Eric Tannier (Ed.)

LNBI 6398

Comparative Genomics

International Workshop, RECOMB-CG 2010
Ottawa, Canada, October 2010
Proceedings

Springer

Lecture Notes in Bioinformatics 6398

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

Subseries of Lecture Notes in Computer Science

Eric Tannier (Ed.)

Comparative Genomics

International Workshop, RECOMB-CG 2010
Ottawa, Canada, October 9-11, 2010
Proceedings

Springer

Volume Editor

Eric Tannier
INRIA Rhône-Alpes
Laboratoire de Biométrie et Biologie Evolutive
Université de Lyon 1
43, boulevard du 11 novembre 1918
69622 Villeurbanne, France
E-mail: Eric.Tannier@inria.fr

Library of Congress Control Number: 2010935849

CR Subject Classification (1998): F.2, G.3, E.1, H.2.8, J.3

LNCS Sublibrary: SL 8 – Bioinformatics

ISSN 0302-9743
ISBN-10 3-642-16180-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-16180-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The complexity of genome evolution has given birth to exciting challenges for computational biologists. A various range of algorithmic, statistical, mathematical techniques to elucidate the histories of molecules are developed each year and many are presented at the RECOMB satellite workshop on Comparative Genomics. It is a place where scientists working on all aspects of comparative genomics can share ideas on the development of tools and their application to relevant questions.

This volume contains the papers presented at RECOMB-CG 2010, held on October 9–11 in Ottawa. The field is still flourishing as seen from the papers presented this year: many developments enrich the combinatorics of genome rearrangements, while gene order phylogenies are becoming more and more accurate, thanks to a mixing of combinatorial and statistical principles, associated with rapid and thoughtful heuristics. Several papers tend to refine the models of genome evolution, and more and more genomic events can be modeled, from single nucleotide substitutions in whole genome alignments to large structural mutations or horizontal gene transfers.

There were 35 submissions. Each submission was reviewed by at least 2, and on average 2.9, program committee members. The committee decided to accept 24 papers. The program also included 6 invited talks:

Brenda Andrews, University of Toronto
Andrew G. Clark, Cornell University
Nicolas Corradi, University of Ottawa
Jan Dvorak, University of California at Davis
Aoife McLysaght, University of Dublin
Nicholas Putnam, Rice University

I would like to thank all the participants of the conference, and in addition all the people who submitted a paper or a poster. Thanks also to the program committee members and the other reviewers, who did a great amount of work for the conference in a limited amount of time.

The work of the Program Committee was considerably facilitated by the EasyChair website. RECOMB CG 2010 was supported in part by grants from the Canadian Institute for Advanced Research, the Fields Institute for Research in Mathematical Sciences, the Mathematics of Information Technology and Complex Systems (MITACS) Network of Centres of Excellence, and the Vice-President of Research of the University of Ottawa.

Thanks are due to Chunfang Zheng for website design and management, and for implementation of the online registration process.

July 2010 Eric Tannier

Organization

Committees and Additional Reviewers

Conference Chair

David Sankoff University of Ottawa, Canada

Steering Committee

Aoife McLysaght	University of Dublin, Trinity College, Ireland
Jens Lagergren	Royal Institute of Technology (KTH), Sweden
David Sankoff	University of Ottawa, Canada

Local Organizing Committee

Anne Bergeron	Université du Québec à Montréal
Frank Dehne	Carleton University
Guy Drouin	University of Ottawa
Nadia El-Mabrouk	Université de Montréal
Evangelos Kranakis	Carleton University
Marcel Turcotte	University of Ottawa

Program Committee Chair

Eric Tannier	INRIA, Université de Lyon, France

Program Committee

Lars Arvestad	Royal Institute of Technology (KTH), Sweden
Anne Bergeron	Université du Québec à Montréal, Canada
Mathieu Blanchette	McGill University, Montreal, Canada
Guillaume Blin	Université Paris-Est, France
Guillaume Bourque	Genome Institute of Singapore, Singapore
Marilia Braga	Universität Bielefeld, Germany
Michael Brudno	University of Toronto, Canada
Jeremy Buhler	Washington University in Saint Louis, USA
Sèverine Bérard	Université de Montpellier, France
Cedric Chauve	Simon Fraser University, Canada
Avril Coghlan	University College Cork, Ireland
Aaron Darling	University of California-Davis, USA
Dannie Durand	Carnegie Mellon University, USA
Nadia El-Mabrouk	University of Montreal, Canada
Niklas Eriksen	University of Gothenburg, Sweden

Patricia Evans	University of New Brunswick, Canada
Guillaume Fertin	Université de Nantes, France
Matthew Hahn	Indiana University, USA
Joao Meidanis	University of Campinas / Scylla Bioinformatics, Brazil
Bernard Moret	Ecole Polytechnique Fédérale de Lausanne, Switzerland
Craig Nelson	University of Connecticut, USA
Aida Ouangraoua	INRIA - Lille Nord Europe, France
Michal Ozery-Flato	Tel-Aviv University, Israel
Teresa Przytycka	NIH, USA
Eric Rivals, CNRS	Université de Montpellier, France
Eduardo Rocha	CNRS, Université Paris 6, France
Hugues Roest Crollius	CNRS, Ecole Normale Supérieure Paris, France
Jens Stoye	Universität Bielefeld, Germany
Krister Swenson	Ecole Polytechnique Fédérale de Lausanne, Switzerland
Glenn Tesler	University of California, San Diego, USA
Elisabeth Tillier	Cancer Institute of Ontario, University Health Network, Canada
Stéphane Vialette	CNRS, Université Paris-Est, France
Tiffani Williams	Texas A&M University, USA
Sophia Yancopoulos	Feinstein Institute for Medical Research, USA
Louxin Zhang	National University of Singapore, Singapore

Additional Reviewers

Sébastien Angibaud	Université de Nantes, France
Robert Beiko	Dalhousie University, Halifax, Canada
Camille Berthelot	Ecole Normale Supérieure Paris, France
Annie Chateau	Université de Montpellier, France
Pedro Feijao	University of Campinas, Brazil
Katharina Jahn	Universität Bielefeld, Germany
Irena Rusu	Université de Nantes, France
Florian Sikora	Université Paris-Est, France
Maureen Stolzer	Carnegie Mellon University, USA
Todd Treangen	Institut Pasteur, CNRS and UPMC University, France
Roland Wittler	Simon Fraser University, Canada
Damian Wojtowicz	NIH, USA
Jie Zheng	NIH, USA

Previous Meetings

2003, Minneapolis, USA
2004, Bertinoro, Italy
2005, Dublin, Ireland
2006, Montreal, Canada
2007, San Diego, USA
2008, Paris, France
2009, Budapest, Hungary

Table of Contents

Genome Aliquoting Revisited

Robert Warren and David Sankoff

University of Ottawa, Ottawa, ON, Canada

Abstract. We prove that the genome aliquoting problem, the problem of finding a recent polyploid ancestor of a genome, with breakpoint distance can be solved in polynomial time. We propose an aliquoting algorithm that is a 2-approximation for the genome aliquoting problem with double cut and join distance, improving upon the previous best solution to this problem, Feijão and Meidanis' 4-approximation algorithm.

Comparing two genomes with duplicated genes is difficult. None of the distances used to compare genomes today (breakpoint distance, reversal distance, double cut and join distance, etc...) handle duplicated genes. However, in the special case where all genes are duplicated the same number of times, there has been some success.

Informally, the *genome aliquoting problem* is the problem of finding a genome with one copy of every gene given a genome with exactly p copies of every gene such that the distance between the given and resulting genomes is minimized according to some distance metric. Thus, the genome aliquoting problem eliminates the duplicate genes allowing a genome to be compared with other genomes using an existing algorithm. Solving this problem will allow genomes which have undergone a recent polyploidization event, common in plants, to be compared.

There have been a number of solutions to the genome aliquoting problem where the genome has exactly two copies of every gene. This restricted version of the problem is called the *genome halving problem* and was first introduced in [3]. It was solved for reversal and translocation distance in [3,1] and for double cut and join distance in [10,6].

The *genome aliquoting problem* was introduced in [9] along with a sketch of a heuristic algorithm for the problem under double cut and join distance. [4] provided an exact solution under single cut or join distance which is also a 4-approximation algorithm under double cut and join distance. In this paper, we provide an exact polynomial-time algorithm under breakpoint distance which is also a 2-approximation algorithm under double cut and join distance. Since our algorithm is similar to that presented in [9] it also bounds that heuristic as a 2-approximation for double cut and join distance.

1 Duplicated Genomes

The fundamental elements that we study are *genes*. We represent each gene as a pair of *extremities* such that a gene x is represented by its *head* $\overrightarrow{x}$, which

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 1–12, 2010.

corresponds to the $3'$ end of the gene, and its *tail* $\overset{\succ}{x}$, which corresponds to the $5'$ end of the gene. A *genome* is represented by a multiset of 2-multisets and 1-sets of extremities, called *adjacencies* and *telomeres* respectively, such that, for each gene, the genome contains exactly the same number of heads as tails. The functions $\mathcal{G}(G)$ and $\mathcal{E}(G)$ return the set of all genes or extremities respectively of a genome G. The collection of copies of the same gene are called a *gene family* with the *size* being the number of copies in the genome (*i.e.* the number of heads, or, alternatively, the number of tails, that appear in a genome).

Genomes with one or more gene families with size greater than one are challenging to manipulate. Not only must there be the same number of heads and tails but to understand the layout of the genome we must know for duplicated genes which head corresponds to which tail, otherwise the layout of the genome is not unique. Similarly, to compute the distance of two genomes we must know which copy in one genome matches which copy in another genome. Genomes where this information is known are called *ordered genomes*. In such genomes we distinguish members of the same gene family by a subscript such that each head corresponds to the tail with the same subscript, *e.g.* $\overrightarrow{a_1}$ corresponds to $\overset{\succ}{a_1}$. Similarly, if we know the ordering between two genomes then each extremity in one genome must correspond to the extremity with the same subscript in the other genome, *e.g.* $\overrightarrow{a_1}$ in one genome must correspond to $\overrightarrow{a_1}$ in the other genome. By default we assume most genomes are ordered with themselves but the challenge of comparing genomes with duplicated genes comes from finding an ordering between two genomes as different orderings change the distance. Thus, to compare genomes with duplicated genes we must find an ordering between them that minimizes their distance.

For most situations, ordered genomes are all that is needed. However, ordered genomes can be difficult to use, *e.g.* $\{\overrightarrow{a_1}, \overrightarrow{b_1}\} \cap \{\overrightarrow{a_2}, \overrightarrow{b_2}\} = \emptyset$ and yet, frequently, it is desirable to perceive them as equal. Since these problems tend to occur frequently in our work we introduce the concept of an *unordered genome* where no effort is made to distinguish the genes. Let $\tilde{G}$ be the unordered counterpart of an ordered genome G. Similarly, if α is an element of G then $\tilde{\alpha}$ is an element of $\tilde{G}$ and if S is a subset of G then $\tilde{S}$ is a subset of $\tilde{G}$. Unordered extremities are distinguished from ordered extremities by the absence or presence respectively of a subscript. Transforming an ordered genome to an unordered genome is trivial but the reverse can be very difficult and is problem specific.

A concept that helps solve the genome aliquoting problem is :

Definition 1. *We define* perfection *between two elements of a genome to be that the two elements are the same or are disjoint. For adjacencies we also have the special requirement that they are not of the form $\{x, x\}$ for some extremity x. Since there are two types of elements, adjacencies and telomeres, two elements α and β can be in one of the following three configurations:*

- *let α and β be a pair of adjacencies. They are* perfect *if and only if they are disjoint or the same and neither are of form $\{x, x\}$ for some extremity x;*
- *let α and β be a pair of telomeres. Since all telomeres are either the same or disjoint all pairs of telomeres are* perfect*;*

– *let α be an adjacency and β be a telomere, since they cannot be the same they must be disjoint in order to be* perfect*. Also, α must not be of the form $\{x, x\}$ for some extremity x;*

A genome, or any set of adjacencies and telomeres, is perfect *if and only if all pairs of elements are perfect.*

For example, given a genome $G = \{\{\vec{a_1}\}, \{\overset{\succ}{a_1}, \overset{\succ}{b_1}\}, \{\vec{b_1}, \overset{\succ}{d_1}\}, \{\vec{d_1}, \overset{\succ}{e_1}\}, \{\vec{e_1}, \overset{\succ}{b_2}\}, \{\vec{b_2}\}, \{\vec{a_2}\}, \{\overset{\succ}{a_2}, \vec{c_1}\}, \{\overset{\succ}{c_1}, \overset{\succ}{c_2}\}, \{\vec{c_2}, \vec{d_2}\}, \{\overset{\succ}{d_2}, \overset{\succ}{e_2}\}, \{\vec{e_2}\}\}$ its corresponding unordered genome is $\tilde{G} = \{\{\vec{a}\}, \{\overset{\succ}{a}, \overset{\succ}{b}\}, \{\vec{b}, \vec{d}\}, \{\vec{d}, \overset{\succ}{e}\}, \{\vec{e}, \overset{\succ}{b}\}, \{\vec{b}\}, \{\vec{a}\}, \{\overset{\succ}{a}, \vec{c}\}, \{\overset{\succ}{c}, \overset{\succ}{c}\}, \{\vec{c}, \vec{d}\}, \{\overset{\succ}{d}, \overset{\succ}{e}\}, \{\vec{e}\}\}$. In $\tilde{G}$, the adjacencies $\{\overset{\succ}{a}, \overset{\succ}{b}\}$ and $\{\vec{b}, \overset{\succ}{d}\}$ are perfect because they are disjoint but $\{\overset{\succ}{a}, \overset{\succ}{b}\}$ and $\{\overset{\succ}{a}, \vec{c}\}$ are not perfect because they are neither disjoint nor equal (they have $\overset{\succ}{a}$ in common but do not share the other extremity). The adjacency $\{\overset{\succ}{c}, \overset{\succ}{c}\}$ can never be perfect because it is of the form $\{x, x\}$ for some extremity x. The telomere $\{\vec{a}\}$ is perfect when compared with both $\{\vec{a}\}$ and $\{\vec{e}\}$ since it is equal to the former and disjoint with the latter. $\{\vec{e}\}$ is not perfect when compared with $\{\vec{e}, \overset{\succ}{b}\}$ because they are not disjoint.

In biology, a *polyploid* is a genome with multiple copies of the same chromosome. A perfect unordered genome where all gene families are of the same size shares this property and, hence, is called a *polyploid*. Similarly, an ordered genome G whose corresponding unordered genome $\tilde{G}$ is a polyploid is a *polyploid*.

Definition 2. *Given an ordered genome G with all gene families of size p, the* genome aliquoting problem *is to find a polyploid H with all gene families of size p ordered in relation to G such that the distance between G and H is minimal.*

2 Breakpoint Distance

A *breakpoint* (BP) is a difference in adjacencies or telomeres between two genomes, *e.g.* in $G = \{\{\vec{a_1}\}, \{\overset{\succ}{a_1}, \overset{\succ}{b_1}\}, \{\vec{b_1}, \overset{\succ}{d_1}\}, \{\vec{d_1}, \overset{\succ}{e_1}\}, \{\vec{e_1}, \overset{\succ}{b_2}\}, \{\vec{b_2}\}, \{\vec{a_2}\}, \{\overset{\succ}{a_2}, \vec{c_1}\}, \{\overset{\succ}{c_1}, \overset{\succ}{c_2}\}, \{\vec{c_2}, \vec{d_2}\}, \{\overset{\succ}{d_2}, \overset{\succ}{e_2}\}, \{\vec{e_2}\}\}$ and $H = \{\{\vec{a_1}\}, \{\overset{\succ}{a_1}, \vec{b_1}\}, \{\overset{\succ}{b_1}, \vec{c_1}\}, \{\overset{\succ}{c_1}, \vec{d_1}\}, \{\overset{\succ}{d_1}, \vec{e_1}\}, \{\overset{\succ}{e_1}, \vec{a_2}\}, \{\overset{\succ}{a_2}, \vec{b_2}\}, \{\overset{\succ}{b_2}, \vec{c_2}\}, \{\overset{\succ}{c_2}, \vec{d_2}\}, \{\overset{\succ}{d_2}, \vec{e_2}\}, \{\overset{\succ}{e_2}\}\}, \{\vec{b_1}, \overset{\succ}{d_1}\}$ is a breakpoint in G since there is no equivalent element in H but $\{\vec{a_1}\}$ is not a breakpoint because it is in both genomes. Extending this notion to entire genomes we arrive at the following definition:

Definition 3. *The* breakpoint distance*, introduced in [7], between two genomes G and H is the number of breakpoints between G and H.*

The breakpoint distance is easy to calculate. From [8], given two genomes G and H, let $\mathcal{A}(G, H)$ be the set of adjacencies shared between G and H and let $\mathcal{T}(G, H)$ be the set of telomeres then the *breakpoint distance* between G and H can be calculated using the following equation:

$$\mathcal{D}_{BP}(G, H) = |\mathcal{G}(G)| - |\mathcal{A}(G, H)| - \frac{|\mathcal{T}(G, H)|}{2} \tag{1}$$

For example, given two duplicated genomes $G = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \underline{\{\overrightarrow{b_1}, \overrightarrow{d_1}\}}, \underline{\{\overrightarrow{d_1}, \overrightarrow{e_1}\}}, \underline{\{\overrightarrow{e_1}, \overrightarrow{b_2}\}}, \underline{\{\overrightarrow{b_2}\}}, \underline{\{\overrightarrow{a_2}\}}, \underline{\{\overrightarrow{a_2}, \overrightarrow{c_1}\}}, \underline{\{\overrightarrow{c_1}, \overrightarrow{c_2}\}}, \underline{\{\overrightarrow{c_2}, \overrightarrow{d_2}\}}, \underline{\{\overrightarrow{d_2}, \overrightarrow{e_2}\}}, \underline{\{\overrightarrow{e_2}\}}\}$ and $H = \{\{\overrightarrow{a_1}\}, \{\overrightarrow{a_1}, \overrightarrow{b_1}\}, \underline{\{\overrightarrow{b_1}, \overrightarrow{c_1}\}}, \underline{\{\overrightarrow{c_1}, \overrightarrow{d_1}\}}, \underline{\{\overrightarrow{d_1}, \overrightarrow{e_1}\}}, \underline{\{\overrightarrow{e_1}, \overrightarrow{a_2}\}}, \underline{\{\overrightarrow{a_2}, \overrightarrow{b_2}\}}, \underline{\{\overrightarrow{b_2}, \overrightarrow{c_2}\}}, \underline{\{\overrightarrow{c_2}, \overrightarrow{d_2}\}}, \underline{\{\overrightarrow{d_2}, \overrightarrow{e_2}\}}, \underline{\{\overrightarrow{e_2}\}}\}$ the breakpoint are underlined; since $\mathcal{A}(G, H) = 1$ and $\mathcal{T}(G, H) = 1$ and $\mathcal{G}(G) = 8$, $\mathcal{D}_{BP}(G, H) = 6.5$.

3 Quasi Maximum Perfect and Covering Sets

Since polyploids are perfect genomes where every gene family has the same size, our strategy to aliquote a genome G is to find a perfect subset of G and then transform it into a perfect genome. Because manipulating G directly is challenging, we will manipulate its unordered counterpart $\tilde{G}$ instead.

There are many possible perfect subsets of $\tilde{G}$ and we need to choose the one which will minimize the distance. From Equation 1 we can see that the distance is minimized by maximizing the number of adjacencies and telomeres present in both genomes. Thus, we define the *weight* of a perfect set P in relation to a genome $\tilde{G}$ to be:

$$\mathcal{W}(P) = \mathcal{A}(\tilde{G}, P) + \frac{\mathcal{T}(\tilde{G}, P)}{2} \tag{2}$$

Define a *maximum perfect set* P as a perfect subset of $\tilde{G}$ with maximum weight.

While it is impossible to compute the distance using an unordered genome, it is easy to see that, once transformed into a genome and reordered, a maximum perfect subset of $\tilde{G}$ will minimize the distance. However, not all maximum perfect sets can be transformed into an ordered genome that can be compared with G. In addition to being ordered, to be compared two genomes must have the same set of extremities and it is possible to construct a maximum perfect set that is missing extremities from $\tilde{G}$. Thus, we introduce the notion of a *covering set*. A set C covers an unordered genome $\tilde{G}$ if and only if it contains at least one copy of every extremity in $\tilde{G}$.

Unfortunately, we cannot always find a maximum perfect and covering subset of $\tilde{G}$, for some unordered genomes no such subset exists. To solve the problem we remove the restriction that it must be a subset. However, without the restriction that the perfect set has to be a subset of $\tilde{G}$, the idea of a maximum perfect set does not make any sense; we could add an infinite number of adjacencies and telomeres to the set if they do not have to come from the genome. Thus, we introduce a *quasi maximum perfect set*: one with a maximum perfect set as a subset but with additional elements not from $\tilde{G}$. All maximum perfect sets are also quasi maximum perfect sets, although the reverse is not true.

It is easy to transform a quasi maximum perfect and covering set with respect to a genome $\tilde{G}$ into a polyploid that, if ordered, could be compared with G: we simply increase the number of copies of each element. Algorithm 1 does exactly this, although we omit the proof of its correctness due to space constraints.

Algorithm 1. REPLICATE

Input: P quasi maximum perfect and covering set with respect to a genome $\tilde{G}$ and p, the size of the gene families in $\tilde{G}$.
Output: A polyploid $\tilde{H}$ with all gene families of size p.

```
1 foreach α ∈ P do
2     if the multiplicity of α is not p then
3         add α to (or remove α from) H̃ until the multiplicity is p
4     end
5 end
6 return P
```

4 Reorder Genes

How a genome is reordered can affect the distance. However, with a quasi maximum perfect and covering set with respect to $\tilde{G}$ as a base, it is easy to reorder the resulting polyploid such that its distance to G is minimal. Algorithm 2 reorders the genome and the following theorem proves its correctness:

Theorem 1. *Given a genome G with all gene families of size p, let P be a quasi maximum perfect and covering set with respect to $\tilde{G}$ and let $\tilde{H}$=*REPLICATE*(P, p), then $\mathcal{D}_{BP}(G, H)$ where $H =$* REORDER*$(\tilde{H}, G)$ is minimal.*

Proof. $\mathcal{D}_{BP}(G, H)$ is minimal if there does not exist an other genome H' such that $\mathcal{D}_{BP}(G, H') < \mathcal{D}_{BP}(G, H)$. Assume towards contradiction that such an H' does exist.

Let E be the set of elements that are identical between H and G and let E' be the set of elements that are identical between H' and G. We observe that because H and H' are polyploids $\tilde{E}$ and $\tilde{E}'$ must be a perfect.

From Equation 1 only adjacencies and telomeres that are the same in both genomes reduce the distance. Since $\mathcal{D}_{BP}(G, H') < \mathcal{D}_{BP}(G, H)$, $\mathcal{W}(\tilde{E}') > \mathcal{W}(\tilde{E})$.

A quasi maximum perfect set must have a maximum perfect set as a subset; let $M \subseteq P$ be such a maximum perfect set of $\tilde{G}$. Since E is a subset of H it follows that $\tilde{E}$ must be a subset of $\tilde{H}$. Thus, M and $\tilde{E}$ are subsets of $\tilde{H}$ and, in fact, we will prove that $M = \tilde{E}$ by assuming towards contradiction that $M \neq \tilde{E}$.

