acad. Sci. U.S.A.* 109(33), 13272–13277 (2012)
10. Peng, Y., Leung, H.C.M., Yiu, S.M., Chin, F.Y.L.: Meta-IDBA: a de novo assembler for metagenomic data. *Bioinformatics* 27(13), i94–i101 (2011)
11. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. U.S.A.* 98(17), 9748–9753 (2001)
12. Porat, E.: An optimal Bloom filter replacement based on matrix solving. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) *CSR 2009*. LNCS, vol. 5675, pp. 263–273. Springer, Heidelberg (2009)
13. Rizk, G., Lavenier, D., Chikhi, R.: DSK: k-mer counting with very low memory usage. *Bioinformatics* (2013)
14. Sacomoto, G., Kielbassa, J., Chikhi, R., Uricaru, R., et al.: KISSPLICE: de novo calling alternative splicing events from RNA-seq data. *BMC Bioinformatics* 13(suppl. 6), S5 (2012)
15. Ye, C., Ma, Z., Cannon, C., Pop, M., Yu, D.: Exploiting sparseness in de novo genome assembly. *BMC Bioinformatics* 13(suppl. 6), S1 (2012)

Author Index

- Alpert, Matthew 70
- Backofen, Rolf 112
- Bafna, Vineet 349
- Bernt, Matthias 200
- Biller, Priscila 230
- Böcker, Sebastian 45, 156
- Brejová, Broňa 287
- Bu, Dongbo 18
- Bucher, Philipp 326
- Bulteau, Laurent 244
- Canzar, Stefan 156
- CiarDO, Gianfranco 70
- Close, Timothy J. 70
- Cui, Xuefeng 18
- Deshpande, Viraj 349
- Dojer, Norbert 259
- Duggal, Geet 300
- Dührkop, Kai 45
- Duma, Denisa 70
- Farhan, Hesso 33
- Fernández-Baca, David 185
- Fertin, Guillaume 244
- Filippova, Darya 300
- Fricke, Markus 112
- Frid, Yelena 126
- Fung, Eric D.K. 349
- Gat-Viks, Irit 33
- Gilbert, Anna C. 70
- Gusfield, Dan 126
- Jansson, Jesper 141
- Kapun, Evgeny 59
- Kingsford, Carl 300
- Klau, Gunnar W. 156
- Kolmogorov, Mikhail 215
- Komusiewicz, Christian 244
- Kucherov, Gregory 364
- Kuosmanen, Anna 85
- Lacroix, Vincent 99
- Leibovich, Limor 273
- Li, Ming 18
- Li, Shuai Cheng 18
- Lin, Yu 326
- Lonardi, Stefano 70
- Ludwig, Marcus 45
- Mäkinen, Veli 85
- Marz, Manja 112
- Mazza, Arnon 33
- Meidanis, João 230
- Meusel, Marvin 45
- Middendorf, Martin 200
- Minkin, Ilya 215
- Modzelewski, Michał 259
- Moret, Bernard M.E. 1, 326
- Nair, Nishanth Ulhas 326
- Nánási, Michal 287
- Ngo, Hung Q. 70
- Onodera, Taku 338
- Patel, Anand 215
- Patro, Rob 300
- Pham, Son 215, 349
- Qin, Jing 112
- Rizzi, Romeo 85
- Rudra, Atri 70
- Rusu, Irena 244
- Sacomoto, Gustavo 99, 364
- Sadakane, Kunihiko 338
- Sagot, Marie-France 99
- Salikhov, Kamil 364
- Sandel, Brody 170
- Sharan, Roded 33
- Shatkay, Hagit 3
- Shen, Chuanqi 141
- Shibuya, Tetsuo 338
- Simha, Ramanuja 3

Stadler, Peter F. 112
Sung, Wing-Kin 141

Tomescu, Alexandru I. 85
Tsarev, Fedor 59
Tsirogiannis, Constantinos 170
Tsourakakis, Charalampos E. 313

Vakati, Sudheer 185
Venkatachalam, Balaji 126

Vinař, Tomáš 287
Vyahhi, Nikolay 215

Wieseke, Nicolas 200
Wootters, Mary 70

Yakhini, Zohar 273

Zanetti, João Paulo Pereira 230