$M \neq \tilde{E}$ if and only if the multiplicity of all elements in M is not equal to the multiplicity of all elements in $\tilde{E}$. Let α be an element that has different multiplicities in both M and $\tilde{E}$. Since M is a maximum perfect set of $\tilde{G}$ all of its elements can have a multiplicity of at most p and, because M is a maximum, the multiplicity of α in M is equal to the multiplicity of α in $\tilde{G}$. Since REPLICATE(P, p) creates $\tilde{H}$ by setting all the elements of P, which includes all of the elements of M, to multiplicity p, the multiplicity of α in $\tilde{H}$ is greater than or equal to that of M. It follows that the multiplicity of α in M is the multiplicity of α in $\tilde{G} \cap \tilde{H}$. From Lines Lines 2 – 6 in Algorithm 2 REORDER$(\tilde{H}, G)$ we can conclude that $\tilde{G} \cap \tilde{H} = \widetilde{(G \cap H)}$. By definition, $E = G \cap H$, hence, $\widetilde{(G \cap H)} = \tilde{E}$. Therefore, the multiplicities of α in M and in $\tilde{E}$ are equal; a contradiction. Thus, $M = \tilde{E}$.

Algorithm 2. REORDER

Input: A polyploid $\tilde{H}$ with all gene families of size p and a genome G with all gene families of size p where $\mathcal{E}(\tilde{G}) = \mathcal{E}(\tilde{H})$.
Output: A polyploid H with all gene families of size p such that $\mathcal{D}_{BP}(G, H)$ is minimal.

```
H ← ∅
foreach element α ∈ G do
    if α̃ ∈ H̃ then
        add α to H
    end
end
E ← E(G) \ E(H)
foreach element α̃ ∈ H̃ do
    if α̃ is an adjacency then
        let {x, y} = α̃ where x, y are extremities
        while there exists an i such that x_i ∈ E and a j such that y_j ∈ E do
            add {x_i, y_j} to H
            remove x_i from E
            remove y_j from E
        end
    else
        let {x} = α̃ where x is an extremity
        while there exists an i such that x_i ∈ E do
            add {x_i} to H
            remove x_i from E
        end
    end
end
return H
```

$\mathcal{W}(\tilde{E}') > \mathcal{W}(M)$ follows from the facts that $M = \tilde{E}$ and $\mathcal{W}(\tilde{E}') > \mathcal{W}(\tilde{E})$. But M is a maximum perfect set; a contradiction. Hence, H must be minimal. □

5 Fully Modified Clique Graphs

We have reduced the problem of aliquoting a genome to that of finding a maximum perfect and covering set. To find the maximum perfect and covering set, we construct a graph from the genome that highlights its important information.

Definition 4. *A* fully modified weighted clique graph *of a genome $\tilde{G}$, denoted $\mathcal{CG}(\tilde{G})$, is defined as follows:*

- *For each gene family x in $\tilde{G}$, $\mathcal{CG}(\tilde{G})$ has four vertices, one labeled $\vec{x}$, one labeled $\overleftarrow{x}$ and two without labels, called* null vertices*, of which one is connected to the vertex labelled with $\vec{x}$ and the other to the vertex labeled with $\overleftarrow{x}$;*

- *For each adjacency* $\{x, y\}$ *in* $\tilde{G}$ *where* $x \neq y$*, there is an edge between the vertices labeled* x *and* y *with weight equal to the multiplicity of* $\{x, y\}$ *in* $\tilde{G}$*;*
- *For each telomere* $\{x\}$ *in* $\tilde{G}$*, the edge between the vertex labeled* x *and its adjacent null vertex has weight equal to half of the multiplicity of* $\{x\}$ *in* $\tilde{G}$*;*
- *All other edges in* $\mathcal{CG}(\tilde{G})$ *have weights of* 0*.*

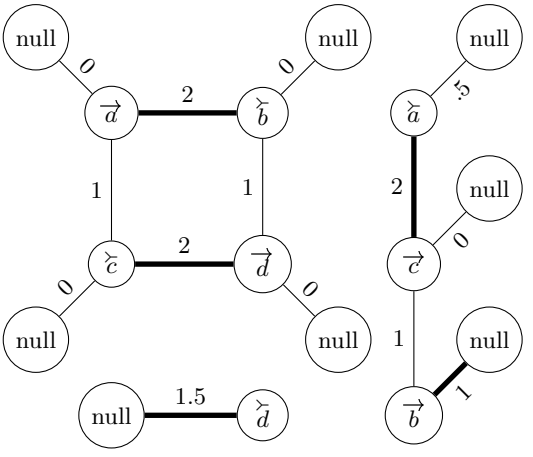


Fig. 1. An example of a fully modified weighted clique graph. The five bold edges represent all the edges that belong to the maximum weight matching of this graph.

For example, the modified weighted clique graph of the genome $\tilde{G} = \{\{\overleftarrow{a}\}, \{\overrightarrow{a}, \overleftarrow{b}\}, \{\overrightarrow{b}, \overrightarrow{c}\}, \{\overleftarrow{c}, \overrightarrow{a}\}, \{\overleftarrow{a}, \overrightarrow{c}\}, \{\overleftarrow{c}, \overrightarrow{d}\}, \{\overleftarrow{d}\}, \{\overleftarrow{d}\}, \{\overrightarrow{d}, \overleftarrow{c}\}, \{\overrightarrow{c}, \overleftarrow{a}\}, \{\overrightarrow{a}, \overleftarrow{b}\}, \{\overrightarrow{b}\}, \{\overrightarrow{b}\}, \{\overleftarrow{b}, \overrightarrow{d}\}, \{\overleftarrow{d}\}\}$ is depicted in Figure 1.

In the fully modified weighted clique graph, edges correspond to adjacencies or telomeres. Thus, the problem is to choose edges from the fully modified weighted clique graph such that the corresponding adjacencies and telomeres form a maximum perfect and covering set. From Definition 4 it is clear that the adjacencies or telomeres corresponding to any two edges in the fully modified weighted clique graph are perfect if and only if the edges share no vertex in common or correspond to the same edge (since there are no loops, there are no edges that correspond to adjacencies of the form $\{x, x\}$ for some extremity x). Finding a set of edges that share no vertices in common is the famous *maximum matching problem*, although, since our edges are weighted, in this case we are actually more interested in the *maximum weight matching problem* [5].

Given a matching, Algorithm 3 constructs its corresponding perfect set. However, it is the quality of the matching that determines the quality of the perfect set. The fully modified weighted clique graph was defined such that the weight of the maximum weight matching is equal to the weight of the perfect set defined by Algorithm 3. Thus, given a maximum weight matching, Algorithm 3 will construct a maximum perfect set:

Algorithm 3. PERFECTSET

Input: A genome $\tilde{G}$ and a matching M.
Output: A perfect set P.

```
P ← ∅
foreach edge e ∈ M do
    let e = {u, v} for vertices u and v
    if v is a null vertex then
        add the telomere that corresponds to u to P
    else if u is a null vertex then
        add the telomere that corresponds to v to P
    else
        add the adjacencies that corresponds to u and v to P
    end
end
return P
```

Lemma 1. *Given a genome $\tilde{G}$, let M be a matching of $\mathcal{CG}(\tilde{G})$ and let $P =$ PERFECTSET$(\tilde{G}, M)$. If M is a maximum weighted matching then P must be a quasi maximum perfect set.*

Proof. Assume towards contradiction that M is a maximum weight matching but P is not a quasi maximum perfect set then there must exist a perfect set P' such that $\mathcal{W}(P) < \mathcal{W}(P')$. Let M' be the matchings that correspond to P', since the weight of the perfect set and its corresponding matching are the same, $\mathcal{W}(M) < \mathcal{W}(M')$. But M is a maximum weight matching, a contradiction. □

While in many cases the maximum weight matching will correspond to a quasi maximum perfect set that also covers the genome, simply finding a maximum weight matching of the fully modified weighted clique graph does not guarantee that the perfect set will be covering. Fortunately, any matching can be easily modified by Algorithm 4 so that this is the case:

Algorithm 4. EXTENDEDMAXIMUMWEIGHTMATCHING

Input: A fully modified weighted clique graph $\mathcal{CG}(\tilde{G})$ of a genome $\tilde{G}$.
Output: A maximum weight matching M of $\mathcal{CG}(\tilde{G})$ where every non-null vertex in $\mathcal{CG}(\tilde{G})$ is incident with an edge in M.

```
M ← MaximumWeightMatching(CG(G̃))
foreach non-null vertex v in CG(G̃) do
    if v is not incident with an edge in M then
        let u be the null vertex adjacent to v
        add {u, v} to M
    end
end
return M
```

Lemma 2. *Given a genome* $\tilde{G}$, $M = \textsc{ExtendedMaximumWeightMatching}(\tilde{G})$ *is a maximum weight matching such that* $P = \textsc{PerfectSet}(\tilde{G}, M)$ *covers* $\tilde{G}$.

Proof. Since every non-null vertex corresponds to an extremity in a genome, the perfect set will only be covering if every non-null vertex is incident with an edge in the matching from which the perfect set was created.

Algorithm 4 first finds any maximum weight matching M'. If every non-null vertex is incident with an edge in M' then the algorithm does not modify the matching and the result is a maximum weight matching whose corresponding perfect set covers $\tilde{G}$.

Each non-null vertex has a null vertex to which only it is adjacent. For non-null vertices not incident in an edge in the maximum weight matching, Algorithm 4 adds the edge between this vertex and its null counterpart to the matching. Clearly, the resulting perfect set will cover $\tilde{G}$. Thus, we must ensure that the resulting set of edges is still a matching of maximum weight.

The resulting set of edges must be a matching since the non-null vertex is not incident with any edges in the matching and the null vertex is only adjacent with the non-null vertex. It must be a maximum weight matching since all weights in a fully modified weighted clique graph are non-negative.

Therefore, $M = \textsc{ExtendedMaximumWeightMatching}(\tilde{G})$ is a maximum weight matching such that $P = \textsc{PerfectSet}(\tilde{G}, M)$ covers $\tilde{G}$. □

From Lemma 1 and Lemma 2 we can conclude the following theorem:

Theorem 2. *Let* $M = \textsc{ExtendedMaximumWeightMatching}(\tilde{G})$ *and* $P = \textsc{PerfectSet}(\tilde{G}, M)$, *where* $\tilde{G}$ *is an unordered genome.* P *is a quasi maximum perfect and covering set of* $\tilde{G}$.

6 Implementation

Algorithm 5 is the complete breakpoint aliquoting algorithm bringing together all the algorithms discussed in the previous sections. It follows from Theorem 2 and Theorem 1 that Algorithm 5 produces the optimal breakpoint aliquoting.

Most of the algorithms discussed in this paper run in linear time, however, the best known algorithm for computing the maximum weight matching runs in

Algorithm 5. BreakpointAliquoting

Input: A genome G with all gene families of size p.
Output: A polyploid H with all gene families of size p.

```
1 M ← ExtendedMaximumWeightMatching(CG(G̃))
2 P ← PerfectSet(G̃, M)
3 H̃ ← Replicate(P, p)
4 H ← Reorder(H̃, G)
5 return H
```

$O(ev \log v)$ time[5] where e is the number of edges and v is the number of vertices. If we have a genome with n gene families of size p, the fully modified weighted clique graph will have $4n$ vertices and $(2p-2)n$ edges. Thus, our algorithm runs in $O(pn^2 \log n)$ time.

7 Double Cut and Join Distance

Double cut and join distance and breakpoint distance are closely related. BP distance counts the number of different elements between two genomes. For the most part this is the same as DCJ distance; it generally takes one DCJ operation to correct one difference between two genomes. However, sometimes a DCJ operation corrects two differences instead of one and the DCJ distance must take this into account; this is the only difference between the two distances and it occurs infrequently with most data sets encountered in practice.

For a detailed explanation of DCJ distance we refer the reader to [2]. However, to understand this section we must briefly explain how DCJ distance is computed. An *adjacency graph* is a bipartite graph with two sets of vertices labeled with the elements of the two genomes that are to be compared respectively. There is an edge connecting two vertices for each extremity in common between their corresponding adjacencies. Recall that to be compared using a distance algorithm like DCJ, both genomes must have the same genes, hence the same extremities, and, in the case of genomes with duplicated genes, must be ordered.

From the adjacency graph, the DCJ distance can be computed:

$$\mathcal{D}_{DCJ}(G,H) = |\mathcal{G}(G)| - c - \frac{i}{2} \tag{3}$$

where c is the number of cycles and i is the number of paths with an odd number of edges in the adjacency graph.

The following theorem gives the relation between the DCJ and BP distances:

Theorem 3. *Given two genomes G and H:*

$$\mathcal{D}_{DCJ}(G,H) \leq \mathcal{D}_{BP}(G,H) \leq 2 \cdot \mathcal{D}_{DCJ}(G,H) \tag{4}$$

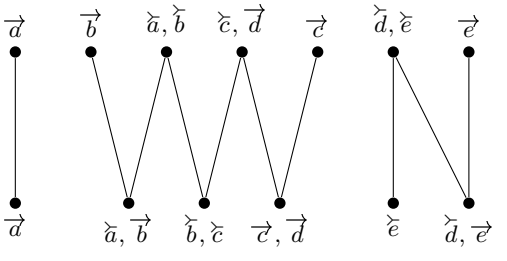


Fig. 2. An example of an adjacency graph

Proof. First, for $\mathcal{D}_{DCJ}(G,H) \leq \mathcal{D}_{BP}(G,H)$, observe that every adjacency shared between the two genomes causes a cycle of size 2 in the adjacency graph. It follows that $|\mathcal{A}(G,H)|$ is equal to the number of cycles of size 2 in the adjacency graph and, hence, less than or equal to the number of cycles in the adjacency graph. By similar reasoning, $|\mathcal{T}(G,H)|$ is equal to the number of paths of size 1 in the adjacency graph and, hence, less than or equal to the number of paths in the adjacency graph. It follows that $|\mathcal{A}(G,H)| + \frac{|\mathcal{T}(G,H)|}{2} \leq c + \frac{i}{2}$ where c is the number of cycles and i the number of odd paths in the adjacency graph. Since this factor decreases the distance, it follows that $\mathcal{D}_{DCJ}(G,H) \leq \mathcal{D}_{BP}(G,H)$.

Second, for $\mathcal{D}_{BP}(G,H) \leq 2 \cdot \mathcal{D}_{DCJ}(G,H)$, we will assume the most extreme case, where $|\mathcal{A}(G,H)| + \frac{|\mathcal{T}(G,H)|}{2} = 0$ but the number of cycles and paths in the adjacency graph is otherwise maximal. This produces the worst possible BP distance, $|\mathcal{G}(G,H)|$, but it is otherwise a maximal DCJ distance.

To determine the maximum number of cycles and paths in the adjacency graph, observe that every edge in the adjacency graph corresponds to one extremity shared between the genomes. Since $|\mathcal{A}(G,H)| + \frac{|\mathcal{T}(G,H)|}{2} = 0$ it follows that there are no cycles of size 2 and odd paths of size 1, so the smallest cycles and odd paths are of size 4 (since all cycles are even) and 3 respectively. Thus, the maximum number of cycles and odd paths is $\frac{|\mathcal{G}(G,H)|}{2}$. Thus, in the worst case scenario, BP distance is equal to twice the DCJ distance. □

Because BP distance can be equal to the DCJ distance, it is possible that the genome that results from Algorithm 5 could represent optimal aliquoting for both BP and DCJ. Because $\mathcal{D}_{BP}(G,H) \leq 2 \cdot \mathcal{D}_{DCJ}(G,H)$,the distance between the original genome and its optimal BP aliquoting is no more than twice the distance between the original genome and its optimal DCJ distance. Thus,

Theorem 4. *Algorithm 5 is a 2-approximation for the genome aliquoting problem using DCJ distance.*

8 Conclusion

Since the input to our algorithm is of size pn where p is the size of the gene families and n is the number of gene families, our algorithm runs in sub-cubic time. Thus, there exists a polynomial time algorithm that solves the genome aliquoting problem for BP distance. However, our algorithm is not without limitations.

The output of our algorithm consists of a mixture of linear and circular chromosomes with the only restriction being that adjacencies of the form $\{x, x\}$ for some extremity x are not permitted. In many cases it is desirable to restrict the output to something more similar to the input. For example, if the input consists of linear chromosomes then the output should consist of linear chromosomes. We believe it is possible to modify the output of our algorithm to provide the desired output in most cases without changing the distance. Some of the genome halving algorithms do precisely this [3,1]. On the other hand, sometimes circular chromosomes with adjacencies of the form $\{x, x\}$ are desirable in the output,

in which case it might be possible, in some cases, to get a better result than produced by our algorithm.

It remains an open problem whether or not a polynomial time algorithm for the genome aliquoting problem with DCJ distance exists. However, as we have shown a good approximation algorithm does exist. We remain optimistic that a polynomial time algorithm for DCJ distance can be found in the future.

References

1. Alekseyev, M.A., Pevzner, P.A.: Whole genome duplications and contracted breakpoint graphs. SIAM Journal on Computing 36(6), 1748–1763 (2007)
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
3. El-Mabrouk, N., Sankoff, D.: The reconstruction of doubled genomes. SIAM Journal on Computing 32, 754–792 (2003)
4. Feijão, P., Meidanis, J.: SCJ: a novel rearrangement operation for which sorting, genome median and genome halving problems are easy. In: Salzberg, S.L., Warnow, T. (eds.) Algorithms in Bioinformatics. LNCS, vol. 5724, pp. 85–96. Springer, Heidelberg (2009)
5. Lovász, L., Plummer, M.D.: Matching Theory. AMS Chelsea Publishing, Providence (2009)
6. Mixtacki, J.: Genome halving under DCJ revisited. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 276–286. Springer, Heidelberg (2008)
7. Sankoff, D., Blanchette, M.: The median problem for breakpoints in comparative genomics. In: Jiang, T., Lee, D.T. (eds.) COCOON 1997. LNCS, vol. 1276, pp. 251–264. Springer, Heidelberg (1997)
8. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. Bioinformatics 10, 120 (2009)
9. Warren, R., Sankoff, D.: Genome aliquoting with double cut and join. BMC Bioinformatics 10(1), S2 (2009)
10. Warren, R., Sankoff, D.: Genome halving with double cut and join. Journal of Bioinformatics and Computational Biology 7(2), 357–371 (2009)

The Problem of Chromosome Reincorporation in DCJ Sorting and Halving

Jakub Kováč[1,2], Marília D.V. Braga[2], and Jens Stoye[2]

[1] Department of Computer Science, Comenius University
kuko@ksp.sk
[2] AG Genominformatik, Technische Fakultät, Universität Bielefeld
mbraga@cebitec.uni-bielefeld.de, stoye@techfak.uni-bielefeld.de

Abstract. We study two problems in the double cut and join (DCJ) model: sorting – transforming one multilinear genome into another and halving – transforming a duplicated genome into a perfectly duplicated one. The DCJ model includes rearrangement operations such as reversals, translocations, fusions and fissions. We can also mimic transpositions or block interchanges by two operations: we extract an appropriate segment of a chromosome, creating a temporary circular chromosome, and in the next step we reinsert it in its proper place. Existing linear-time algorithms solving both problems ignore the constraint of reincorporating the temporary circular chromosomes immediately after their creation. For the restricted sorting problem only a quadratic algorithm was known, whereas the restricted halving problem was stated as open by Tannier, Zheng, and Sankoff. In this paper we address this constraint and show how to solve the problem of sorting in $O(n \log n)$ time and halving in $O(n^{3/2})$ time.

1 Introduction

During evolution, genomes undergo large-scale mutations: a segment of DNA can get reversed, or moved to another position. In genome rearrangement problems we try to find a shortest sequence of operations transforming one genome into another. Such a sequence explains the differences between the genomes and its length can be used to estimate the evolutionary distance.

The *double cut and join* (DCJ) operation, introduced by Yancopoulos et al. [1], models most of the large-scale mutation events, such as reversals, translocations, fusions, and fissions in a unified way. Furthermore, transpositions and block interchanges can be simulated by two operations: an appropriate segment of a chromosome is extracted, creating a temporary circular chromosome, which is then reinserted at the proper place in the next step.

The sorting algorithm given by Yancopoulos et al. [1] running in quadratic time guarantees that each new circular chromosome is immediately reincorporated, thus mimicking transpositions and block interchanges.

Bergeron et al. [2] restated the model and gave a simple linear-time algorithm for DCJ sorting ignoring the reincorporation constraint. However, the algorithm

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 13–24, 2010.

finds a sequence of DCJ operations without any explicit mention of the underlying operations (reversals, translocations, block interchanges, etc.) and many circular chromosomes may coexist in intermediate stages of the sorting process. Such sorting sequences are not biologically plausible e.g. in eukaryotic organisms that typically have only linear chromosomes.

In this work we revisit the original study of Yancopoulos et al. [1]. We borrow techniques from other studies on sorting by reversals and block interchanges [3, 4, 5] and propose a new algorithm that sorts multichromosomal linear genomes in the DCJ model, reincorporates circular chromosomes and runs in $O(n \log n)$ time.

Furthermore, we present a new result on the *halving* problem. In the halving problem we imagine a genome that underwent a whole genome duplication and then evolved by large-scale rearrangements. Given a present genome in which all markers are in two copies (paralogs), we try to reconstruct the genome right after the duplication, where each chromosome has its perfectly duplicated copy.

If no restriction on the linearity of chromosomes is imposed and no guarantee concerning circular reintegration is required, we can use linear-time algorithms proposed by Warren and Sankoff [6] and Mixtacki [7]. However, given a multilinear genome, these algorithms may predict some circular chromosomes in the ancestral genome. In the worst case, these algorithms may even produce $\Omega(n)$ circular chromosomes given a single linear chromosome of length n. Again, this is not biologically plausible, when organisms with linear genomes are considered.

The restricted halving problem has not been studied previously and is stated as open in [8]. In this paper we propose an algorithm to solve the halving problem for multichromosomal linear genomes with circular reincorporation in $O(n^{3/2})$ time.

The paper is organized as follows: in Section 2 we introduce the DCJ model and review the previous results, and in Section 3 we describe efficient data structures representing multilinear genomes. We solve the restricted versions of sorting and halving problems in Sections 4 and 5, respectively, and conclude in Section 6.

2 Preliminaries

Genome model. In the DCJ model, genomes Π and Γ consist of the same set of markers (genes, synteny blocks). Every marker g has two ends, called extremities, which we denote g^- and g^+.

Each extremity p is either adjacent to some other extremity q (two consecutive markers on a chromosome), or it is a telomere – the end of a linear chromosome. In the first case we say the extremities form an *adjacency* pq, in the second case we have a *telomeric adjacency* $p\circ$. Thus a genome is a set of adjacencies such that every extremity is in exactly one (possibly telomeric) adjacency.

For genome Π we can draw a genome graph G_Π where vertices are extremities and edges connect either adjacent extremities or two extremities of the same marker. Every vertex in this graph has degree 1 or 2, so the components of G_Π are paths and cycles. These components represent chromosomes – linear and circular, respectively.

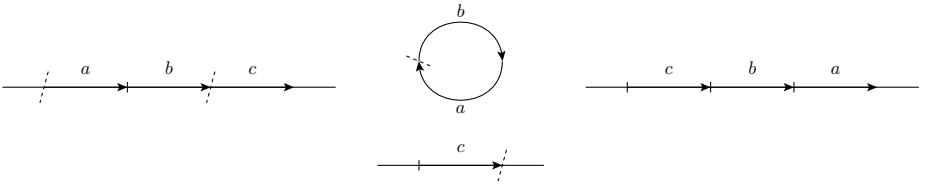


Fig. 1. To interchange blocks a and c (left) in the DCJ model we cut before a and after b and create a temporary circular chromosome (middle). The next operation cuts between a and b and after c and reincorporates the blocks in the correct order (right).

DCJ operation. A *double cut and join operation acting on adjacencies* pq *and* rs replaces them by either adjacencies pr, qs, or ps, qr (the adjacencies pq and rs can be telomeric or even an empty chromosome $\circ\circ$). We say the operation *cuts* pq and rs and *joins* either pr, qs, or ps, qr. By these operations we can mimic every common rearrangement operation in genomes: To invert a segment, we cut at its ends and join reversed. By cutting and joining adjacencies on different linear chromosomes, we get a translocation. By cutting two telomeric adjacencies $p\circ$ and $q\circ$ and joining pq we can fuse two chromosomes into one or create a circular chromosome from a linear one (as a byproduct we get an empty chromosome $\circ\circ$). By two DCJ operations we can mimic transpositions and block interchanges: We first cut out an appropriate segment and by joining its ends create a temporary circular chromosome. In the next step we reincorporate it into the original chromosome (see Fig. 1).

DCJ distance and scenarios. A sequence of k DCJ operations transforming a given genome Π into Γ is called a *DCJ scenario of length* k. A scenario of minimum length is called *optimal* and its length is the *DCJ distance between* Π *and* Γ, denoted $d(\Pi, \Gamma)$. A sequence of k DCJ operations transforming Π into Π' is *optimal* (with respect to Γ), if $d(\Pi, \Gamma) = d(\Pi', \Gamma) + k$.

The distance and a sorting scenario can be calculated using an adjacency graph $AG(\Pi, \Gamma)$. This is a bipartite graph where vertices are adjacencies of Π and Γ; an adjacency in Π is connected with an adjacency in Γ, if they share an extremity. Since every adjacency is connected with one (telomeric) or two other adjacencies, this graph consists of paths and cycles only. If Π and Γ share a *common adjacency*, this corresponds to a cycle of length 2 or path of length 1 (common telomere) in the adjacency graph. Note that when Π and Γ are equal, their adjacency graph consists of 2-cycles and 1-paths only.

The following theorem gives the DCJ distance between two genomes:

Theorem 1 (Bergeron et al. [2]). *Given two genomes* Π *and* Γ *on* n *markers, let* c *be the number of cycles and* p_o *the number of odd length paths in the adjacency graph* $AG(\Pi, \Gamma)$. *Then the distance between* Π *and* Γ *is*

$$d(\Pi, \Gamma) = n - (c + p_o/2) \ .$$

Halving problem. In the halving problem we imagine a genome that underwent a whole genome duplication and then evolved into genome Γ. We are given

Γ and our goal is to reconstruct the genome before the whole genome duplication. More formally: In a duplicated genome, every marker g has two copies – $g_1^- g_1^+$ and $g_2^- g_2^+$. If p is an extremity, we will denote by $\bar{p}$ the other copy – the paralogous extremity. Similarly, if $x = pq$ is an adjacency (possibly telomeric), then $\bar{x} = \bar{p}\bar{q}$ is the paralogous adjacency, and if C is a chromosome (set of adjacencies), then $\overline{C}$ is the set of paralogous adjacencies.

We say that genome Θ is *perfectly duplicated*, if for each adjacency pq in Θ, adjacency $\bar{p}\bar{q}$ is also in Θ and $p \neq \bar{q}$. This is the same as saying that if we ignore the subscripts (1's and 2's), every linear chromosome has an identical copy and every circular chromosome has either an identical copy, or is itself composed of two successive identical copies.

The genome halving problem can be stated as follows: Given a duplicated genome Γ, find a perfectly duplicated genome Θ such that $d(\Theta, \Gamma)$ is minimal.

The halving distance and scenario can be calculated using an analogy of an adjacency graph – a natural graph $NG(\Gamma)$ introduced by El-Mabrouk and Sankoff [9]. Vertices of this graph are adjacencies of Γ, and two adjacencies are connected by an edge, if they share a paralogous extremity. The natural graph consists of paths and cycles only, and Θ is perfectly duplicated if and only if $NG(\Theta)$ consists of 2-cycles and 1-paths only.

The following theorem gives the DCJ halving distance:

Theorem 2 (Mixtacki [7]). *Let Γ be a duplicated genome with $2n$ markers. The minimal distance between Γ and any perfectly duplicated genome Θ is*

$$d(\Gamma, \Theta) = n - (c_e + \lfloor p_o/2 \rfloor),$$

where c_e is the number of even cycles and p_o the number of odd paths in the natural graph $NG(\Gamma, \Theta)$.

Linear chromosomes. From now on we will be interested in genomes with linear chromosomes only. These *multilinear* genomes are more comfortably written as signed permutations: Choose a direction of a linear chromosome, and list the markers from left to right; write $\overrightarrow{g}$, if extremity g^- is before g^+ and $\overleftarrow{g}$ otherwise. Thus chromosome $(\overrightarrow{1}, \overrightarrow{3}, \overleftarrow{2})$ (which is the same as $(\overrightarrow{2}, \overleftarrow{3}, \overleftarrow{1})$) corresponds to the set of adjacencies $\{ \circ 1^-, 1^+ 3^-, 3^+ 2^+, 2^- \circ \}$. We will write $-g$ for the reversed marker g, i.e. $-\overleftarrow{g} = \overrightarrow{g}$ and $-\overrightarrow{g} = \overleftarrow{g}$.

Restricted sorting and halving. Given multilinear genomes, we call a sorting or halving DCJ scenario *restricted*, if every DCJ operation that creates a circular chromosome is immediately followed by another operation that reintegrates it into the original chromosome. Such scenarios can be viewed as sequences of reversals, translocations, fusions, fissions (with weight 1) and block interchanges, which have weight 2, i.e. count as two operations. In the restricted sorting and halving problems, we are searching for restricted scenarios of minimal length. Note that in both cases the distance remains the same as in their unrestricted versions.

3 Data Structures for Handling Permutations

Our algorithms use two efficient data structures for handling permutations described by Kaplan and Verbin [10] and Han [11].

Tree-based data structure. The following data structure from [10] can be traced back to Chroback et al. [12]. It supports the following three operations in logarithmic time: find the i^{th} marker in a linear chromosome, return the position of marker g and perform a reversal operation.

Linear chromosomes can be represented by a balanced tree supporting operations split and merge (e.g. red-black tree or splay tree). The order is the same as the left-to-right order of markers on the chromosome. In each node of the tree we store one marker, its orientation, number of descendants and a reverse flag. A reverse flag being "on" signifies that the whole subtree is reversed. The reverse flag of node v can be cleared ("pushed down") by changing v's orientation, swapping its children and flipping their reverse flags.

Reversing a segment from i to j can be implemented as follows:

1. Find the i^{th} and j^{th} marker (using the information about sizes of subtrees and reverse flags).
2. Split the tree into three parts: T_1 with markers before i, T_3 with markers after j and T_2 with the segment from i to j.
3. Flip the reverse flag in the root of T_2, and
4. Merge T_1, T_2 and T_3.

We store a lookup table with a pointer to the corresponding node of a tree for every marker. In this way we can find the position of any marker in logarithmic time.

This data structure can be easily extended to multiple linear chromosomes and to support different operations such as translocations or block interchanges. Actually, we do not need to have one tree per chromosome: simply concatenate the chromosomes with a delimiter between them and in each node store the number of delimiters in its subtree. This way given a marker g we can tell on which chromosome it is by counting the number of delimiters before g and all the rearrangement operations can be performed using a few reversal operations.[1]

Block-based data structure. The second data structure by Kaplan and Verbin [10] is a two-level version of the previous one. This is the data structure used in the subquadratic algorithms for sorting by reversals [13] and sorting by translocations [14].

As with the previous data structure, we concatenate all the chromosomes using delimiters. We divide the whole sequence into blocks of size between $\frac{1}{2}\sqrt{n \log n}$ and $2\sqrt{n \log n}$. Note that there are $O(\sqrt{n/\log n})$ blocks and one block can contain several chromosomes. In each block we store an array of markers and a

[1] For example block interchange can be mimicked by 4 reversals; if we add sufficiently many delimiters at the end of the sequence (representing empty chromosomes), we can also mimic fusions and fissions.

tree-based data structure storing their paralogs ordered by *positions of the paralogs* in the genome. Furthermore, for each block we keep the number of markers in it and a reverse flag which signifies that the whole block is reversed. We have an additional lookup table with blocks and indices of the markers, so that we can tell the position of a given marker in constant time.

Finding the i^{th} marker can be done trivially in $O(\sqrt{n/\log n})$ time and can be improved to $O(\log n)$ by building a balanced tree over the blocks.

Reversal of a segment can be implemented in $O(\sqrt{n \log n})$ time as follows:

1. Find the two endpoints and split the two blocks (if necessary) so that the endpoints of the reversal correspond to the endpoints of blocks (we can temporarily break the invariant about the block size; the trees with paralogs are rebuilt from scratch in $O(\sqrt{n \log n})$ time).
2. Reverse the order of the blocks between the endpoints and flip their reverse flags.
3. For each block (inside and outside the reversal) take its tree with paralogs T and split it into three parts: T_2 with paralogs within the reversal, T_1 with paralogs before and T_3 with paralogs after the reversal. If a block is outside the reversal, flip the reverse flag in the root of T_2, otherwise flip the flags in roots of T_1 and T_3. Merge T_1, T_2 and T_3. Since there are $O(\sqrt{n/\log n})$ blocks and all the split and merge operations can be done in $O(\log n)$ time, this step can be implemented in $O(\sqrt{n \log n})$ time.
4. Split and merge blocks so that the size of each one is between $\frac{1}{2}\sqrt{n \log n}$ and $2\sqrt{n \log n}$.

Again, we can simulate any other DCJ operation by a constant number of reversals.

The neat thing about this block-based data structure is that we actually maintain the markers according to two orders – by their position in the genome and by the position of their paralogs. This property can be used to implement the following query used in our halving algorithm in $O(\sqrt{n \log n})$ time: Given a chromosome C and two markers i and j on a possibly different chromosome, find the right-most marker on C that has a paralog between markers i and j:

1. Temporarily split the blocks at the ends of chromosome C, so that C is contained in several whole blocks.
2. Find the rightmost block within C containing a marker with paralog between i and j. Since the paralogs are stored in balanced trees, the membership questions can be answered in $O(\log n)$ time and there are $O(\sqrt{n/\log n})$ blocks.
3. In this block find the required marker by a sequential search in $O(\sqrt{n \log n})$ time.
4. Merge the temporarily split blocks.

Note that only a slightly more complicated data structure achieving time complexity $O(\sqrt{n})$ for the same operations was given by Han [11]. We refer the interested reader to this paper.

4 Restricted DCJ Sorting

Previous work. Bergeron et al. [2] gave a linear-time algorithm for DCJ sorting disregarding the constraint of reincorporating circular chromosomes immediately. The solution can be easily adapted to a quadratic algorithm for the restricted version: after each step check whether a circular chromosome was created and if so, find the appropriate DCJ operation acting on adjacencies in the circular and the original linear chromosome that reintegrates the circular chromosome. It is not obvious how to do this fast (say in polylogarithmic time).

Yancopoulos et al. [1] proposed to transform Π into Γ by restricted sorting in four stages: 0. Add caps to the ends of linear chromosomes. 1. By translocations, fusions and fissions transform Π into Π' such that chromosomes in Π' and Γ have the same marker contents. 2. Perform oriented reversals to get Π'' with all markers in the same direction as in Γ. 3. Finally, use block interchanges to transform Π'' into Γ.

Stages 2 and 3 can be implemented in $O(n \log n)$ time using the data structure described in Section 3 [5,4]. Thus a *unichromosomal* restricted DCJ sorting can be solved in $O(n \log n)$ time. However, it is not obvious how to implement stage 1 in a fast way.

Capping. The ends of linear chromosomes, telomeres, produce some difficulties and nasty special cases. Capping is an elegant technique to deal with them: we adjoin new markers (caps) to the ends so that we do not change the distance and we do not have to worry about telomeres any more.

We find all the paths in the adjacency graph $AG(\Pi, \Gamma)$. Paths of odd length have one end in Π and one in Γ – simply adjoin a new marker (properly oriented) to the two telomeres. This increases the number of markers by one, but instead of an odd path we have a cycle and a 1-path, so the distance does not change. For paths starting and ending in Π add two new markers to the ends of Π and a new chromosome consisting of just these two markers (properly oriented) to Γ. The case with a path starting and ending in Γ is symmetric. The number of markers increases by 2, but instead of an even path, we have a cycle and two odd paths, so the distance does not change. Capping of all chromosomes can be done in linear time.

Our algorithm. The algorithm is based on the following observation:

Observation 1. *Let g, h be two markers that are adjacent in Γ, but not in Π. If g and h are on different chromosomes in Π, there is a translocation that puts them together. This is an optimal operation in the DCJ model. If g and h are on the same chromosome and have a different orientation, there is a reversal that puts them together. This is an optimal operation in the DCJ model. Transposition and block interchange take two DCJ operations. These operations are optimal if they create two new non-telomeric common adjacencies and destroy none.*

This is simply because, even more generally, k operations that create k new non-telomeric adjacencies and destroy none create k new cycles in the adjacency graph, and thus decrease the distance by k.

Theorem 3. *A restricted optimal DCJ scenario transforming multilinear genome* Π *into* Γ *can be found in* $O(n \log n)$ *time.*

Proof. Cap all chromosomes first. Without loss of generality we may assume that the markers in chromosomes of Γ are consecutive numbers $(\overrightarrow{k_0}, \ldots, \overrightarrow{k_1 - 1})$, $(\overrightarrow{k_1}, \ldots, \overrightarrow{k_2 - 1})$, $\ldots$,$(\overrightarrow{k_{s-1}}, \ldots, \overrightarrow{k_s - 1})$ where $0 = k_0 < k_1 < k_2 < \cdots < k_s = n$ (otherwise renumber the markers).

We will be transforming Π into Γ gradually "from left to right": once we have transformed the beginning of a chromosome in Π to $\overrightarrow{k_i}, \overrightarrow{k_i + 1}, \ldots, \overrightarrow{j}$, we extend it by moving $j + 1$ next to $\overrightarrow{j}$.

There are several cases we need to consider:

1. If $\overrightarrow{j+1}$ is already next to $\overrightarrow{j}$, we are done.
2. If $j+1$ is on a different chromosome than $\overrightarrow{j}$, we can always use a translocation. In the rest of the proof we assume that $j+1$ is on the same chromosome, to the right of $\overrightarrow{j}$.
3. If $\overrightarrow{j}$ and $\overleftarrow{j+1}$ have different orientation, we can use a reversal.

Otherwise, following [3], find the marker m with the highest number between $\overrightarrow{j}$ and $\overrightarrow{j+1}$ and find $m + 1$.

4. If $m + 1$ is on a different chromosome, we can use a translocation to move it next to m; this operation also moves $\overrightarrow{j+1}$ to another chromosome, so we can use another translocation to move it next to $\overrightarrow{j}$.

Otherwise the situation is $\overrightarrow{j}, \ldots, m, \ldots, \overrightarrow{j+1}, \ldots, m+1$ (since m is the highest number between $\overrightarrow{j}$ and $\overrightarrow{j+1}$ and the part of the chromosome to the left of $\overrightarrow{j}$ is already sorted, $m + 1$ must be to the right of $\overrightarrow{j+1}$).

5. If m and $m + 1$ have different orientation, we can use a reversal to move $m+1$ next to m; this will also change the orientation of $\overrightarrow{j+1}$, so in the next step we can use another reversal to move $\overleftarrow{j+1}$ next to $\overrightarrow{j}$.
6. Finally, if m and $m + 1$ have the same orientation, we interchange blocks

$$\overrightarrow{j}, [\ldots, \overrightarrow{m}], \ldots, [\overrightarrow{j+1}, \ldots], \overrightarrow{m+1} \quad \rightsquigarrow \quad \overrightarrow{j}, \overrightarrow{j+1}, \ldots, \overrightarrow{m}, \overrightarrow{m+1}$$

if both $\overrightarrow{m}$ and $\overrightarrow{m+1}$ have positive direction and

$$\overrightarrow{j}, [\ldots], \overleftarrow{m}, \ldots, [\overrightarrow{j+1}, \ldots, \overleftarrow{m+1}] \quad \rightsquigarrow \quad \overrightarrow{j}, \overrightarrow{j+1}, \ldots, \overleftarrow{m+1}, \overleftarrow{m},$$

if $\overleftarrow{m}$ and $\overleftarrow{m+1}$ have both negative direction. By two operations we move $\overrightarrow{j+1}$ to $\overrightarrow{j}$ and $\overrightarrow{m}$ to $\overrightarrow{m+1}$.

Every step can be implemented in $O(\log n)$ time using an extended version of the data structure from Section 3. We need the data structure to support the following operations: 1. Given a marker, find the chromosome that contains it. 2. Given interval $i, \ldots, j$ find the marker with the highest number on the chromosome between i and j (store the highest number in the subtree in each node). 3. Perform a DCJ operation (this can be done by splitting, merging trees and lazy reversals as described). □

Perfect DCJ scenarios. Bérard et al. [15] studied the problem of finding a scenario transforming genome Π into Γ that does not break a given set of common intervals. An *interval* in genome Π is a set of markers such that the subgraph of G_Π induced by their extremities is connected. Intervals of Π have zero or two borders – adjacencies such that one extremity is inside and one outside. Let I be any set of markers with zero or two borders. A DCJ operation *preserves* I, if I still has zero or two borders in the resulting genome. They showed that for nested sets of common intervals (when the intervals do not overlap) the shortest scenario can be found in polynomial time and for weakly separable sets the problem is NP-hard, but fixed parameter tractable.

Since their algorithm uses algorithms for DCJ distance and sorting as a black box, one can use it in conjunction with our algorithm to get perfect restricted DCJ scenarios.

5 Restricted DCJ Halving

Previous work. The halving problem in the DCJ model was studied by Warren and Sankoff [6] and corrected and simplified by Mixtacki [7]. However, the restricted version of the halving problem has not been studied and is stated as open by Tannier, Zheng, and Sankoff [8].

The simple approach that works for sorting – do a DCJ operation, test whether a circular chromosome was created and reincorporate it – does not work for halving: In some cases the circular chromosome cannot be reincorporated. For example take chromosome $(1_1, 1_2, 2_1, 2_2)$ – after excision of circular chromosome $[1_1, 1_2]$ it is not possible to reincorporate it and the algorithm of Mixtacki [7] ends with two (perfectly duplicated) circular chromosomes $[1_1, 1_2]$ and $[2_1, 2_2]$. On the other hand, by fission and translocation we can get

$$(1_1 \mid 1_2, 2_1, 2_2) \quad \rightsquigarrow \quad (1_1 \mid),\ (1_2, 2_1 \mid 2_2) \quad \rightsquigarrow \quad (1_1, 2_2),\ (1_2, 2_1)\ .$$

By giving an algorithm for the restricted halving problem we also show that the halving distance is the same in the restricted and the unrestricted case.

Capping. Find all paths in the natural graph $NG(\Gamma)$. If the length of path is odd, adjoin two paralogous copies of a new marker at both ends. This will create a new 1-path and close the odd path into an even cycle. Thus we will have an extra marker and an extra cycle, the distance is unchanged. If the length of the path is even, we adjoin two different new markers at the ends and create a new linear chromosome consisting of its paralogous copies. The number of markers is increased by 2, but also the number of odd paths increases by 2 and number of even cycles increases by 1, so the distance is unchanged.

Our algorithm. An analogy of Observation 1 from the previous section holds also for the DCJ halving problem:

Observation 2. *Reversals and translocations that create one and block interchanges that create two new non-telomeric common adjacencies and destroy none are optimal.*

Theorem 4. *A restricted optimal DCJ halving scenario transforming duplicated genome* Π *into perfectly duplicated genome* Θ *can be found in* $O(n^{3/2})$ *time. The distance is the same as the unrestricted halving distance.*

Proof. Cap all chromosomes first. This can be done in linear time. Take any chromosome C_1; let i be its first marker (cap). Then there are two cases: the paralog $\overline{i}$ is either on a different chromosome C_2, or it is at the end of C_1 reversed.

In general, we have either two chromosomes C_1 and C_2 starting with paralogous markers, say $i, \ldots, j$ and $\overline{i}, \ldots, \overline{j}$ (with the same orientation), or there is chromosome C starting and ending with paralogous markers (in the opposite orientation).

Case I. The transformation will again go "from left to right": once we have C_1 starting with markers $i, \ldots, j, k$ and C_2 starting with $\overline{i}, \ldots, \overline{j}, \overline{\ell}$, we either move $\overline{k}$ next to $\overline{j}$ in C_2, or ℓ next to j in C_1.

If ℓ is not in C_1, we can use a translocation. Otherwise, if j, ℓ and $\overline{j}, \overline{\ell}$ have different orientation, we can use a reversal to move ℓ next to j. The situation is symmetric and the same holds for $\overline{k}$.

The only hard case arises when j, k, ℓ are on one chromosome, $\overline{j}, \overline{k}, \overline{\ell}$ on another, j, k have the same orientation as $\overline{j}, \overline{k}$ and j, ℓ the same as $\overline{j}, \overline{\ell}$. Then we can write the two chromosomes as

$$C_1 = (i, \ldots, j, k, x_1, \ldots, x_p) \quad \text{and} \quad C_2 = (\overline{i}, \ldots, \overline{j}, y_1, \ldots, y_q, \overline{k}, z_1, \ldots, z_r) \ .$$

We find marker y between $\overline{j}$ and $\overline{k}$ such that its paralog $\overline{y}$ is the rightmost among the x-markers on chromosome C_1. (Note that such a marker exists, since at least $y_1 = \overline{\ell}$ is a marker between $\overline{j}$ and $\overline{k}$ with paralog in C_1.) Let $\overline{y} = x_t$ and let $\overline{z} = x_{t+1}$ – then z is either one of the z-markers on the second chromosome C_2, or it does not lie on C_2 at all. In the latter case we perform two translocations moving first z to y and then $\overline{k}$ to $\overline{j}$, in the former we perform (depending on the mutual orientation of y, z and $\overline{y}, \overline{z}$) either two reversals or one of the two indicated block interchanges:

$$\begin{aligned} & (\overline{i}, \ldots, \overline{j}, \big[y_1, \ldots, y\big], \ldots, y_q, \big[\overline{k}, z_1, \ldots\big], z, \ldots, z_{r+1}) \\ \text{or} \quad & (\overline{i}, \ldots, \overline{j}, \big[y_1, \ldots\big], y, \ldots, y_q, \big[\overline{k}, z_1, \ldots, z\big], \ldots, z_{r+1}) \ . \end{aligned}$$

This way we put $\overline{k}$ next to $\overline{j}$ and y next to z at the same time.

Case II. Chromosome C starts and ends with paralogous markers $i, \ldots, j$:

$$C = (i, \ldots, j, k, \ldots\ldots, -\overline{\ell}, -\overline{j}, \ldots, -\overline{i}) \ .$$

The transformation will go "from outside to the middle": we either move $\overline{k}$ next to $\overline{j}$ or ℓ next to j.

Again, if one of the markers is on a different chromosome, or has opposite orientation, we can move it by a translocation or a reversal to its proper place.

Otherwise, find the leftmost marker m between k and $-\overline{k}$ such that $\overline{m}$ is *not* between k and $-\overline{k}$. If such an m exists, let n be the marker preceding m; note

that $\overline{n}$ is between k and $-\overline{k}$. Depending on the orientation of n, m and $\overline{n}, \overline{m}$ and whether $\overline{m}$ is in C or on a different chromosome, we perform either two translocations, two reversals, or a block interchange moving $\overline{k}$ to $\overline{j}$ and $\overline{n}$ to $\overline{m}$. The situation is symmetric and we can try the same with ℓ.

Finally, if in either case such an m does not exist, we have chromosome

$$C = (i, \ldots, j, k,\ x_1, \ldots, x_{2p},\ -\overline{k},\ y_1, \ldots, y_q,\ \ell,\ z_1, \ldots, z_{2r},\ -\overline{\ell}, -\overline{j}, \ldots, -\overline{i}),$$

where $I = \{x_1, \ldots, x_{2p}\}$ and $J = \{z_1, \ldots, z_{2r}\}$, possibly empty, are closed under taking paralogs – these intervals either contain both paralogs g and $\overline{g}$ or neither one. In this case we first recursively reorder markers in I and J and transform C into

$$C' = (i, \ldots, j,\ k, k_1, \ldots, k_p, \left[-\overline{k}_p, \ldots, -\overline{k}_1, -\overline{k}\right],\ y_1, \ldots, y_q,$$
$$\ell, \ell_1, \ldots, \ell_r, \left[-\overline{\ell}_r, \ldots, -\overline{\ell}_1, -\overline{\ell}\right],\ -\overline{j}, \ldots, -\overline{i}),$$

(the case analysis is the same as Case II) and then perform the indicated block interchange.

At the end we may end up with several chromosomes of the form $C - \overline{C}$, which should be fissioned in two perfectly duplicated chromosomes.

This method shows that the halving distance is the same as the unrestricted distance and the algorithm can be easily implemented in quadratic time. For a more efficient algorithm we need the data structure that supports the following operations: 1. Given a marker find the chromosome that contains it. 2. Given an interval $i, \ldots, j$, find the right-most marker on a given chromosome that has a paralog in the interval $i, \ldots, j$. 3. In a given interval $i, \ldots, j$ find the leftmost/ rightmost marker m such that $\overline{m}$ is not in the interval. 4. Perform all the DCJ operations. Using the block-based data structure from Section 3 and the improvement by Han [11] every operation can be performed in $O(\sqrt{n})$ time. □

6 Conclusion

In this work we revisited the restricted DCJ model for multichromosomal linear genomes, where a temporary circular chromosome is immediately reincorporated after its excision. We improved the quadratic algorithm by Yancopoulos et al. [1] and proposed an algorithm that runs in $O(n \log n)$ time.

Furthermore we solved an open problem from [8] by giving an algorithm for the restricted halving problem. The algorithm shows that the halving distance for the restricted version is the same as the distance for the unrestricted version and given a multilinear duplicated genome an optimal *multilinear* perfectly duplicated genome can always be found.

This is not the case for example in the median problem which we did not study and is still open: Consider three linear genomes $(1, 2, 3)$, $(2, 1, 3)$ and $(2, 3, 1)$. Their median in the unrestricted case consists of linear chromosome $(2, 3)$ and circular [1]. The circular genome [1] can be reincorporated into any of the given chromosomes by one operation giving the median score 3. This score however

cannot be achieved in the restricted model. Generalizing this pattern, we can get genomes of length $3n$ with unrestricted median score $3n$ and restricted median score $4n$.

Acknowledgements. The authors would like to thank Broňa Brejová for many constructive comments. The research of Jakub Kováč is supported by Marie Curie Fellowship IRG-224886 to Dr. Tomáš Vinař, VEGA grant 1/0210/10, and the Partnership Stipend from the Bielefeld University.

References

1. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
3. Christie, D.A.: Sorting permutations by block-interchanges. Inf. Process. Lett. 60, 165–169 (1996)
4. Feng, J., Zhu, D.: Faster algorithms for sorting by transpositions and sorting by block interchanges. ACM Transactions on Algorithms 3 (2007)
5. Swenson, K.M., Rajan, V., Lin, Y., Moret, B.M.E.: Sorting signed permutations by inversions in $O(n \log n)$ time. JCB 17, 489–501 (2010)
6. Warren, R., Sankoff, D.: Genome halving with double cut and join. JBCB 7, 357–371 (2009)
7. Mixtacki, J.: Genome halving under DCJ revisited. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 276–286. Springer, Heidelberg (2008)
8. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics 10 (2009)
9. El-Mabrouk, N., Sankoff, D.: The reconstruction of doubled genomes. SIAM J. Comput. 32, 754–792 (2003)
10. Kaplan, H., Verbin, E.: Sorting signed permutations by reversals, revisited. J. Comput. Syst. Sci. 70(3), 321–341 (2005)
11. Han, Y.: Improving the efficiency of sorting by reversals. In: Arabnia, H.R., Valafar, H. (eds.) BIOCOMP, pp. 406–409. CSREA Press (2006)
12. Chrobak, M., Szymacha, T., Krawczyk, A.: A data structure useful for finding hamiltonian cycles. Theor. Comput. Sci. 71, 419–424 (1990)
13. Tannier, E., Bergeron, A., Sagot, M.F.: Advances on sorting by reversals. Discrete Applied Mathematics 155, 881–888 (2007)
14. Ozery-Flato, M., Shamir, R.: An $O(n^{3/2}\sqrt{\log(n)})$ algorithm for sorting by reciprocal translocations. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 258–269. Springer, Heidelberg (2006)
15. Bérard, S., Chateau, A., Chauve, C., Paul, C., Tannier, E.: Computation of perfect dcj rearrangement scenarios with linear and circular chromosomes. JCB 16, 1287–1309 (2009), PMID: 19803733

Advances on Genome Duplication Distances

Yves Gagnon[1], Olivier Tremblay Savard[2], Denis Bertrand[2], and Nadia El-Mabrouk[2]

[1] DIRO, Université de Montréal, H3C 3J7, Canada
y.gagnon@umontreal.ca
[2] DIRO
olivier.tremblay-savard@umontreal.ca,
{bertrden,mabrouk}@iro.umontreal.ca

Abstract. Given a phylogenetic tree involving Whole Genome Duplication events, we contribute to the problem of computing the rearrangement distance on a branch of a tree linking a duplication node d to a speciation node or a leaf s. In the case of a genome G at s containing exactly two copies of each gene, the *genome halving problem* is to find a perfectly duplicated genome D at d minimizing the rearrangement distance with G. We generalize the existing exact linear-time algorithm for genome halving to the case of a genome G with missing gene copies. In the case of a known ancestral duplicated genome D, we develop a greedy approach for computing the distance between G and D that is shown time-efficient and very accurate for both the rearrangement and DCJ distances.

1 Introduction

Inferring the structure of ancestral genomes is a major step towards answering numerous biological questions such as the mechanisms of evolution, the variation in rearrangement and loss rates among the different branches of a phylogenetic tree, and the consequence of such variations on the genetic and physiological specificities of species. Even though manual approaches can not be avoided when analyzing specific biological datasets, the availability of automated methods can largely facilitate and orient the study [19]. In this context, since 1995, the computational biology community working on genome rearrangements has contributed to provide many accurate and rapid algorithms dedicated to the evolutionary study of a set of genomes represented as ordered sequences of genes [8,17,21]. However, most of these methods can not be applied to genomes with multiple gene copies, in particular genomes arising from whole genome duplication events.

Whole genome duplication (WGD) is a spectacular evolutionary event that has the effect of simultaneously doubling all the chromosomes of a genome. Right after a WGD, the resulting genome contains a complete set of duplicated chromosomes. However, this initial perfect duplicate status is obscured by subsequent rearrangement events and gene losses, eventually leading to an extant *rearranged duplicated genome* (RD genome) containing exactly two copies of each gene, or a *rearranged*

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 25–38, 2010.

duplicated genome with losses (RDL genome), containing at most two copies of each gene. Evidence of WGD events has shown up across the whole eukaryote spectrum, from the protist *Giardia* to the yeast species [12], including most plant lineages [10], several fishes [22], amphibians [23], and mammalian species [18].

Consider a set of genomes that have been subject to WGD events during their evolution and a phylogenetic tree reflecting the speciation events leading to these genomes. Then, under the assumptions that WGD is the only mechanism leading to gene duplicates and that at least one gene reflects the doubling status of each genome, WGD events can be placed on the phylogenetic tree as new internal nodes, called *WGD nodes* [25]. The *rearrangement phylogeny problem* seeks for ancestral gene orders leading to a most "plausible" evolutionary scenario. The parsimony approach is based on inferring gene orders at the internal nodes of the tree so that the sum of distances among all branches is minimized. When studying genome rearrangements, the most natural distance between two gene orders (distance on a branch) is the minimum number of rearrangements required to transform one gene order into the other. The rearrangements that have been most studied are inversions and reciprocal translocations (including fusion and fission). In the case of two genomes G and H with no gene duplicates and the same gene content, a key result is the Hannenhalli and Pevzner (HP) formula [15,21] for computing the rearrangement distance, leading to a polynomial-time algorithm. Another distance that has been extensively studied in the last years is the Double-Cut-and-Join (DCJ) distance which includes all known rearrangement events and gives rise to simpler formal results [5,6,24].

In the case of genomes with no gene duplicates, one of the main approaches to the rearrangement phylogeny problem is based on iterating an algorithm for the median problem to all overlapping triplets of the phylogenetic tree [7,8,16]. A prerequisite for applying such methodology to a phylogeny with WGD nodes is to be able to compute the distance on a branch of the phylogeny. However, this is far from being straightforward, as the orthology relationship between duplicated genes is not set. In particular, computing the distance between an RD genome G and a perfectly duplicated genome D (called the *double distance* in [11,20]) has been shown to be NP-hard for the DCJ distance [20]. When the ancestral genome D is unknown, the *genome halving problem* seeks for a perfectly duplicated genome D minimizing the rearrangement distance between G and D. In 2003, we have presented the first formal result related to genome duplication, which is an exact linear-time algorithm for solving the genome halving problem [9].

In this paper, we contribute to solving a number of problems related to the computation of the rearrangement and DCJ distances on a branch of a phylogenetic tree connecting a first WGD node to a speciation node or a leaf, in both cases of a known and an unknown preduplicated genome (label of the WGD node). In the case of an unknown ancestral genome, our result is a generalization of the genome halving algorithm to a genome G with missing gene copies (i.e. G is an RDL genome instead of an RD genome). In the case of a known ancestral genome D, we present a very efficient and accurate greedy heuristic for computing both the rearrangement and DCJ distance between G and D.

2 Preliminaries

Let Σ be a set of n genes. A *string* is a sequence of genes from Σ, where each gene is signed (+ or −) depending on its orientation. The *reverse* of a string $X = x_1x_2\dots x_r$ is the string $-X = -x_r\ -x_{r-1}\dots - x_1$. A *chromosome* is a string, and a *genome* is a collection of chromosomes. A *unichromosomal* genome has a single chromosome, and a *multichromosomal* genome has at least two nonnull chromosomes $C_1, C_2, \dots C_N$. A *circular chromosome* is a string $x_1 \dots x_r$, where x_1 is considered to follow x_r. A chromosome that is not circular is *linear*. To represent its endpoints, we add an "artificial gene", denoted O, at each extremity. In other words, a linear chromosome is a string of the form $Ox_1 \dots x_rO$.

In this paper, we consider both uni- and multichromosomal genomes. As most unichromosomal genomes are formed by a circular chromosome, and most multichromosomal genomes are formed by linear chromosomes, only circular unichromosomal genomes, and linear multichromosomal genomes are considered.

2.1 Evolutionary Events and Genomic Distances

The following evolutionary events apply to both uni- and multichromosomal genomes, except translocations (only relevant to the multichromosomal case).

- A *reversal* (or *inversion*) is an operation that replaces some proper substring of a chromosome into its reverse.
- A *translocation* between two chromosomes $X = X_1X_2$ and $Y = Y_1Y_2$ is an operation transforming the two chromosomes into X_1Y_2, Y_1X_2, or into $X_1(-Y_1), (-Y_2)X_2$. Two special cases of reciprocal translocations are *fusions* (if one of the two chromosomes generated by the translocation is an empty string) and *fissions* (if one of the two input chromosomes is the empty string).
- A *Whole Genome Duplication* (WGD) is an operation transforming a multichromosomal genome $G = \{C_1, C_2, \dots C_N\}$ into a multichromosomal genome $D = \{C_1, C'_1, C_2, C'_2, \dots C_N, C'_N\}$ containing $2N$ chromosomes where, for each $1 \leq i \leq N$, $C_i = C'_i$. In the case of a circular genome G represented by the string $x_1x_2\dots x_r$, a WGD transforms G into a circular genome D represented by either of the two strings : $x_1x_2\dots x_rx'_1x'_2\dots x'_r$, or $x_1x_2\dots x_r - x'_r\dots - x'_2 - x'_1$ where, for each $1 \leq j \leq r$, $x_j = x'_j$.
- Finally, a *loss* is an operation removing a chromosome proper substring.

A *rearrangement* will refer to an inversion or a translocation event. The *rearrangement distance* between two genomes G and H (with the same gene content or not), denoted $d_R(G, H)$, is the minimum number of rearrangements required in a scenario transforming G into H. In the case of genomes with single gene copies, computing the inversion and/or translocation distance has been shown to be a polynomial-time problem, and the best developed method runs in linear time [3,4].

Another distance that has been extensively studied in the last years is the DCJ distance [5,6,24]. Given a genome G, a Double-Cut-and-Join (DCJ) is an operation that "cuts" two adjacencies pq and rs in a genome, and replaces them

with either pr and qs, or ps and qr. The DCJ distance is an "artificial" one in the sense that some DCJ operations are not relevant from a biological point of view. However, it is interesting from a theoretical point of view as it leads to a unifying formula including all previously studied rearrangement events, as well as transpositions, for which no polynomial-time exact method is known. Computing the DCJ distance is a linear-time problem.

2.2 Genome Definitions

Let G be a genome defined on a set Σ of genes, i.e. g is in G iff $g \in \Sigma$.
• G is a *singleton genome* iff each gene is present exactly once in G.
• G is a *rearranged duplicated (RD) genome* iff each gene is present exactly twice.
• G is a *perfectly duplicated genome* (or *duplicated genome* for short) iff:

- *The multichromosomal case:* G is an RD genome containing an even number $2N$ of chromosomes, with two identical copies of each chromosome. If D is the set of the N different chromosomes, then we write $G = (D \oplus D)$.
- *The circular case:* G is an RD genome and there is a string D such that G is exactly D followed by D (we write $G = D \oplus D$), or D followed by $-D$ (we write $G = D \oplus -D$).

• G is a *rearranged duplicated genome with losses (RDL genome)* iff each gene in Σ is present at least once and at most twice in G.
• G is a *duplicated genome with losses (DL genome)* if each gene of Σ is present in one or two copies in G, and if a duplicated genome D can be obtained from G by an appropriate insertion of an additional copy of each *singleton* (gene present in one copy in G).

A DL genome A is said to be *induced* by an RDL genome G if each gene in Σ has the same copy number in A and G.

Let G be an RD genome and H be an RDL genome. We can define the evolutionary cost $\mathcal{E}(G, H)$ as the minimum number of inversions, translocations and losses required to transform G into H.

2.3 The Breakpoint Graph

In a series of papers published in 1995 [13,14,15], Hannenhalli and Pevzner (HP) developed polynomial-time algorithms for computing the rearrangement distance (inversion only, translocation only, or inversion+translocation) between two singleton genomes G and H on Σ. They all depend on a bicolored graph $\mathcal{B}(G, H)$, called the *breakpoint graph*, constructed as follows (Tesler's formalism [21]).

Graph $\mathcal{B}(G, H)$*:* If gene x of Σ has a positive sign, replace it by the pair $x^t x^h$, and if it is negative, replace it by $x^h x^t$. Then the set V of vertices of $\mathcal{B}(G, H)$ is the set of x^t and x^h for all x in Σ. Any two vertices of V that are adjacent in some chromosome in G, other than x^t and x^h deriving from the same x, are connected by a black edge (thick lines in Figure 2(b)), and any two adjacent

vertices in H are connected by a gray edge (thin lines in Figure 2(b)). Notice that adjacencies to O are not represented.

In the case of circular chromosomes, each vertex in V is incident to exactly one black and one gray edge, and thus the graph uniquely decomposes into $c(G, H)$ disjoint cycles of alternating edge colors.

In the case of G and H being multichromosomal genomes, let an *endpoint vertex of* G (resp. of H) be a vertex of V adjacent to O in G (resp. in H). Then any vertex has degree zero if it is an endpoint in both G and H, one if it is an endpoint in exactly one of the two genomes or two otherwise. Thus, the graph decomposes into $c(G, H)$ cycles and $p(G, H)$ paths of alternating edge colors. Note that a path may contain only one vertex and no edges. We denote by p_{GG} (resp. p_{HH}) the number of paths linking two endpoints of G (resp. of H). If G and H have the same number of chromosomes, then $p_{GG} = p_{HH}$. Otherwise, suppose w.l.o.g. that G has more chromosomes than H, then $p_{HH} \leq p_{GG}$.

The rearrangement distance: Although somehow different algorithms are required for sorting by translocation only, inversion only or inversion+translocation, all results in [13,14,15] (revisited by Tesler [21] for multichromosomal genomes) can be summarized by a unique formula given below:

$$\mathsf{HP}: \quad d_R(G, H) = n + N - C(G, H) + h(G, H)$$

where n is the number of genes, N is the number of chromosomes of G, and $C(G, H) = c(G, H) + p(G, H) - p_{GG}$. In the case of circular genomes, $N = p(G, H) = p_{GG} = 0$. As for $h(G, H)$, it is a correction parameter that has a different value depending on the considered model. In all cases, it is related to the decomposition of $\mathcal{B}(G, H)$ into components (maximal sets of crossing cycles). A component is termed good if it can be transformed into a set of cycles of size 1 by increasing the number of cycles at each step, and bad otherwise. The parameter $h(G, H)$ reflects the number of bad components of the graph. As the probability for a component to be bad is low, the value of $h(G, H)$ is usually low compared to the dominating parameter $C(G, H)$.

The DCJ distance: Based on the breakpoint graph, the DCJ distance between G and H can be expressed as follows [5,20]:

$$\mathsf{DCJ}: \quad d_{DCJ}(G, H) = n - \left(c(G, H) + \frac{p_{even}}{2} \right)$$

where p_{even} is the number of paths with an even number (≥ 0) of edges.

3 Genome Halving with Losses

Given an RD genome G, the *genome halving problem* is to find a duplicated genome D minimizing the rearrangement distance with G. In other words, we

define $d_R(G)$ as the minimum rearrangement distance between G and any duplicated genome D. Then the problem is to find a duplicated genome D such that $d_R(G) = d_R(G, D)$.

In [9] we have developed an exact linear-time algorithm, called Algorithm Dedouble, for the reversals-only version of the problem, the translocations-only version, and the version with both reversals and translocations. The approach was to start from a *partial breakpoint graph* $\mathcal{B}(G)$, i.e. the breakpoint graph with the set of edges restricted to the black edges representing G, and to complete this graph with a set of "valid" gray edges, i.e. gray edges representing a duplicated genome D, maximizing the number of cycles and paths (parameter $c(G, D)$ and $p(G, D)$ in the HP formula). The second step was then to perform modifications on the obtained graph in order to remove bad components that can be avoided, and obtain a duplicated genome D minimizing the rearrangement distance with G (i.e. minimizing the HP formula).

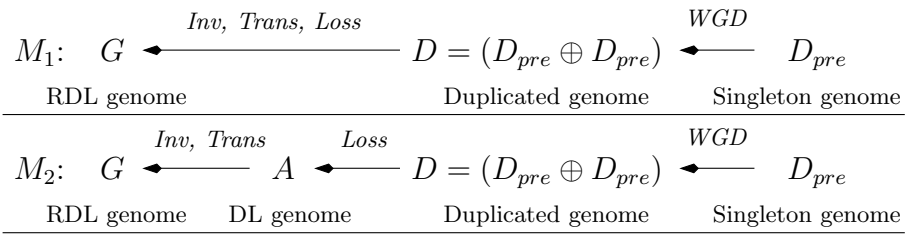


Fig. 1. Evolutionary models M_1 and M_2 considered for a present-day rearranged duplicated genome with losses G. Direction of evolution is represented by arrows orientation.

Here, we seek to generalize Algorithm Dedouble to a present-day genome G containing both duplicated genes and singletons, i.e. to an RDL genome. Let G be an RDL genome. We assume that G has evolved from an ancestral singleton genome through a WGD, and a sequence of inversions, translocations and loss events. We are then interested in finding such a pre-duplicated singleton genome D_{pre} minimizing the number of rearrangements needed to obtain G (see model M_1 in Figure 1). Note that we do not attempt to minimize the number of losses.

The following theorem allows to reduce the evolutionary model to a simpler one (model M_2 in Figure 1), where all losses occur first, followed by all rearrangement events.

Theorem 1. *Let G be an RDL genome and D be a duplicated genome. Then there exists a DL genome A induced by G such that $d_R(G, A) = d_R(G, D)$.*

Proof: Removed for space reason. Will be included in a full version of this paper.

Corollary 1. *Let G be an RDL genome, and A be a DL genome induced by G minimizing the cost $d_R(G, A)$. If D is the duplicated genome obtained from A, then $d_R(G) = d_R(G, D)$.*

Proof: Let A be a DL genome induced by G minimizing the cost $d_R(G, A)$, and D be the duplicated genome obtained from A. Then we have $d_R(G, D) = d_R(G, A)$. Suppose $d_R(G) \neq d_R(G, A)$, i.e. $d_R(G, A) > d_R(G)$. Let D' be a duplicated genome such that $d_R(G, D') = d_R(G)$. Then, from Theorem 1, there is a DL genome A' such that $d_R(G, A') = d_R(G, D') = d_R(G)$. And thus $d_R(G, A') < d_R(G, A)$, which is a contradiction with the fact that A minimizes the rearrangement cost. □

Therefore, finding a duplicated genome D such that $d_R(G) = d_R(G, D)$ can be reduced to the problem of finding a DL genome A induced by G such that $d_R(G, A)$ is minimal over all DL genomes induced by G. In other words, loss events can be ignored.

To find such DL genome A, we use a generalization of Algorithm Dedouble, called Algorithm Dedouble-RDL(G), that proceeds as follows:

1. Consider the RD genome G' obtained from G by "gluing" singletons to an adjacent gene. More precisely, consider a given orientation for chromosomes. Then, for each maximum sequence S of singletons in G: (1) if S is a chromosome, then just remove this chromosome; (2) otherwise, if S is connected to a left extremity of a chromosome, then replace its successor x (the gene representing the right adjacency of S in G) by the artificial gene $x' = Sx$; (3) otherwise, if S is not connected to a left extremity of a chromosome, then replace its predecessor x (possibly already updated in step (2)) by a new artificial gene x' representing the sequence xS.
2. Use Algorithm Dedouble to infer a duplicated genome A' from G'.
3. Recover a DL genome A from A' by replacing each of its artificial genes by its corresponding sequence of singletons, and by adding all removed chromosomes of G (formed exclusively by singletons).

The following theorem immediately follows from the fact that Algorithm Dedouble outputs a doubled genome A' minimizing the distance to G', and that singletons are preserved in the same order in G and A.

Theorem 2. *Let G be an RDL genome and A be the DL genome resulting from* Algorithm Dedouble-RDL(G). *Then $d_R(G, A) = d_R(G)$.*

4 An Algorithm for the Double Distance

Let G be an RD genome and $D = (D_{pre} \oplus D_{pre})$ be a duplicated genome. The problem of computing the DCJ distance between G and D has been shown to be an NP-hard problem [20], contrary to the polynomial-time complexity of computing the distance between two singleton genomes. This difference in complexity is the result of the missing one-to-one orthology relationship between the gene copies. In other words, given a labeling of the genes in G, the problem is to find a labeling of the genes in D leading to a minimum distance between G and D.

Consider a given beginning gene, in the case of a circular genome, or a given order and left-to-right orientation of chromosomes in the case of a multichromosomal genome G. Then, for each gene x (present in two copies in G and also in D), label the first occurrence of x in G as x_1 and the second as x_2. Let $\mathcal{B}(G)$ be the partial breakpoint graph for G. To complete this partial graph, each double adjacency (x^r, y^s) in D (where $r, s \in \{t, h\}$) should be represented in the completed graph $\mathcal{B}(G, D)$ by either of the following pairs of gray edges: $\{(x_1^r, y_1^s), (x_2^r, y_2^s)\}$, or $\{(x_1^r, y_2^s), (x_2^r, y_1^s)\}$. Each of these two cases leads to a different labeling of the gene copies in D. The problem is then to choose the pairs of gray edges allowing to minimize the HP formula in the case of the rearrangement distance, or the DCJ formula in the case of minimizing the DCJ distance.

Here, we focus on maximizing the dominating value $C(G, D)$ in the HP formula. In the case of genome halving, this simplification has been called the Weak Genome Halving Problem [2]. We similarly define our simplified problem as follows:

Weak Double Distance Problem. *For a given labeled RD genome G and a duplicated genome D, find a labeling of gene copies in D that maximizes the parameter $C(G, D)$ in the breakpoint graph $\mathcal{B}(G, D)$ of the labeled genomes G and D.*

Notice that, in the case of a circular genome, a labeling of D maximizing the parameter $C(G, D)$ also maximizes the DCJ formula, as $C(G, D) = c(G, D)$ in this case. In the multichromosomal case, a labeling of D maximizing $C(G, D)$ is likely to also maximize the DCJ formula, though there is no guarantee for that.

Clearly, the "best" exhaustive approach trying all possible labelings for D has a worst running-time complexity in $O(n.2^n)$ for $n = |\Sigma|$. Indeed, D has 2^n possible labelings, and for each labeling, the most efficient approach for computing the rearrangement distance between G and D is linear.

4.1 Circular Genomes

Let G be a circular RD genome and D be a circular duplicated genome. We consider the contracted breakpoint graph representation $\mathcal{CB}(D, G)$ defined as follows: the set of vertices of $\mathcal{CB}(D, G)$ is $V = \{x^r, \text{ for all } x \in \Sigma \text{ and } r \in \{t, h\}\}$. Any two vertices which are adjacent in D (except the extremities of a same gene) are connected by two parallel gray edges, and any two adjacent in G (except the extremities of a same gene) are connected by a black edge (see Figure 2 (a)). Such representation has previously been used in the context of genome halving for circular [1] and multichromosomal genomes [11], with the difference that each gray edge was represented exactly once. It follows that each vertex of $\mathcal{CB}(D, G)$ is adjacent to exactly two gray edges and two black edges.

Then, for each cycle of alternating edge color (just called cycle in the rest of this paper) in $\mathcal{CB}(D, G)$, there is a labeling of D giving rise to a corresponding cycle in $\mathcal{B}(G, D)$. This observation leads to a greedy approach for labeling the genome D, or equivalently completing the partial graph $\mathcal{B}(G)$. Formally, a *completed graph* $\mathcal{B}(G, D)$ is a graph obtained from $\mathcal{B}(G)$ by adding gray edges such

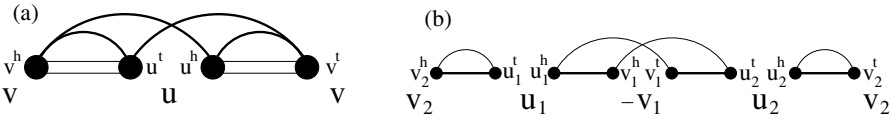


Fig. 2. (a) The contracted breakpoint graph $\mathcal{CB}(D, G)$ constructed for the circular RD genome $G = (u, -v, u, v)$ and the circular duplicated genome $D = (u, v) \oplus (u, v)$. Gray edges (thin lines) represent genome D and black edges (thick lines) represent genome G. (b) The breakpoint graph $\mathcal{B}(G, D)$ corresponding to the labeling $G = (u_1, -v_1, u_2, v_2)$ and $D = (u_1, v_1) \oplus (u_2, v_2)$. Given the above labeling of G, the labeling of D, leading to 3 cycles, is optimal. The resulting rearrangement distance is 1.

that each vertex of $\mathcal{B}(G, D)$ is adjacent to exactly 2 edges (one black and one gray), and such that the set of gray edges represent a given labeling of genome D (Figure 2 (b)).

The general idea of Algorithm Complete-Graph(G,D) given in Figure 3 is: at each step, pick a cycle of minimum size from $\mathcal{CB}(D, G)$, construct the corresponding cycle in $\mathcal{B}(G)$, and then remove from $\mathcal{CB}(D, G)$ all used edges. The algorithm stops when the partial graph is completed.

```
Algorithm Complete-Graph(G,D)
1.  For CSize = 1 to n Do ;
2.      For CVertex = b_1^l to b_n^l Do
3.          If CB(D,G) is empty (i.e. no edges left)
4.              Return ;
5.          If there is a cycle C_CB of size CSize beginning at CVertex Then
6.              Construct the corresponding cycle C_B in B(G);
7.              Remove from CB(D,G) all edges of C_CB;
8.          End If
9.      End For
10. End For
```

Fig. 3. A greedy approach for completing the partial graph $\mathcal{B}(G)$ with gray edges representing the genome D. Here, $n = |\Sigma|$ is the number of different genes, and $b_1, b_2, \ldots b_n$ is a left-to-right ordering of the black edges of $\mathcal{CB}(D, G)$. For each i, b_i^l is the vertex representing the left adjacency of b_i. The size of a cycle is the number of black (or equivalently gray) edges of the cycle.

The following proposition immediately follows from the fact that all gray edges of $\mathcal{CB}(D, G)$ are placed in $\mathcal{B}(G)$.

Proposition 1. *Given a circular RD genome G and a circular duplicated genome D, the output of* Algorithm Complete-Graph (G,D) *is a completed graph $\mathcal{B}(G, D)$.*

As each vertex is adjacent to two black edges, finding a cycle of size k beginning at a given vertex of $\mathcal{CB}(D, G)$ (line 5) can be done in $O(2^k)$ time. Therefore, the algorithm has a worst running-time complexity bounded by $\Sigma_{k=1}^{n} n.2^k$, which is not better than the exhaustive approach in $O(n.2^n)$. However, as demonstrated in the experimental part of this paper, it is actually a much faster approach in practice. This is due to the edge removal step (line 7), which allows to reduce the graph quickly, and to stop the process after a small number of iterations.

4.2 Multichromosomal Genomes

In the case of G and D being multichromosomal genomes, we define the contracted breakpoint graph $\mathcal{CB}(D, G)$ as before, except that it contains an additional vertex O such that any endpoint vertex in D is connected to O by two gray edges, and any endpoint vertex in G is connected to O by a black edge. It follows that, except O that is adjacent to $2N_G$ black edges and $2N_D$ gray edges, N_G being the number of chromosomes of G, and N_D the number of chromosomes of D, each other vertex is adjacent to exactly two gray edges and two black edges.

Algorithm Complete-Graph(G,D) can be used in the case of multichromosomal genomes if we replace line 3 with " **If** $\mathcal{CB}(D, G)$ is acyclic". The output of the algorithm is then an acyclic partially completed graph, where the only remaining paths connect two vertices that are both endpoints of G, or both endpoints of D (as a path connecting two endpoints of two different genomes would have been closed by Algorithm Complete-Graph (G,D) to form a cycle). Then, to complete the graph $\mathcal{B}(G)$, it suffices to add the remaining paths of $\mathcal{CB}(D, G)$.

Due to the $2(N_G + N_D)$ edges incident to O, the worst-time complexity is the one for circular genomes multiplied by $N_G.N_D$, i.e. $O(n.N_G.N_D.2^n)$. Hopefully in practice, n is not a tight upper bound as exploration eventually stops for much smaller cycle sizes.

5 Results

We focus on testing the performance of our greedy method for computing the double distance. We generated datasets through simulated evolutions between a duplicated genome D and an RD genome G for both circular and multichromosomal genomes, as follows.

Simulated datasets: We first determine n, the number of genes, and N, the number of chromosomes in D. Then, we generate D, and a series of rearrangement events are performed on D to obtain G. Those are simply the rearrangements allowed by our model, namely inversions only in the case of circular genomes or inversions and translocations (including fusions and fissions) in the case of multichromosomal genomes. The number of events, μ, is a parameter chosen prior to the data generation, and the size of each rearrangement is chosen randomly. As for the rates of rearrangement operations, we chose (Inv : Trans : Fus+Fiss) $= (5 : 4 : 1)$ to follow the rates reported in a lineage where a WGD occurred [12].

In order to validate the distances obtained with our greedy approach, we use an exact algorithm described below.

Exact algorithm: Let L (*resp.* L^*) be a complete (*resp.* partial) labeling of the gene copies of D, and $\mathcal{B}(G, D_{L^*})$ the breakpoint graph where the only defined gray edges are those adjacent to the genes of L^*. The idea is to compute a lower bound for $d_R(G, D)$ as we progressively construct L^*. More precisely, if at one step we have c cycles and p paths in $\mathcal{B}(G, D_{L^*})$, we know that the number of cycles in $\mathcal{B}(G, D_L)$ will be at most equal to $c + p$. Thus it is possible to use the following lower bound in a branch and bound strategy: $d_R(G, D) \geq n - c - p$.

Due to the high running-time complexity of the exact method, validation with the exact distance can only be done for "simple" datasets obtained with a low number of genes and a low number of rearrangements.

5.1 Time Efficiency

Since the running-time complexity is function of n for the exact approach, we generated genomes containing different number of genes to evaluate the time efficiency of our greedy heuristic. For the exact method, n varies from 10 to 100, with an increment of 10. The parameters μ and N are arbitrarily fixed to 15 and 4 respectively. For the greedy heuristic, n varies from 100 to 1000 with an increment of 100. With μ fixed to 15, the running-time of the heuristic does not vary (below 0.001 seconds for all values of n). Thus, the number of rearrangements has been changed to $\mu = n$ in order to see a variation in the running-time. For each of those n values, multiple datasets were generated and the running time was averaged.

We can clearly observe the exponential running-time of the exact approach when the number of genes increases (see Figure 4 Left). In contrast, our greedy algorithm is less limited by the genome size and more by the number of rearrangements. In Figure 4 Right, we can see that even for datasets with a high number of rearrangements ($\mu = n$), the running-time is less than the anticipated worst-time complexity and remains under, or close to 1 second.

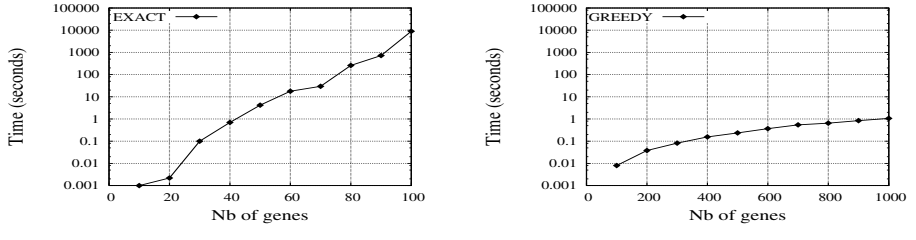


Fig. 4. Left: Running-time of the exact algorithm computing the double distance between D and G with various numbers of genes and a fixed number of rearrangements (μ=15). Right: Running-time of the heuristic approach to compute the double distance with various numbers of genes and rearrangements ($\mu = n$).

5.2 Heuristic Accuracy

Comparison with the exact approach. We now test whether our greedy heuristic infers an accurate rearrangement distance by comparing its results against those of the exact approach. Recall that because of the high running-time complexity of the exact approach, we can only perform this algorithm on simple datasets exhibiting low numbers of genes and rearrangements. The genomes are generated with n fixed to 25, N to 4 and μ varying from 0 to 50 by increments of 5. For each value of μ, 500 datasets were simulated. The error rate is the proportion of datasets for which the exact method found a more accurate distance than our greedy algorithm. Results are averaged over all datasets showing a comparable number of rearrangement events.

As observed in Figure 5, the error rate of the greedy approach is close to 0 when the number of rearrangements is less than 25. Notice that the distance inferred by the greedy algorithm is in average really close to the optimal distance for both types of genomes (circular and multichromosomal). When the distance is not the same, it differs in average by 1 rearrangement. Naturally, the error rate of the greedy approach is more apparent when the number of rearrangements increases. This behavior is due to the fact that when a high number of rearrangements is performed, different cycles of equal size can be selected and a choice must be made affecting the remaining set of cycles. As stated before, in this experiment we seek to optimize the rearrangement distance, but we obtain similar results if we seek to optimize the DCJ distance (results not shown).

Complex datasets. As a final experiment, simulations were performed with $n = 1000$, $N = 8$ and μ varying from 0 to 1000. The distances obtained with our greedy approach are compared with μ. Results shown in Figure 6 demonstrate that our method infers distances close to the number of rearrangement events performed on the original genome (for circular and multichromosomal genomes). However, when the number of rearrangement events increases, our approach underestimates that value. As in the comparison with the exact approach, the results are similar with the DCJ distance (results not shown).

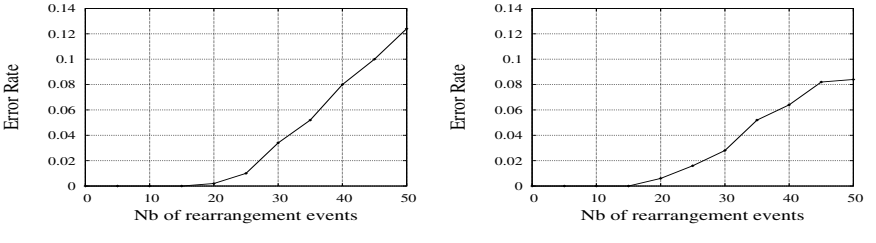


Fig. 5. Comparison of the greedy heuristic with the exact approach, showing the error rate of the inferred rearrangement distance for circular (Left) and multichromosomal genomes (Right)

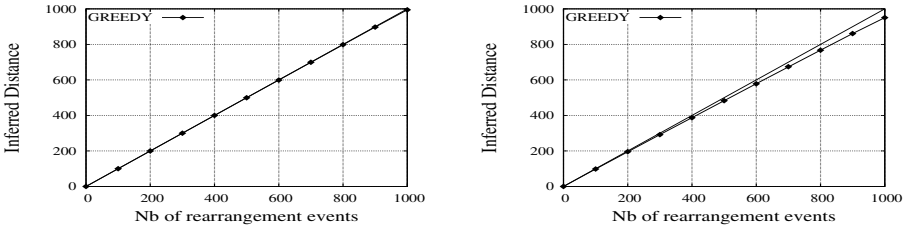


Fig. 6. Inferred rearrangement distances with complex datasets for circular genomes (Left) and multichromosomal genomes (Right)

6 Conclusion

We presented a linear time algorithm to solve the genome halving problem for genomes with missing gene copies. We also presented a greedy heuristic to compute the distance between an RD genome and a duplicated genome for the rearrangement and DCJ distances. Our experiments on simulated datasets showed that our greedy approach is time-efficient and accurate.

The proposed heuristic for the double distance could be easily generalized to compute the distance between two RD genomes. Moreover, it could be adapted to genomes that have undergone more than one WGD, but with an increase in running-time complexity, as the number of possible labelings for a gene would increase. Our algorithm could then be used for the rearrangement phylogeny problem with one or many WGDs, but without gene losses. However, there is evidence of massive gene losses in lineages that have undergone a whole genome duplication event [12]. Thus, another interesting future work will concern the generalization of Algorithm Complete-Graph(G,D) to an RDL genome G. Notice that the approach of Algorithm dedouble-RDL can not be used directly (i.e. removing the singleton genes from G and the corresponding copies in D, and reinserting them after having completed the breakpoint graph) because of the constraints imposed by the duplicated genome D.

References

1. Alekseyev, M., Pevzner, P.: Whole genome dup., multi-break rear., genome halv. prob. In: Proc. 18th annu. ACM-SIAM symp. discrete alg., pp. 665–679 (2007)
2. Alekseyev, M.A., Pevzner, P.A.: Colored de bruijn graphs and the genome halving problem. IEEE/ACM Trans. Comput. Biol. Bioinf. 4(1), 98–107 (2007)
3. Bader, D., Moret, B., Yan, M.: A linear-time algo. for comput. inv. dist. between signed perm. with an experimental study. J. Comp. Biol. 8, 483–491 (2001)
4. Bergeron, A., Mixtacki, J., Stoye, J.: Reversal distance without hurdles and fortresses. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 388–399. Springer, Heidelberg (2004)
5. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)

6. Bergeron, A., Mixtacki, J., Stoye, J.: A new linear time algo. to compute the genomic dist. via double-cut-join dist. Theo. Comp. Sci. 410(51), 5300–5316 (2009)
7. Blanchette, M., Kunisawa, T., Sankoff, D.: Gene order breakpoint evidence in animal mitochondrial phylogeny. J. Mol. Evol. 49, 193–203 (1999)
8. Bourque, G., Pevzner, P.A.: Genome-scale evolution: Reconstructing gene orders in the ancestral species. Genome Research 12, 26–36 (2002)
9. El-Mabrouk, N., Sankoff, D.: The reconstruction of doubled genomes. SIAM Journal on Computing 32(1), 754–792 (2003)
10. Salse, J., et al.: Ident. and charac. of shared dup. between rice and wheat provide new insight into grass genome evol. The Plant Cell 20, 11–24 (2008)
11. Gavranović, H., Tannier, E.: Guided genome halving ⋯ predup. ancestral genome of S. cere. In: Pacific Symp. Biocomput., vol. 15, pp. 21–30 (2010)
12. Gordon, J., Byrne, K., Wolfe, K.: Add., losses, and rear. on the evol. route from a recons. ancestor to the modern s. cerev. genome. PloS Genetics, 5(5) (2009)
13. Hannenhalli, S.: Polynomial-time algorithm for computing translocation distance between genomes. In: Galil, Z., Ukkonen, E. (eds.) CPM 1995. LNCS, vol. 937, pp. 162–176. Springer, Heidelberg (1995)
14. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). J. ACM 48, 1–27 (1999)
15. Hannenhalli, S., Pevzner, P.A.: Transforming men into mice. In: Proc. of the IEEE 36th Annual Symp. on Found. of Comp. Sci, pp. 581–592 (1995)
16. Moret, B., Wang, L., Warnow, T., Wyman, S.: New approaches for reconstructing phylogenies from gene order data. In: ISMB 2001, pp. 165 – 173 (2001)
17. Moret, B.M.E., Tang, J., Warnow, T.: Mathematics of Evolution and Phylogeny, ch. 12. Oxford Univ. Press, Oxford (2005)
18. Boore, J.L., Dehal, P.: Two rounds of whole genome duplication in the ancestral vertebrate. Plos. Biology 3(10), e314 (2005)
19. Sankoff, D.: Recons. the hist. of yeast genomes. PLOS Genetics 5(5) (2009)
20. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics 10(120) (2009)
21. Tesler, G.: Efficient algorithms for multichromosomal genome rearrangements. Journal of Computer and System Sciences 65, 587–609 (2002)
22. Postlethwait, J.H., et al.: Vertebrate genome evolution and the zebrafish gene map. Nature Genetics 18, 345–349 (1998)
23. Xu, R.H., et al.: Differential regulation of neurogenesis by the two xenopus gata-1 genes. Molecular and Cellular Biology 17, 436–443 (1997)
24. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic perm. by transloc., inver. and block interchange. Bioinformatics 21, 3340–3346 (2005)
25. Zheng, C., Zhu, Q., Sankoff, D.: Descendants of whole genome duplication within gene order phylogeny. J. Comp. Biol. 15(8), 947–964 (2008)

Listing All Parsimonious Reversal Sequences: New Algorithms and Perspectives

Ghada Badr[1,4], Krister M. Swenson[2,3], and David Sankoff[2]

[1] SITE, School of Information Technology and Engineering, University of Ottawa, Ontario, K1N 6N5, Canada
badrghada@gmail.com
[2] Department of Mathematics and Statistics, University of Ottawa, Ontario, K1N 6N5, Canada
[3] LaCIM, UQAM, Montréal (QC), H3C 3P8, Canada
[4] IRI - Mubarak city for Science and Technology, University and Research District, P.O. 21934 New Borg Alarab, Alex, Egypt

Abstract. In comparative genomics studies, finding a minimum length sequences of reversals, so called *sorting by reversals*, has been the topic of a huge literature. Since there are many minimum length sequences, another important topic has been the problem of listing all parsimonious sequences between two genomes, called the All Sorting Sequences by Reversals (ASSR) problem. In this paper, we revisit the ASSR problem for uni-chromosomal genomes when no duplications are allowed and when the relative order of the genes is known. We put the current body of work in perspective by illustrating the fundamental framework that is common for all of them, a perspective that allows us for the first time to theoretically compare their running times. The paper also proposes an improved framework that empirically speeds up all known algorithms.

1 Introduction

In this paper we focus on minimum length sequences of reversals that transform one genome into another. When no duplicate genes are allowed, genomes can be represented as permutations. For unsigned permutations, when only the order of genes is known, the sorting by reversal problem is NP-hard [7]. In 1995 Hannenhalli and Pevzner [11] showed that the signed versions of the problems can be solved in polynomial time. Since then, many refinements and speed improvements have been developed; the fastest known algorithms that find an optimal sequence are [13,16].

In 2002, Siepel and Ajana *et al.* [1,12] proposed $O(n^3)$ algorithms to list all sorting reversals. They called the problem of listing such reversals the All Sorting Reversals (ASR) problem since *sorting reversals* are the reversals that produce a permutation that is one step closer to the target permutation. By applying their algorithms repetitively, Siepel was able to generate All Sorting Sequences by Reversals (ASSR) that transform one genome into another. Recently, their methods have been improved due to an average-case $O(n^2)$ algorithm for the ASR problem [14].

Bergeron *et al.* [3] adopted the concept of *traces* [9] so as to group sequences into equivalence classes based on the commuting properties of reversals, which can be represented conveniently by a normal form. However, they provided no algorithm to do so.

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 39–49, 2010.

In 2007, Braga *et al.* [4,6], combined the results of Siepel and of Bergeron *et al.* by developing an algorithm that enumerates the normal form of every trace and provides the count of the number of sorting sequences. The implemetation of their algorithm (called "baobabLUNA") was shown to run much faster than that of Siepel in experiments.

More recent work by Baudet *et al.* [2] also uses normal forms of traces to represent classes of sorting sequences. However, their approach visits the normal forms in an economical way, a manner that allows them to generate normal forms depth-first as apposed to the inherently breadth-first approach of Braga. As a result, their algorithm eliminates the need for a potentially exponential amount of memory (or extensive disk use as in Braga's implementation of her algorithm). This also led to a speed-up of up to 11x on some input [2]. The drawbacks of their algorithm are that it cannot count the total number of solutions represented by the traces (counting the number of linear extensions of a poset is #P-complete [10]) and that it cannot always find traces when certain constraints are imposed on the sorting sequence [5].

In this paper we first survey the state of the art for solving the ASSR problem, illustrate the general framework that is common for all approaches, and experimentally and theoretically compare them. We then propose a new framework for solving the ASSR problem that empirically speeds up all known algorithms. Section 2 starts by giving definitions. Section 3 describes the general framework that is common for all current approaches. Section 4 details the particularities of each approach within the standard framework. We then prove that the algorithm of Baudet *et al.* is always faster than that of Braga. Section 5 proposes a new framework for exploring all sequences that is based on grouping permutations corresponding to partial solutions. It motivates the method and then discusses the application of the framework to each of the previous approaches. Section 6 provides the experimental setup and results showing the speed-up that can be achieved by applying the new model. Finally Section 7 concludes the paper.

2 Background

Consider a signed permutation $\pi = \pi_1, \ldots, \pi_n$ on the integers from 1 to n. Define a (signed) *reversal* ρ as a subset of $\{1,\ldots,n\}$, where the elements are ordered increasingly and appear contiguously in π. That way, reversals can be compared by a lexicographic order. Applying the reversal ρ, where ρ is a subset of elements from π_i to π_j that appear contiguously in the permutation π gives:

$$\pi \circ \rho = \pi_1, \ldots, {}^{-}\pi_j, \ldots, {}^{-}\pi_i, \ldots, \pi_n.$$

The *reversal distance* $d(\pi_1, \pi_2)$ is the smallest k such that $\pi_2 = \pi_1 \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_k$. Without loss of generality, since $I = \pi_2^{-1} \circ \pi_1 \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_k$, we consider $\pi_2 = I = 1,2,\ldots,n$ to be the identity permutation, and in that case, $d(\pi_1, \pi_2) = d(\pi_1, I) = d(\pi_1)$. Thus, a reversal ρ is a *sorting* reversal on π if $d(\pi \circ \rho) = d(\pi) - 1$. Siepel [12] called the problem of finding all individual sorting reversals the *All Sorting Reversals* (ASR) problem.

We can also define a *sorting i-sequence* as a sequence of sorting reversals $s = (\rho_1 \circ \rho_2 \circ \cdots \circ \rho_i)$ such that $d(\pi \circ s) = d(\pi) - i$. Thus, a *sorting sequence* is an i-sequence such that $d(\pi) = i$, and enumerating all such minimum length sorting sequences is called the *All Sorting Sequences by Reversal* (ASSR) problem.

3 The Current Framework for ASSR Approaches

Assume that we are sorting a permutation π with a reversal distance $d(\pi)$. An i-solution is a partial solution for the ASSR problem that represents a minimum length sorting sequence(s) of length i. Let the i-level be the set of all i-solutions. So every i-solution in an i-level will generate a permutation with distance equal to $d(\pi) - i$.

The current framework for solving the ASSR problem applies Siepel's $O(n^3)$ algorithm for ASR[1] repeatedly. This framework, ASSR-FW, can be described as follows. *For each i-solution:*

- Step 1: Generate the set of $(i+1)$-solutions in the following two steps:
  - Apply Siepel's ASR algorithm to the permutation corresponding to the i-solution, generating the next $O(n^2)$ sorting reversals.
  - Append each sorting reversal to the i-solution to generate a list of $O(n^2)$ $(i+1)$-solutions.
- Step 2: Add the $(i+1)$-solutions to the $(i+1)$-level.

These two steps are repeated iteratively until all $d(\pi)$-solutions are obtained.

4 Representations of Sorting Solutions

Current approaches for solving the ASSR problem enumerate sorting solutions as either *sorting sequences* or in a more compact form as *normal forms of traces*. In the following sub-sections we will discuss each of these representations and their differences, using the ASSR-FW as described in the previous section.

4.1 Generating Sorting Solutions Using Sorting Sequences

Siepel [12] was the first to enumerate all sorting solutions as sequences of sorting reversals. His algorithm (SE) design is basically the ASSR-FW described in Section 3, where solutions are represented as sequences. According to the ASSR-FW, the number of i-sequences at the i-level will be at most $n^2 * n^2 * \cdots * n^2$ i times. This implies that at any i-level there are at most n^{2i} i-sequences. Thus, the SE algorithm has to repeat the two steps in ASSR-FW $O(n^{2i})$ times for each i-level. Adding up the time complexity for all levels, SE lists all sorting sequences in $O(n^{2n+3})$ [6] time.

The algorithm is only feasible for small distances due to the huge number of sequences generated. For example, for the permutation (−10, 9, −8, 5, 11, −4, 2, 6, −7, 3, 1), the number of solutions is 8278540!.

[1] The framework we present in Section 5 also uses this algorithm. Our recent result [14] shows that ASR can be solved in $O(n^2)$ time on average, which would introduce a significant speed-up on all known algorithms for the ASSR problem. In this paper, we focus on the improvement gained through our new framework although using the two together is sure to provide even stronger results.

4.2 Generating Solutions Using Normal Forms of Traces

Since the number of sorting sequences is huge, Bergeron *et. al.* [3] proposed that for a given signed permutation π, the set of all sorting sequences can be classified into equivalence classes. They adapted the concept of *traces* to sorting sequences. However the authors did not provide an algorithm to enumerate the classes without enumerating all the sequences.

Two reversals are said to *overlap* if they intersect but neither is contained in the other. For example, in the permutation $\pi = (2, 4, 1, -3)$, the reversals $\{2,4\}$ and $\{1,3,4\}$ overlap, while $\{2,4\}$ and $\{1,2,4\}$ do not. If we identify a sequence of reversals with a word on the alphabet of reversals, an equivalence relation on these words can be established and forms classes, or "traces", of solutions. A *trace* is an equivalence class of sorting sequences of reversals, where the equivalence relation is defined as follows: if ρ and θ are reversals and do not overlap, then the words $\rho\theta$ and $\theta\rho$ are equivalent. We say that ρ and θ *commute*. Under this relation, two sorting sequences are said to be equivalent if one can be obtained from the other by a sequence of commutations of non-overlapping reversals.

For the permutation $\pi = (2, 4, 1, -3)$, consider the solution given by the sequence of reversals $\{1,3,4\}\{4\}\{1,2,3\}\{2,3\}$. Here, $\{1,3,4\}$ and $\{4\}$ commute, so $\{1,3,4\}\{4\}$ $\{1,2,3\}\{2,3\}$ and $\{4\}\{1,3,4\}\{1,2,3\}\{2,3\}$ are equivalent. These two permutations, along with 6 others, form a trace. The concept of traces is well studied in combinatorics [9]. The number of subwords in a trace is its *height*, and the number of solutions a trace represents is its *size*.

A theorem by Cartier and Foata [8] states that, for any trace, there is a unique word that is in normal form.

Definition 1 (normal form). *A trace T is in* normal form *if it can be decomposed into subwords $s = u_1 \mid \ldots \mid u_m$ such that:*

- *every pair of elements of a subword u_i commute;*
- *for every element ρ of a subword $u_i (i > 1)$, there is at least one element θ of the subword u_{i-1} such that ρ and θ do not commute;*
- *every subword u_i is a nonempty increasing word under the lexicographic order.*

For example, the permutation $\pi = (2, 4, 1, -3)$ has two normal forms representing sorting sequences: $\{1,3,4\}\{4\} \mid \{1,2,3\}\{2,3\}$ and $\{1,4\}\{2\} \mid \{1,2\}\{3,4\}$. The two normal forms of the traces describes the entire set of 16 sequences in a compact way.

Braga *et al.* [4,6] developed an algorithm (BR) to generate normal forms of traces. BR is also basically the ASSR-FW as described in Section 3, where solutions are represented as normal forms of traces, *NFtraces*. Their idea was based on the fact that each $(i-1)$-NFtrace is a prefix for an i-NFtrace.

Again, as in the ASSR-FW, for each i-NFtrace, BR [4] applies Siepel's ASR algorithm to generate the next sorting reversals. These sorting reversals are then inserted into the normal form in the appropriate position to build a normal form of length $i+1$. The costly operation here is inserting the new normal form into the exponentially large set of $(i+1)$-NFtraces. The BR algorithm lists all sorting solutions as $d(\pi)$-NFtraces. Since we can count the number of times a particular normal form is inserted into the

set of $(i+1)$-NFtraces, we can count the number of sorting sequences leading to any normal form.

It was shown empirically that the BR algorithm had a great impact on the speed of enumerating traces. The savings come from the fact that the number of i-NFtraces at each i-level in BR is much smaller than the number of i-sequences at the same i-level in SE. Accordingly, the number of times the two steps in ASSR-FW are repeated is much smaller in BR than that in SE. The complexity of BR depends on the total number of i-NFtraces at each i-level. Since each i-NFtrace is a prefix of a d-NFtrace, this number is bounded by the number of d-NFtraces times the number of prefixes of each i-NFtrace. The complete analysis was done in [6], where the i-NFtraces were represented as posets, and where it was proved that the complexity of BR is bounded by $O(Nn^{k_{max}+4})$, where N is the number of d-NFtraces and k_{max} is the maximum width of any normal form.

BR is a general algorithm in that many constraints can be applied to the sorting sequences it considers [5]. Unfortunately, the BR algorithm needs a potentially exponential amount of memory or extensive disk use, making it feasible only for distances up to 13 [4].

4.3 Generating Solutions Using Normal Forms of Traces with Appended Sorting Reversals

More recent work by Baudet *et al.* [2] generates normal forms of traces in a more precise way. As with BR, their algorithm (BD) follows the ASSR-FW representing i-solutions as i-NFtraces. To improve memory usage, they made use of the fact that not all sorting reversals calculated by Siepel's ASR algorithm are needed for every i-NFtrace.

Let an *appended sorting reversal* to a NFtrace be a sorting reversal that, when added to the normal form, will be the last reversal in the NFtrace. For example, for $\pi = (2, 4, 1, -3)$ with NFtraces $\{1,3,4\}\{4\} \mid \{1,2,3\}\{2,3\}$ and $\{1,4\}\{2\} \mid \{1,2\}\{3,4\}$. The set of 1-NFtraces is $\{\{1,3,4\},\{4\},\{1,4\},\{2\}\}$. For the 1-NFtrace $\{1,3,4\}$ the sorting reversal $\{4\}$ is an appended sorting reversal, while for the 1-NFtrace $\{4\}$ the sorting reversal $\{1,3,4\}$ is a non-appended sorting reversal.

Baudet et al. changed BR in two ways. First, they add a sorting reversal to an i-NFtrace only if that reversal is an appended sorting reversal. Second, they generated i-NFtraces in a depth-first manner. As a result of the first improvement, every $(i+1)$-NFtrace added to the $(i+1)$-level will be unique; each $(i+1)$-NFtrace can be visited in exactly one way by appended reversals. This decreases the time needed since there is no longer a search for duplicate $(i+1)$-NFtraces in the $(i+1)$-level. As a result of the second improvement, the algorithm requires only $O(n^3)$ space, whereas BR keeps all i-NFtraces. They used a stack of stacks to process these prefixes in a depth-first manner.

We now examine the connection between BR and BD in more detail.

Remark 1. The number of i-NFtraces, $1 \leq i \leq d$, that BR and BD visit are the same.

This can be seen by again noting that for every i-NFtrace, there is exactly one series of appended sorting reversals that will create it. Remark 1 highlights the fact that Siepel's ASR algorithm is called the same number of times by BR and BD so that the only difference is that BD will generate a given i-NFtrace exactly once, whereas BD could

generate the same i-NFtrace an exponential (in the size of the permutation) number of times.

BD also lends itself to a simple worst-case analysis. This is the first analysis for the problem of ASSR to give a bound on the time required to list each trace.

Theorem 1. *For a permutation of length n at distance d from the identity, the time that it takes BD to list a normal form is* $O(n^4 2^n)$.

Proof. Call an i-NFtrace that has no appended sorting reversals (in the set computed by Siepel's ASR algorithm) a *dead-end*. We show that for each $d-NFtrace$ BD finds, there are $O(2^n)$ dead-ends. Assume a $d-NFtrace\ T$ has only one subword (i.e. all sorting reversals commute), then for every subset of i ($1 \leq i \leq d$) reversals from T, there exists a dead-end $i-NFtrace$. So the number of dead-ends corresponding to T is the same as the number of substrings of a string, 2^d. Now take a $d-NFtrace\ S$ with more than one subword. The number of substrings corresponding to S is fewer than the number of substrings corresponding to T since the existence of more than one subword indicates that at least two reversals do not commute. This, along with the fact that each dead-end visited costs $O(n^4)$, gives us the desired bound. □

An issue with the BD algorithm is that it cannot count the total number of sorting sequences corresponding to the generated normal forms (counting the number of linear extensions on a poset is #P-complete [10]). Also, it cannot work when certain constraints are put on the sorting sequences; since it directly enumerates the normal forms of the traces, it cannot find traces that have compliant sorting sequences when the normal form itself does not comply with a particular constraint.

5 Permutation Grouping: An Improved Framework for ASSR

In the ASSR-FW described in Section 3, Siepel's ASR algorithm is called for every i-solution at each i-level. In this section, we propose a new framework for the ASSR problem (ASSR-PG-FW), where i-solutions are grouped at each i-level according to the corresponding permutations that they generate. The proposed ASSR-PG-FW ensures that the ASR algorithm is called exactly once for each permutation that is generated throughout the course of the algorithm. The theorem below motivates this new approach.

Lemma 1. *Consider the permutations along a sorting sequence* $\pi_0, \pi_1, \ldots, \pi_d$ *from* π_0 *to* π_d. *The number of i-NFtraces that when applied to* π_0 *yields* π_i *is monotonically increasing with i.*

Proof. Call T the set of i-NFtraces that yield π_i when applied to π_0. There is exactly one reversal ρ such that $\pi_i \circ \rho = \pi_{i+1}$. By inserting ρ into each of the normal forms for T we get a *unique* $(i+1)$-NFtrace because each pair of normal forms in T differ by at least one reversal and we add the same element to all of them. □

Theorem 2. *The ratio of the number of i-NFtraces to the number of permutations at an i-level is monotonically increasing with i.*

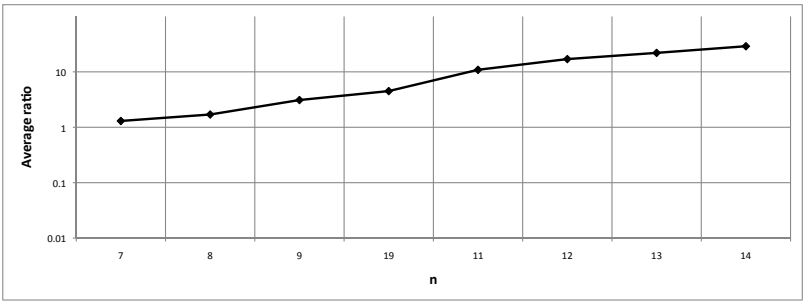


Fig. 1. The average ratio between the number of normal forms of traces to the number of permutations visited during a run of BR, for permutations chosen uniformly at random

Proof. The proof follows directly from Lemma 1.

Figure 1 shows the average ratio (over 20 runs) of the number of normal forms of traces to the number of permutations for $7 \leq n \leq 14$.

5.1 The ASSR-PG-FW

Let us first define the following notation:

- i-plevel: all permutations that are generated at distance $d(\pi) - i$ by any i-solution.
- i-p: a permutation in the i-plevel.
- i-S: the set of optimal sorting reversals obtained by applying Siepel's ASR algorithm to an i-p.
- i-pgroup: a group of i-solutions, where each i-solution in that group generates the same permutation i-p. Each i-p has one group.

The new framework, ASSR-PG-FW, can be modeled as follows:
For each i-p with corresponding i-pgroup, generate a set of $(i+1)$-solutions in the following two steps:

- Step 1: generate *i-S* for the i-p.
- Step 2: for each reversal ρ in *i-S*:
  - insert ρ in each i-solution in its corresponding i-pgroup to generate a set of $(i+1)$-solutions.
  - generate $(i+1)$-p = i-p $\circ$ ρ. Then we have the following two cases:

  Case 1 $(i+1)$-p is a new permutation: create a new group $(i+1)$-pgroup using the generated set of $(i+1)$-solutions.

  Case 2 $(i+1)$-p has already been generated: append the generated set of $(i+1)$-solutions to the corresponding $(i+1)$-pgroup.

These two steps are repeated iteratively until all optimal sorting $d(\pi)$-solutions are in the $d(\pi)$-pgroup and the $d(\pi)$-plevel will have only the identity permutation.

5.2 ASSR-PG-FW Algorithms

In order to process permutations, we use a hash table to store the permutation groups that can be generated at each *i*-plevel. The key for the hash table is the permutation *i*-p, and the data is the corresponding *i*-pgroup. This will help to test whether a given permutation has already been generated and to easily add solutions to their corresponding groups. Variations of the ASSR-PG approaches will again depend on how sorting solutions are represented. We have the following algorithms:

- *Solutions are represented as sorting sequences* (SE-PG). The SE algorithm [12] can be updated so it conforms to the proposed framework, where solutions are represented as sequences. Additional savings will come from the fact that the ASR algorithm will be applied once per permutation rather than once per *i*-sequence generated.
- *Solutions are represented as normal forms of traces, where normal forms are generated using all sorting reversals* (BR-PG). Again, the BR algorithm [4] can be easily updated so that it conforms to the proposed framework, where solutions are represented as normal forms of traces. When using normal forms of traces, Braga *et al.* has to sort each *i*-pgroup to remove any repetitions. Since the number of NFtraces per group in BR-PG is much smaller than the number of NFtraces per level in BR, savings will be also achieved in sorting the NFtraces per group versus per level. Savings will come from the fact that Siepel's ASR algorithm will be applied once per permutation rather than once per *i*-NFtrace.
- *Solutions are represented as normal forms of traces, where normal forms are generated with appended sorting reversals* (BD-PG). The BD algorithm [2] is the best known algorithm for solving the ASSR problem. However, the BD algorithm still calls the ASR algorithm more than once per permutation generated during the sorting process. We can update BD to optimize these calls and at the same time represent solutions as normal forms of traces that have been generated by appended sorting reversals as described in Section 4.3. However, the BD-PG algorithm will process the normal forms of traces in a breadth-first manner, grouping together those that generate the same permutation at each level.

To analyze the time complexity of these algorithms with high precision, the ratio of the number of solutions to the number of permutations at each level would have to be known. This will vary depending on the permutation so an average-case analysis may be the best we can hope for. We do not attempt such an analysis here. However, empirical results will be provided in Section 6 showing the savings that can be obtained when applying the new ASSR-PG-FW for each of the three approaches.

6 Empirical Results

Extensive experiments were done to compare the original ASSR-FW to the enhanced ASSR-PG-FW. Starting from the original Java source code for Braga *et al.* (baobabLUNA), we implemented the new techniques with the same Java objects. All the tests

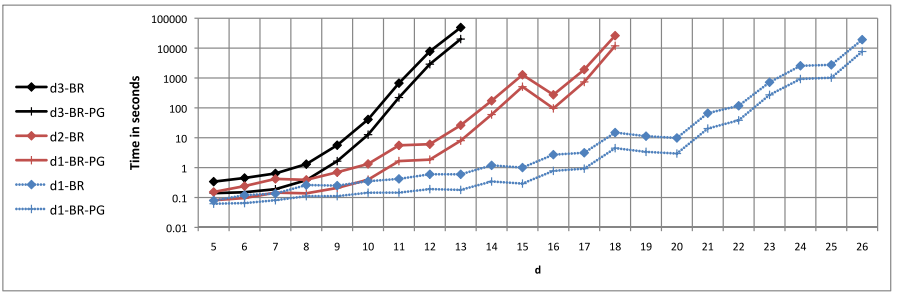


Fig. 2. The time in seconds comparing BR and BR-PG, where $d_1 = \lceil (n+1)/2 \rceil$, $d_2 = \lceil 3n/4 \rceil$, and $d_3 = n$

were performed using a single core of a 2.2GHz Quad-Core AMD Opteron(tm) Processor 8354 with 512 KB of cache and 132G of memory.

We used the same experimental setup as Baudet *et al.* [2]. We generated random permutations with size range between $5 \leq n \leq 26$. For each value of n, we considered permutations with reversal distances $d_1 = \lceil (n+1)/2 \rceil$, $d_2 = \lceil 3n/4 \rceil$, and $d_3 = n$. Since hurdles have a small probability of occurrence in random permutations [15], Braga *et al.* and Baudet *et al.* considered only the permutations that have no hurdles. We did the same.

To show the advantages of the proposed ASSR-PG-FW, for each set of permutations with parameters (n,d), we compared each approach when the original ASSR-FW is applied to the same approach when the ASSP-PG-FW is applied. In other words, we conduct three sets of experiments, where set 1 compares SE with SE-PG, set 2 compares BR with BR-PG, and set 3 compares BD with BD-PG.

The proposed ASSR-PG-FW shows time improvements for all three base methods. Figure 2, shows the results comparing BR and BR-PG. Note the logarithmic scale. Improvements in time performance is up to 70%. Figure 3 shows time performance in seconds comparing BD and BD-PG, where applying the new framework decreases

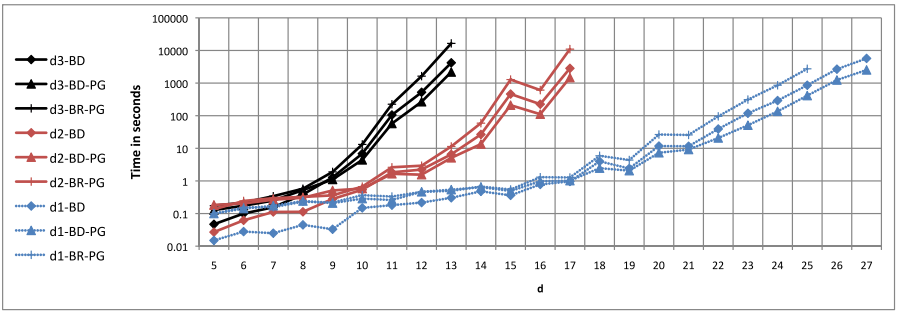


Fig. 3. The time in seconds comparing BR-PG, BD, and BD-PG, where $d_1 = \lceil (n+1)/2 \rceil$, $d_2 = \lceil 3n/4 \rceil$, and $d_3 = n$

BD time by up to 50%. The results shows that BR-PG cannot beat BD or BD-PG. The number of paths explored in the latter algorithms is much smaller, because only normal forms with reversals added at the end are explored. Results for comparing SE and SE-PG are not shown, but our experiments show that applying the new ASSR-PG-FW greatly decreases the time required by SE.

7 Conclusions

We revisited the problem of finding All Sorting Sequences by Reversals (ASSR). We showed that all approaches for solving the problem can be situated in the same fundamental framework, the ASSR-FW. This allowed us to compare all current approaches. We also proposed an enhanced framework, the ASSR-PG-FW, for solving the same problem by grouping partial solutions according to the corresponding permutations that they generate. The ASSR-PG-FW ensures that the All Sorting Reversals (ASR) algorithm is called exactly once for each permutation that is generated throughout the course of the algorithms. Extensive experiments were done that empirically demonstrate the speed-up that can be achieved by applying the ASSR-PG-FW. The results showed that by applying the new framework, we achieved an algorithm that beats the fastest known algorithm to solve the ASSR problem.

Acknowledgments

The authors would like to thank M.D.V. Braga for providing the Java source for the baobabLUNA software [4]. Research partially funded by an NSERC postdoctoral fellowship to GB and an NSERC Discovery Grant to DS.

References

1. Ajana, Y., Lefebvre, J.-F., Tillier, E.R.M., El-Mabrouk, N.: Exploring the set of all minimal sequences of reversals - an application to test the replication-directed reversal hypothesis. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 300–315. Springer, Heidelberg (2002)
2. Baudet, C., Dias, Z.: An improved algorithm to enumerate all traces that sort a signed permutation by reversals. In: Proc. the 2010 ACM Symposium on Applied Computing, pp. 1521–1525 (2010)
3. Bergeron, A., Chauve, C., Hartman, T., Saint-Onge, K.: On the properties of sequences of reversals that sort a signed permutation. In: JOBIM, pp. 99–108 (June 2002)
4. Braga, M.D.V.: Baobabluna: the solution space of sorting by reversals. Bioinformatics 25(14) (2009)
5. Braga, M.D.V., Gautier, C., Sagot, M.: An asymmetric approach to preserve common intervals while sorting by reversals. Algorithms for Molecular Biology 4(16) (2009)
6. Braga, M.D.V., Sagot, M., Scornavacca, C., Tannier, E.: The solution space of sorting by reversals. In: Măndoiu, I.I., Zelikovsky, A. (eds.) ISBRA 2007. LNCS (LNBI), vol. 4463, pp. 293–304. Springer, Heidelberg (2007)

7. Caprara, A.: Sorting by reversals is difficult. In: Proc. 1st Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB 1997), pp. 75–83. ACM Press, New York (1997)
8. Cartier, P., Foata, D.: Problèmes combinatoires de commutation et réarrangements. Lecture Notes in Maths, vol. 85 (1969)
9. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific, Singapore (1995)
10. Brightwell, G., Winkler, P.: Counting linear extensions is #P-complete. In: Proc. of the twenty-third annual ACM symposium on Theory of Computing, New Orleans, Louisiana, United States, pp. 175–181 (1991)
11. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In: Proc. 27th Ann. ACM Symp. Theory of Comput. (STOC 1995), pp. 178–189. ACM Press, New York (1995)
12. Siepel, A.C.: An algorithm to find all sorting reversals. In: Proc. 6th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB 2002), pp. 281–290. ACM Press, New York (2002)
13. Swenson, K.M., Rajan, V., Lin, Y., Moret, B.M.E.: Sorting signed permutations by inversions in $O(n \log n)$ time. In: Batzoglou, S. (ed.) RECOMB 2009. LNCS, vol. 5541, pp. 386–399. Springer, Heidelberg (2009)
14. Swenson, K.M., Badr, G., Sankoff, D.: Listing all sorting reversals in quadratic time. In: Singh, M. (ed.) WABI 2010. LNCS, vol. 6293, pp. 102–110. Springer, Heidelberg (2010)
15. Swenson, K.M., Lin, Y., Rajan, V., Moret, B.M.E.: Hurdles hardly have to be heeded. In: Nelson, C.E., Vialette, S. (eds.) RECOMB-CG 2008. LNCS (LNBI), vol. 5267, pp. 239–249. Springer, Heidelberg (2008)
16. Tannier, E., Bergeron, A., Sagot, M.-F.: Advances on sorting by reversals. Disc. Appl. Math. 155(6-7), 881–888 (2007)

Ultra-Perfect Sorting Scenarios

Aïda Ouangraoua[1], Anne Bergeron[2], and Krister M. Swenson[2,3]

[1] INRIA LNE, LIFL, Université Lille 1, Villeneuve d'Ascq, France
[2] Lacim, Université du Québec à Montréal, Montréal, Canada
[3] Department of Mathematics and Statistics, University of Ottawa, Canada

Abstract. Perfection has been used as a criteria to select rearrangement scenarios since 2004. However, there is a fundamental bias towards extant species in the original definition: ancestral species are not bound to perfection. Here we develop a new theory of perfection that takes an egalitarian view of species, and apply it to the complex evolution of mammal chromosome X.

1 Introduction

In mathematical biology, the genome sorting problem is to find a sequence of rearrangement operations that transforms one genome into another. The type of rearrangement operations is fixed, and a sorting sequence of operations is called a *scenario*. Given two genomes, there can be an exponential number of scenarios, which makes difficult the choice of one particular scenario, even among those of minimum length. Parsimony is only one of the criteria that can be used for the selection of a scenario, and there are many alternatives that are worth exploring.

Here we consider the problem of *perfect sorting* which was initially stated roughly as follows: given two genomes, find a sorting scenario between the genomes that preserves common genomic segments in intermediate states of the transformation. Such scenarios are called *perfect* and this problem was first introduced under the inversion rearrangement model by [7] who showed the NP-hardness of the problem. It was later shown that for some classes of instances, the problem could be solved in polynomial time [1,2,6,12]. More recently [3], the problem was explored under the double-cut-and-join (DCJ) rearrangement model, using a less stringent definition of perfection that allows temporary circular chromosomes.

In this paper, we address the problem of perfect sorting by DCJ under the initial definition of perfection. We also reexamine the original idea of perfection which applies only to the two compared extant species. What about all intermediate species that are generated by the scenario? Any good biological argument in favor of perfect scenarios would apply to any pair of these intermediate species. In this paper, we intend to correct this injustice.

We introduce a new, more restrictive class of perfection, called *ultra-perfection* with the corresponding problem: given two genomes, find a sorting scenario such that any sub-scenario is perfect. Our main results are the description of combinatorial properties of ultra-perfection that leads to a polynomial time algorithm

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 50–61, 2010.

for computing ultra-perfect scenarios between genomes, and the construction of a *near ultra-perfect* scenario between human, mouse and rat chromosome X.

The paper is organized as follows: in Section 2, we give the definitions of genomes, common intervals, rearrangement scenarios and ultra-perfection. In Section 3, we characterize ultra-perfection in terms of commutation of inversion scenarios, which leads to an algorithm for computing ultra-perfect scenarios. In section 4, we study the ultra-perfection of scenarios between the human and rodent chromosomes X. We conclude in Section 5 with remarks on the scenario described for human and rodents chromosome X, and on the algorithmic complexity of finding ultra-perfect or near ultra-perfect scenarios between several species.

2 Models and Definitions

In this section, we give the main definitions and notation that are used in the paper: genomes, inversions, double-cut-and-join operations, commuting inversions, perfect and ultra-perfect scenarios.

2.1 Genomes

Genomes are compared by identifying homologous segments along their DNA sequences, called *blocks*, organized in circular or linear chromosomes. A genome is *circular* (resp. *linear*) if it is only composed of circular (resp. linear) chromosomes. Each genome contains exactly one occurence of each block, and the order and orientation of the blocks may differ between genomes. A linear chromosome will be represented by an ordered sequence of signed integers, one for each block, flanked by the unsigned block $\circ$ at each end, and a circular chromosome will be represented by a circularly ordered sequence of signed integers. For example, genome $(-5\ \ 7\ \ 1\ -3\ \ 2)\ (\circ\ -6\ \ 4\ \ 8\ \ 9\ \circ)$ consists of one circular chromosome and one linear chromosome.

An *adjacency* in a genome is a pair of consecutive blocks. Since a chromosome can be read in two directions; the adjacencies $(x\ y)$ and $(-y\ -x)$ are equivalent. Moreover, since the block $\circ$ is unsigned, the adjacencies $(\circ\ y)$ and $(-y\ \circ)$ are equivalent. An *interval* in a genome is a set of blocks that appear consecutively in the genome. A *common interval* between genomes A and B is an interval that exists in both A and B. A *maximal common interval* between A and B is a common interval that is not included in any other common interval between A and B.

2.2 Rearrangement Scenarios

In this paper, we consider two models of rearrangements: the inversion model and the double-cut-and-join model. An *inversion* of a set of contiguous blocks reverses the order of those blocks and change their signs. A *double-cut-and-join* (DCJ) operation on a genome A cuts two different adjacencies in A and glues

pairs of the four exposed extremities to form two new adjacencies, no other adjacency is altered. The *circularization* of a linear chromosome $(\circ\ x \dots y\ \circ)$ is the DCJ operation that cuts adjacencies $(\circ\ x)$ and $(y\ \circ)$ to produce $(y\ x)$ and $(\circ\ \circ)$, thus creating the circular chromosome $(x \dots y)$. The opposite operation called a *linearization* is a DCJ operation that transforms a circular chromosome into a linear chromosome.

For example, a DCJ operation on genome $(-5\ \ 7\ \ 1\ -3\ \ 2)\ (\circ\ -6\ \ 4\ \ 8\ \ 9\ \circ)$ that cuts the adjacencies $(2\ -5)$ and $(-6\ \ 4)$ to form $(2\ \ 4)$ and $(-6\ -5)$ would produce genome $(\circ\ -6\ -5\ \ 7\ \ 1\ -3\ \ 2\ \ 4\ \ 8\ \ 9\ \circ)$. Note that we consider the empty chromosome $(\circ\ \ \circ)$ to belong to any genome so that the DCJ on the circular genome $(-5\ \ 7\ \ 1\ -3\ \ 2)$ which cuts adjacencies $(7\ \ 1)$ and $(\circ\ \ \circ)$ will produce the linear genome $(\circ\ 1\ -3\ \ 2\ -5\ \ 7\ \circ)$.

Let A and B be two genomes. A *scenario* from A to B is a sequence of rearrangements that transforms A into B. An *inversion scenario* from A to B contains only inversions, and *DCJ scenario* contains only DCJ operations. Since an inversion can always be realized by one DCJ operation, an inversion scenario is always a DCJ scenario.

Note that the application of an inversion on a given genome only affects a single chromosome of the genome. Thus, an inversion scenario on a genome A can always be decomposed into a set of independent inversion scenarios, each acting on a single chromosome of genome A. In the following, we will restrict our study of inversion scenarios to unichromosomal genomes.

2.3 Perfection and Commutation

The following definition of perfection is used in [1,2,7,12]:

Definition 1 (Perfection). *A rearrangement scenario from genome A to genome B is* perfect *if all common intervals of A and B are also intervals in the intermediate genomes of the scenario.*

For the purposes of this paper we introduce a new, more restrictive class of perfection that we call *ultra-perfection.*

Definition 2 (Ultra-Perfection). *A rearrangement scenario from genome A to genome B is* ultra-perfect *if all sub-scenarios are perfect.*

The difference between the two notions is illustrated by the following example. Consider the two genomes $(\circ\ -3\ -1\ \ \ 4\ \ \ 2\ \circ)$ and $(\circ\ 1\ \ \ 2\ \ \ 3\ \ \ 4\ \circ)$ that have no common intervals, except trivial ones. Definition 1 implies that any inversion scenario between the two genomes is perfect. However, as we will show in Section 3, none of these scenario is ultra-perfect. In particular, the following scenario creates the common interval $\{2, 3, 4\}$ between chromosomes C_2 and C_4, but destroys it in C_3:

$$C_1 = (\circ\ -3\ \underline{-1\ \ \ 4\ \ \ 2}\ \circ)$$
$$C_2 = (\circ\ -3\ -2\ \underline{-4\ \ \ 1}\ \circ)$$
$$C_3 = (\circ\ \underline{-3\ -2\ -1}\ \ \ 4\ \circ)$$
$$C_4 = (\circ\ 1\ \ \ 2\ \ \ 3\ \ \ 4\ \circ)$$

As we will see, the notion of ultra-perfection is intimately related with commutation of inversions. We first recall the definition of commutation used in [1]: two sets of blocks commute if they are either disjoint or one is included in the other. Two inversions *commute* if their associated sets of blocks commute. An inversion scenario is *commuting* if all pairs of inversions contained in the scenario commute.

When considering circular chromosome, the definition of commutation must be adapted. Indeed, in a circular chromosome containing the set of blocks G, an inversion associated to a set $S \subset G$ produces the same result as an inversion associated to the set $G \setminus S$. We thus introduce the *circularly commuting* property defined as: given a set G of blocks, two subsets S and T of G *commute circularly* if S and T commute, or $G \setminus S$ and T commute.

Note that in [3] a less stringent definition of perfection is used for DCJ scenarios. In this version, the notion of common interval is relaxed to allow subsets of common intervals to form circular chromosomes.

3 Ultra-Perfect Scenarios

We now consider the problem of computing an ultra-perfect scenario between two genomes. In Section 3.1, we show that each maximal common interval can be considered independently. This implies an ultra-perfect scenario that first sorts each substring associated with a maximal common interval, and then sorts the whole genome in its final configuration. Section 3.2 then describes conditions for the existence of ultra-perfect DCJ scenarios between substrings associated to each maximal common interval.

Throughout the section we refer to a substring of a genome associated with an interval as a *segment* of the genome. The segment of a genome A induced by an interval I is denoted A_I.

3.1 Independent DCJ Sorting of Maximal Common Intervals

The following proposition states that, in any ultra-perfect DCJ scenario between genomes A and B, the segments induced by maximal common intervals of A and B are sorted independently.

Proposition 1. *Let I be a maximal common interval between genomes A and B. If $\mathcal{S}$ is an ultra-perfect DCJ scenario from A to B, then there exists an equal length ultra-perfect scenario $\mathcal{S}' = \mathcal{S}_I\mathcal{T}$ such that $\mathcal{S}_I$ is an ultra-perfect DCJ scenario transforming A_I into B_I.*

Proof. Let C be a genome obtained by applying some (possibly empty) prefix of $\mathcal{S}$, and D be the genome obtained by applying some subsequent operations of $\mathcal{S}$ on C. Since $\mathcal{S}$ is ultra-perfect, I is an interval of C and D. Say that the operation transforming the intermediate genome C into the intermediate genome D *modifies* I if $C_I \neq D_I$.

Let f be an operation of $\mathcal{S}$ such that f modifies I and the operation e preceding f does not modify I. If f cuts no adjacency created by e, then simply switch the order of e and f. Otherwise we replace e and f by two DCJs e' and f' such that e' precedes f', through the following process: say e cuts the adjacency $(u\ v)$ and $(y\ x)$, u being a block belonging to I, to create $(u\ x)$ and $(y\ v)$, and that f then cuts $(u\ x)$ and $(t\ s)$ to create $(u\ s)$ and $(t\ x)$. Then take DCJs e' and f' such that e' cuts the adjacency $(u\ v)$ and $(t\ s)$ to create $(u\ s)$ and $(t\ v)$, and f' cuts $(t\ v)$ and $(y\ x)$ to create $(t\ x)$ and $(y\ v)$. In this way, any operations modifying I can be moved to the beginning of the DCJ scenario. Each move does not effect the ultra-perfection of the scenario since e' cannot create an interval that is later broken: any new interval created by e' (and later broken by some DCJ g) would have to include some elements of I and some adjacent elements to I. But this implies the existence of a larger interval that would be broken by g in the original scenario.

Moreover, at the end of this process, the scenario obtained can be decomposed into two sequences $\mathcal{S}_I$ and $\mathcal{T}$ such that all operations in $\mathcal{S}_I$ modify I but no operation in $\mathcal{T}$ modifies I. Then, $\mathcal{S}_I$ is an ultra-perfect DCJ scenario between the segments A_I and B_I. □

3.2 Ultra-Perfect Scenarios between Unichromosomal Genomes

Proposition 1 implies that, for each maximal common interval I of A and B, the ultra-perfect sorting of A_I into B_I can be examined independently from the rest of the scenario. We now give characterizations of the existence of ultra-perfect DCJ scenarios between unichromosomal genomes A and B by establishing properties of commuting inversion scenarios.

Ultra-perfect inversion scenarios. First we characterize ultra-perfect inversion scenarios.

Proposition 2. *An inversion scenario between two linear genomes is ultra-perfect if and only if the scenario is commuting.*

Proof. [2] shows that a commuting inversion scenario is always perfect. So, if a scenario $\mathcal{S}$ is commuting, then all sub-scenarios of $\mathcal{S}$ are commuting and thus perfect. Commutation then implies ultra-perfection.

Next, let $\mathcal{S}$ be an ultra-perfect scenario. Suppose that $\mathcal{S}$ is not commuting. Then there exists two inversions in $\mathcal{S}$, e preceding f, such that e and f do not commute but e and f commute with all inversions in $\mathcal{S}$ between e and f.

Say, without loss of generality, that the genome looks like $UVWXY$ before applying inversion e where U, V, W, X, and Y represent sets of blocks. In this configuration $e = \{V, W\}$ and $f = \{V, X\}$, and all inversions in the scenario between e and f, then, do not change the relative order of V,W, and X to each other (since they commute with e and f). So e creates the order WVX while f creates the order WXV. But this contradicts the hypothesis that $\mathcal{S}$ is ultra-perfect since the interval $\{W, X\}$ is destroyed by e and recreated by f. □

In [2], it was shown that the commuting inversion scenarios between linear genomes could be characterized in terms of the structure of a tree, called the *strong interval tree*, representing the set of all common intervals. The *strong interval tree* of two linear genomes is a tree whose vertices are the common intervals that commute with any other common interval and there is an edge (I, J) between two vertices if $J \subset I$ and there exist no third vertex K such that $J \subset K \subset I$. The strong interval tree was first described in [8,11,4] and inspired from a data structure called the *PQ-tree*. PQ-trees are used to represent all consecutive-ones orderings of the columns of a matrix that has the consecutive-ones property.

In the remainder of the section we develop the counterpart characterization for DCJ scenarios by relating ultra-perfect DCJ scenarios to a tree representing the set of all common intervals between two circular genomes. We present the *circular common interval tree* that is the circular analogue of the strong interval tree. Circular common interval trees are inspired from an analogous structure of a PQ-tree, called a *PC-tree*. The PC-tree was introduced in [9,10] where it was used to represent all circular-ones orderings of the columns of a matrix that has the circular-ones property.

Circular common interval tree. Here, we define a tree representing the set of all common intervals between two circular genomes. Let G be a set of n blocks, and A and B be two circular genomes on G. A *circular common interval* of A and B is either a singleton block, or a subset I of G such that I is a common interval between A and B, and $|I| \neq n - 1$. The *circular strong intervals* of A and B are the circular common intervals of A and B that commute circularly with any other circular common interval.

The *circular common interval tree* of two circular genomes is then defined as follows:

Definition 3. *The* circular common interval tree *of two circular genomes A and B, denoted by $T(A, B)$ is defined as follows: the vertices of $T(A, B)$ are the circular strong intervals of A and B that commute with any other circular strong interval; a vertex J is a child of a vertex I if $J \subset I$, and there exist no third vertex K such that $J \subset K \subset I$.*

For example, let us consider the following circular genomes $A = (\ 2\ \ 4\ \ 3\ \ 1\ \ 5\ )$ and $B = (\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ )$. The circular common intervals of A and B are $\{1, 5\}$, $\{1, 5, 2\}$, $\{4, 3\}$, $\{2, 4, 3\}$, and the singletons $\{1\} \ldots \{5\}$. The circular strong intervals are all the circular common intervals. The vertices of the circular common interval tree $T(A, B)$ are $\{1, 5\}$, $\{4, 3\}$ and the singletons. $T(A, B)$ is depicted in Figure 1.a.

Given a vertex I of $T(A, B)$, two orderings, either both circular, or both linear, of the set of children of I can be inferred from the orderings in A and B of the set of blocks composing I. Following the notation of [2], the vertex I is *linear* if both orderings are identical or reciprocal, otherwise the vertex I is *prime*. If I is a linear vertex, I has a *positive sign* if both orderings are identical, and a *negative sign* if they are reciprocal (see Figure 1 for an example).

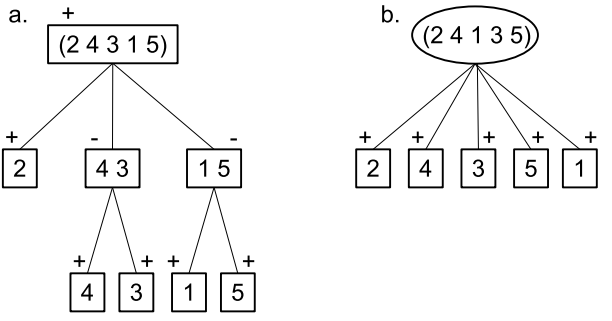


Fig. 1. The circular common interval trees of a. A and C, and b. B and C where genomes A, B, C are $A = (\ 2\ \ 4\ \ 3\ \ 1\ \ 5\ )$, $B = (\ 2\ \ 4\ \ 1\ \ 3\ \ 5\ )$, and $C = (\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ )$. Rectangular nodes correspond to linear vertices and round nodes to prime vertices. The signs of linear vertices are indicated by a $+$ or $-$ symbol.

A circular common interval tree is *definite* if its vertices are linear. For example, in Figure 1, the left-hand circular common interval tree is definite, while the right-hand one is not definite.

The definition of linear or prime vertices, and definite trees, also hold for the strong interval tree of two linear genomes. Given a vertex I of the strong interval tree of two linear genomes A and B, if I is linear (resp. prime), we also say that the common interval I between A and B is linear (resp. prime).

Ultra-perfect DCJ scenarios. In [2], it was shown that the existence of a perfect inversion scenario between unichromosomal linear genomes A and B was conditioned on properties of the strong interval tree of A and B: there exists a commuting scenario if and only if the strong interval tree is definite. In the following, we give the equivalent theorem for DCJ scenarios with circular common interval trees: there exists an ultra-perfect DCJ scenario if and only if the circular common interval tree is definite. We start by stating an obvious but useful property of ultra-perfect DCJ scenarios.

Property 1. Let A and B be unichromosomal genomes on the same set of blocks G. G is a common interval between A and B. So, if a DCJ scenario $\mathcal{S}$ from A to B is ultra-perfect, then any operation in $\mathcal{S}$ is either an inversion, or a circularization, or a linearization.

The *circular version* of a unichromosomal genome A is A itself if A is already a circular genome, otherwise it is the circular genome obtained by applying the circularization DCJ on A. We denote by A_c the circular version of a genome A.

Theorem 1. *Let A and B be two unichromosomal genomes with the same set of blocks. There exists an ultra-perfect DCJ scenario from A to B if and only if the circular common interval tree of A_c and B_c is definite.*

Proof. Let $T = T(A_c, B_c)$ be a circular common interval tree of A_c and B_c.

First, if T is definite, then an inversion scenario $\mathcal{S}$ composed of inversions whose associated sets are the vertices of T that have a sign different from their parents is a commuting inversion scenario from A_c to B_c. So, $\mathcal{S}$ is an ultra-perfect scenario. Say $e\ \mathcal{S}\ f$ is the DCJ scenario from A to B where e is the eventual circularization transforming A into A_c and f is the eventual linearization transforming B_c into B. It is easy to see that $e\ \mathcal{S}\ f$ is an ultra-perfect scenario from A to B.

Now, if T is not definite, let I be a vertex of T that is prime. Since a DCJ scenario from A to B contains only inversions, circularization and linearization, then there exists no ultra-perfect DCJ scenario that sorts A_I into B_I. □

For example, consider the linear genomes $A = (\circ\ 3\ \ 1\ \ 5\ \ 2\ \ 4\ \circ)$ and $B = (\circ\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \circ)$. The circular common interval tree of of the circular versions of A and B is depicted in Figure 1. Since it is a definite tree, then there exists the following ultra-perfect DCJ scenario:

$(\circ\ 3\ \ 1\ \ 5\ \ 2\ \ 4\ \circ)$	*circularization*	$(4\ -5\ -1\ 2\ \ 3)$	*inversion of* $\{1\}$
$(3\ \ 1\ \ 5\ \ 2\ \ 4)$	*inversion of* $\{4,3\}$	$(4\ -5\ \ 1\ \ 2\ \ 3)$	*inversion of* $\{5\}$
$(-4\ \ 1\ \ 5\ \ 2\ -3)$	*inversion of* $\{1,5\}$	$(4\ \ 5\ \ 1\ \ 2\ \ 3)$	*linearization*
$(-4\ -5\ -1\ 2\ -3)$	*inversion of* $\{4\}$	$(\circ\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \circ)$	
$(4\ -5\ -1\ 2\ -3)$	*inversion of* $\{3\}$		

4 Ultra-Perfection Meets Reality

The study of perfect rearrangement scenarios was, in large part, motivated by scenarios of inversions linking the human, mouse and rat chromosomes X [1]. In that first study, the presence of prime intervals arising from the comparison between human and rodents was largely ignored, since any inversion scenario that sorts a prime interval is perfect. In the framework of ultra-perfection, it makes sense to revisit this fundamental example.

In this section, we will show that, although no ultra-perfect scenario exists between human and rodents, there exist a scenario which is minimally disruptive. We will first describe a framework for dealing with imperfection. We will then show that it is possible to construct a unique scenario for the human, mouse and rat chromosome X that is the closest possible to ultra-perfection.

4.1 Imperfect Sorting

When no ultra-perfect scenario exists, we wish to define a way to score scenarios that are nearly ultra-perfect. There is no easy or straightforward way to do this: the competing parameters include broken common intervals, overlapping inversions, prime intervals and parsimony. In this section, we propose a first measure that is relatively simple to define, and that allows to compare scenarios that would otherwise be difficult to rank.

Our first simplification is that, when trying to build a scenario for two or more species, each common interval should be sorted independently. Since linear

common intervals are rather easy to sort with an ultra-perfect scenario, we focus on the sorting of individual prime common intervals.

A scenario $\mathcal{S}$ between two or more species can be represented as an unrooted tree whose nodes are the genomes, and whose branches are the rearrangement operations. *Removing* an operation r from a scenario $\mathcal{S}$ is done by cutting the branch labeled by r yielding two subtrees called *subscenarios*. We have:

Definition 4. *The* imperfection score *of a scenario $\mathcal{S}$ is the minimum number of operations that can be removed from $\mathcal{S}$ such that each of the remaining subscenarios is ultra-perfect.*

Our goal is to find, among all possible scenarios with minimum imperfection score, one that is of minimum length.

It turns out that the data on human, mouse, and rat chromosomes X is a very interesting instance of this problem: there are prime common intervals in both the Human-Mouse and the Human-Rat strong interval trees. The prime common interval in the Human-Mouse comparison is maximal, but there is no ultra-perfect scenario since the induced permutation [2], $(\circ\ -4\ \ 6\ \ 1\ -3\ -5\ \ 2\ \circ)$, does not have a commuting scenario, even with circularization.

The following permutations, obtained from the blocks of [5], model the homologous blocks of the human, mouse, and rat chromosomes X :

$$
\begin{aligned}
H &= (\circ\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 10\ \ 11\ \ 12\ \ 13\ \ 14\ \ 15\ \ 16\ \circ)\\
M &= (\circ\ -6\ -5\ \ 4\ \ 13\ \ 14\ -15\ \ 16\ \ 1\ -3\ \ 9\ -10\ \ 11\ \ 12\ -7\ \ 8\ -2\ \circ)\\
R &= (\circ\ -13\ -4\ \ 5\ -6\ -12\ -8\ -7\ \ 2\ \ 1\ -3\ \ 9\ \ 10\ \ 11\ \ 14\ -15\ \ 16\ \circ)
\end{aligned}
$$

We first apply single block inversions (they have no impact on the perfection of a scenario) that create an adjacency in one genome that already exists in the other two. There are 5 of them in the chromosome data: $\{4\}$ and $\{15\}$ applied to the human chromosome, $\{7\}$ and $\{10\}$ applied to the mouse chromosome, and $\{6\}$ applied to the rat chromosome. The resulting chromosomes are the following, renamed with a subscript that indicates how far the new chromosome is from the original.

$$
\begin{aligned}
H_{+2} &= (\circ\ 1\ \ 2\ \ 3\ -4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 10\ \ 11\ \ 12\ \ 13\ \ 14\ -15\ \ 16\ \circ)\\
M_{+2} &= (\circ\ -6\ -5\ \ 4\ \ 13\ \ 14\ -15\ \ 16\ \ 1\ -3\ \ 9\ \ 10\ \ 11\ \ 12\ \ 7\ \ 8\ -2\ \circ)\\
R_{+1} &= (\circ\ -13\ -4\ \ 5\ \ 6\ -12\ -8\ -7\ \ 2\ \ 1\ -3\ \ 9\ \ 10\ \ 11\ \ 14\ -15\ \ 16\ \circ)
\end{aligned}
$$

Next, there are three adjacencies that are shared by the rodents, but are not in the human lineage, and that require inversions longer than single blocks: $(1\ -3)$, $(4\ \ 13)$ and $(-3\ \ 9)$. The corresponding inversions associated to the sets, $\{2,3\}$, $\{4,5,6,7,8,9,10,11,12\}$ and $\{2,9,10,11,12\}$ can be applied to the H_{+2} genome to yield $H_{+5} = (\circ\ 1\ -3\ \ 9\ \ 10\ \ 11\ \ 12\ \ 2\ -8\ -7\ -6\ -5\ \ 4\ \ 13\ \ 14\ -15\ \ 16\ \circ)$.

In the next section, we will construct an ultra-perfect scenario for the three permutations H_{+5}, M_{+2} and R_{+1}. Since the first two inversions applied to the H_{+2} chromosome are commuting, the imperfection score of the global scenario will be 1, obtained by removing inversion $\{2,9,10,11,12\}$, which is the best that

can be achieved. The scenario between H_{+2} and H_{+5} is also parsimonious, since it constructs 3 adjacencies present in both the mouse and rat genome, implying that any alternate solution should have the same length. Up to commutation of the two initial inversions, it is easy to show that this is the only solution constructing the 3 adjacencies.

4.2 The Ultra-Perfect Median of Three Genomes

After applying the inversions of the preceding section, the three genomes are the following, where adjacencies common to all three chromosomes are indicated by dots:

$$H_{+5} = (\circ\ 1 \cdot -3 \cdot 9 \cdot 10 \cdot 11\ \ 12\ \ 2\ -8 \cdot -7\ -6 \cdot -5 \cdot 4 \cdot 13\ \ 14 \cdot -15 \cdot 16\ \circ)$$
$$M_{+2} = (\circ\ -6 \cdot -5 \cdot 4 \cdot 13\ \ 14 \cdot -15 \cdot 16\ \ 1 \cdot -3 \cdot 9 \cdot 10 \cdot 11\ \ 12\ \ 7 \cdot 8\ -2\ \circ)$$
$$R_{+1} = (\circ\ -13 \cdot -4 \cdot 5 \cdot 6\ -12\ -8 \cdot -7\ \ 2\ \ 1 \cdot -3 \cdot 9 \cdot 10 \cdot 11\ \ 14 \cdot -15 \cdot 16\ \circ)$$

It is convenient, at this point, to relabel the blocks so that the remaining differences are more apparent. There are six blocks that we label with respect to the H_{+5} genome order:

$$\overbrace{1 \cdot -3 \cdot 9 \cdot 10 \cdot 11}^{\mathbf{1}}\ \overbrace{12}^{\mathbf{2}}\ \overbrace{2}^{\mathbf{3}}\ \overbrace{-8 \cdot -7}^{\mathbf{4}}\ \overbrace{-6 \cdot -5 \cdot 4 \cdot 13}^{\mathbf{5}}\ \overbrace{14 \cdot -15 \cdot 16}^{\mathbf{6}}$$

This yields the new representation:

$$H_{+5} = (\circ\ \mathbf{1}\ \ \mathbf{2}\ \ \mathbf{3}\ \mathbf{4}\ \ \mathbf{5}\ \ \mathbf{6}\ \circ)$$
$$M_{+2} = (\circ\ \mathbf{5}\ \ \mathbf{6}\ \ \mathbf{1}\ \ \mathbf{2}\ -\mathbf{4}\ -\mathbf{3}\ \circ)$$
$$R_{+1} = (\circ\ -\mathbf{5}\ -\mathbf{2}\ \ \mathbf{4}\ \ \mathbf{3}\ \ \mathbf{1}\ \ \mathbf{6}\ \circ)$$

Given three genomes, deciding if an ultra-perfect scenario connecting them exists begins with a simple check. Indeed, the median of three genomes belongs to all implied pairwise scenarios, thus must share all common intervals of all pairs of genomes. Formally we have:

Proposition 3. *The median M of an ultra-perfect scenario linking three permutations A, B and C contains all common intervals of A and B, of A and C, and of B and C.*

In order to apply Proposition 3 to the mammal chromosomes, we first compute their common intervals:

H_{+5} and M_{+2}: $\{\mathbf{1},\mathbf{2}\}, \{\mathbf{3},\mathbf{4}\}, \{\mathbf{5},\mathbf{6}\}, \{\mathbf{2},\mathbf{3},\mathbf{4}\}, \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4}\}$
H_{+5} and R_{+1}: $\{\mathbf{3},\mathbf{4}\}, \{\mathbf{2},\mathbf{3},\mathbf{4}\}, \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4}\}, \{\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5}\}, \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5}\}$
M_{+2} and R_{+1}: $\{\mathbf{1},\mathbf{6}\}, \{\mathbf{2},\mathbf{4}\}, \{\mathbf{3},\mathbf{4}\}, \{\mathbf{2},\mathbf{3},\mathbf{4}\}, \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4}\}, \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{6}\}$

The – unique – permutation that contains all these intervals is a circular chromosome! Its block order, (**5 6 1 2 −4 −3**), is the circularization of the mouse genome. It is then a simple exercise to transform the median into each genome. The whole scenario has 7 inversions, 5 of them inverting single blocks; the remaining 2 inversions are $\{3,4\}$ towards the human chromosome, and $\{2,3,4\}$ towards the rat chromosome, see Figure 2.

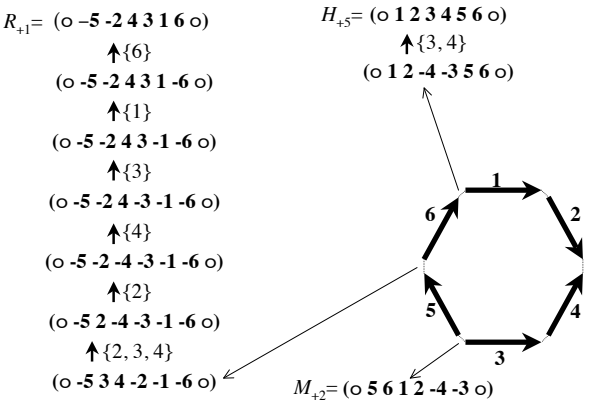


Fig. 2. An ultra-perfect scenario between chromosomes H_{+5}, M_{+2} and R_{+1}. The location where the circular median is cut is shown by thin arrows. The set of inverted blocks is shown between each pair of permutations.

5 Discussion and Conclusions

The search for an ultra-perfect scenario for the human, mouse and rat chromosome X lead to a surprising circular median, deduced by combinatorial techniques. We are certainly not inferring that actual species had circular chromosomes X. The fact that the number of blocks is quite small, $n = 6$, might be the simplest explanation: more than half of the random trios of permutations on six elements have a circular median. However, the remarkable preservation of the circular order of blocks between the human and mouse chromosome X asks for a more satisfying answer. Are there some biological mechanisms that would allow rearrangement operations that preserve a circular order? Among the well known combinatorial operations with this property are the *shift* operation, or a double centromeric inversion.

On the algorithmic side, the circular common interval tree allows us to easily find the set of inversions of an ultra-perfect scenario between two genomes; ultra-perfect DCJ scenarios are essentially ultra-perfect inversion scenarios on a circular version of the genomes. In the case of multiple genomes, the existence and computation of an ultra-perfect scenario should be easy to characterize using the sets of inversions corresponding to pairwise genomes. In the case of nearly ultra-perfect scenario, methods still have to be developed but most efforts will likely lead to hardness results. However, interesting instances of the problem, such as rearrangements between human and rodents, are still quite manageable by manual techniques, and should get easier with the sequencing of additional rodent genomes. It is also relatively easy to score scenarios: the scenario proposed by GRIMM [13] for the human, mouse and rat chromosome X has an imperfection score of 2.

Finally we argue that perfection without ultra-perfection remains, at best, a mathematical exercise. On the other hand, reality is neither perfect, nor ultra-perfect. In this paper we explored some definitions of near ultra-perfection, fully aware that much remains to be done.

References

1. Bérard, S., Bergeron, A., Chauve, C.: Conservation of combinatorial structures in evolution scenarios. In: Lagergren, J. (ed.) RECOMB-WS 2004. LNCS (LNBI), vol. 3388, pp. 1–14. Springer, Heidelberg (2005)
2. Bérard, S., Bergeron, A., Chauve, C., Paul, C.: Perfect sorting by reversals is not always difficult. IEEE/ACM Trans. Comput. Biology Bioinform. 4(1), 4–16 (2007)
3. Bérard, S., Chateau, A., Chauve, C., Paul, C., Tannier, E.: Perfect DCJ rearrangement. In: Nelson, C.E., Vialette, S. (eds.) RECOMB-CG 2008. LNCS (LNBI), vol. 5267, pp. 158–169. Springer, Heidelberg (2008)
4. Bergeron, A., Chauve, C., de Montgolfier, F., Raffinot, M.: Computing Common Intervals of k Permutations, with Applications to Modular Decomposition of Graphs. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 779–790. Springer, Heidelberg (2005)
5. Bourque, G., Pevzner, P.A., Tesler, G.: Reconstructing the genomic architecture of ancestral mammals: Lessons from human, mouse, and rat genomes. Genome Research 14(4), 507–516 (2004)
6. Braga, M.D., Gautier, C., Sagot, M.-F.: An asymmetric approach to preserve common intervals while sorting by reversals. Algorithms for Molecular Biology 4(16) (2009)
7. Figeac, M., Varré, J.-S.: Sorting by reversals with common intervals. In: Jonassen, I., Kim, J. (eds.) WABI 2004. LNCS (LNBI), vol. 3240, pp. 26–37. Springer, Heidelberg (2004)
8. Heber, S., Stoye, J.: Finding all common intervals of k permutations. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 207–218. Springer, Heidelberg (2001)
9. Hsu, W.-L.: PC-trees vs. PQ-trees. In: Wang, J. (ed.) COCOON 2001. LNCS, vol. 2108, pp. 207–217. Springer, Heidelberg (2001)
10. Hsu, W.-L., McConnell, R.M.: PC trees and circular-ones arrangements. Theor. Comput. Sci. 296(1), 99–116 (2003)
11. Landau, G.M., Parida, L., Weimann, O.: Using PQ trees for comparative genomics. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 128–143. Springer, Heidelberg (2005)
12. Sagot, M.-F., Tannier, E.: Perfect sorting by reversals. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 42–51. Springer, Heidelberg (2005)
13. Tesler, G.: GRIMM: genome rearrangements web server. Bioinformatics 18(3), 492–493 (2002)

On Sorting Genomes with DCJ and Indels

Marília D.V. Braga

Technische Fakultät, Universität Bielefeld, Germany
mbraga@cebitec.uni-bielefeld.de

Abstract. A previous work of Braga, Willing and Stoye compared two genomes with unequal content, but without duplications, and presented a new linear time algorithm to compute the genomic distance, considering double cut and join (DCJ) operations, insertions and deletions. Here we derive from this approach an algorithm to sort one genome into another one also using DCJ, insertions and deletions. The optimal sorting scenarios can have different compositions and we compare two types of sorting scenarios: one that maximizes and one that minimizes the number of DCJ operations with respect to the number of insertions and deletions.

1 Introduction

The approach of sorting a genome by double cut and join (DCJ) operations, introduced by Yancopoulos *et al.* in 2005 [10], has been the topic of many studies in the latest years [3,7,8]. In particular, linear time algorithms have been proposed to compute the DCJ distance and to find an optimal DCJ sorting sequence [1].

A DCJ allows us to represent most large scale mutation events, such as inversions, translocations, fusions and fissions, that can occur in genomes. A related approach is the one that considers only inversions in unichromosomal genomes [6]. Since a DCJ or an inversion cannot perform an insertion or a deletion, most of the studies under these models consider genomes with the same content.

In 2001, El-Mabrouk [5] introduced an approach to compare unichromosomal genomes with unequal content, considering inversions, insertions and deletions, such that a block of contiguous markers can be inserted (or deleted) at once. Recently Braga *et al.* [4] studied a similar problem and proposed a linear time algorithm to compute the genomic distance with DCJ, insertions and deletions between genomes with unequal content, but without duplications. Braga *et al.* [4] borrowed some ideas from a study of Yancopoulos and Friedberg [9], but allowed a block of contiguous markers to be inserted (or deleted) at once, as well as the related approach of El-Mabrouk [5].

Here we derive from the approach of Braga *et al.* [4] an algorithm to sort one genome into another one with DCJ, insertions and deletions. We show that the optimal sorting scenarios can have different compositions with respect to the number of each type of operation and propose two types of sorting scenarios: one that minimizes the number of DCJ operations (respectively maximizing the number of insertions and deletions) and one that minimizes the number of insertions and deletions (respectively maximizing the number of DCJ operations).

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 62–73, 2010.

2 DCJ-Indel Distance

In this section we summarize the results given in [4], that allow us to compute the genomic distance considering DCJ operations, insertions and deletions.

We analyze genomes with unequal gene content but without duplications. Given two genomes A and B, denote by $\mathcal{G}$ the "reduced" genome [5], that is the set of markers that occur once in A and once in B. Moreover, the set $\mathcal{A}$ contains the markers that occur only in A and the set $\mathcal{B}$ contains the markers that occur only in B. Observe that the sets $\mathcal{G}$, $\mathcal{A}$ and $\mathcal{B}$ are disjoint. Each genome is possibly composed of linear and circular chromosomes and can be represented by a set of strings as follows. From each chromosome $\mathcal{C}$ of each genome, we can build a string s, obtained by the concatenation of all markers in $\mathcal{C}$, read in any of the two directions. Each marker g is a DNA fragment and is represented by the symbol g, if it is read in direct orientation, or by the symbol $\overline{g}$, if it is read in reverse orientation. Each end of a linear chromosome is called a *telomere*, represented by the symbol $\circ$. Thus, if $\mathcal{C}$ is linear, it is represented by $\circ s\circ$. If $\mathcal{C}$ is circular, it is simply represented by s (we can start to build s in any symbol of $\mathcal{C}$). An example of a pair of genomes is given in Fig. 1.

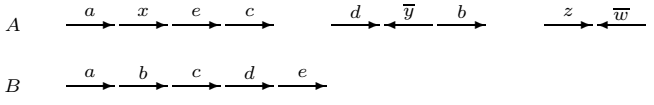


Fig. 1. For genomes $A = \{\circ axec\circ, \circ d\overline{y}b\circ, \circ z\overline{w}\circ\}$, composed of three linear chromosomes, and $B = \{\circ abcde\circ\}$, composed of one single chromosome, we have $\mathcal{G} = \{a, b, c, d, e\}$, $\mathcal{A} = \{x, y, z, w\}$ and $\mathcal{B} = \emptyset$

2.1 Adjacency Graph, the DCJ Operation and DCJ Distance

First we recall some concepts given in [4], that are generalizations of definitions introduced by Bergeron *et al.* [1].

For each marker $g \in \mathcal{G}$, denote its two extremities by g^t (tail) and g^h (head). Given a genome A, a $\mathcal{G}$-*adjacency* in A is in general a linear string $v = \gamma_1\ell\gamma_2$, such that γ_1 and γ_2 are telomeres or extremities of markers in $\mathcal{G}$ and ℓ, the substring composed of the markers that are between γ_1 and γ_2 in A, contains no marker that also belongs to $\mathcal{G}$. The substring ℓ is said to be the *label* of v, and the extremities γ_1 and γ_2 are said to be $\mathcal{G}$-*adjacent*. If ℓ is a non-empty string, v is said to be *labeled*, otherwise v is said to be *clean*. Observe that a $\mathcal{G}$-adjacency $\gamma_1\ell\gamma_2$ can also be represented by $\gamma_2\overline{\ell}\gamma_1$. Moreover, a labeled $\mathcal{G}$-adjacency $u = \circ\ell\circ$ indicates that A contains a linear chromosome composed only of markers that are not in $\mathcal{G}$, that is, u corresponds to a whole linear chromosome. In the same way, if s is a circular chromosome in A composed only of markers that are not in $\mathcal{G}$, then s is also a $\mathcal{G}$-adjacency. This is the only special case of $\mathcal{G}$-adjacency in which we have a circular instead of a linear string.

Two genomes A and B can be also represented by the sets $V_{\mathcal{G}}(A)$ and $V_{\mathcal{G}}(B)$, containing their $\mathcal{G}$-adjacencies. For the genomes in Figure 1, we have $\mathcal{G} = \{a, b, c, d, e\}$, $V_{\mathcal{G}}(A) = \{\circ a^t, a^h x e^t, e^h c^t, c^h \circ, \circ d^t, d^h \overline{y} b^t, b^h \circ, \circ z \overline{w} \circ\}$ and $V_{\mathcal{G}}(B) = \{\circ a^t, a^h b^t, b^h c^t, c^h d^t, d^h e^t, e^h \circ\}$. Observe that, since $\mathcal{B} = \emptyset$, $V_{\mathcal{G}}(B)$ contains only clean adjacencies.

A *cut* performed on a genome A separates two adjacent markers of A. A cut affects a $\mathcal{G}$-adjacency v of $V_{\mathcal{G}}(A)$ as follows: if v is linear, the cut is done between two symbols of v, creating two open ends in two separate linear strings; if v is circular, the cut creates two open ends in one linear string. A *double-cut and join* or DCJ applied on a genome A is the operation that performs two cuts in $V_{\mathcal{G}}(A)$, creating four open ends, and joins these open ends in a different way. As an example, considering the genome A from Fig. 1 and $\mathcal{G} = \{a, b, c, d, e\}$, if we apply a DCJ on $a^h x e^t$ and $d^h \overline{y} b^t$ of $V_{\mathcal{G}}(A)$ we can create $a^h b^t$ and $d^h \overline{y} x e^t$.

The problem of sorting A into B can be studied with the help of the following graph, introduced by Bergeron *et al.* [1]. The *adjacency graph* $AG(A, B)$ is the graph that has a vertex for each $\mathcal{G}$-adjacency in $V_{\mathcal{G}}(A)$ and a vertex for each $\mathcal{G}$-adjacency in $V_{\mathcal{G}}(B)$. Then, for each $g \in \mathcal{G}$, we have one edge connecting the vertex in $V_{\mathcal{G}}(A)$ and the vertex in $V_{\mathcal{G}}(B)$ that contain g^h and one edge connecting the vertex in $V_{\mathcal{G}}(A)$ and the vertex in $V_{\mathcal{G}}(B)$ that contain g^t. Due to the 1-to-1 correspondence between the vertices of $AG(A, B)$ and the $\mathcal{G}$-adjacencies in $V_{\mathcal{G}}(A)$ and $V_{\mathcal{G}}(B)$, we can identify each adjacency with its corresponding vertex.

The graph $AG(A, B)$ is a collection of connected components and can have cycles and paths, that alternate vertices in $V_{\mathcal{G}}(A)$ and $V_{\mathcal{G}}(B)$ [1]. A path that has one endpoint in $V_{\mathcal{G}}(A)$ and the other in $V_{\mathcal{G}}(B)$ is called an *AB-path*. In the same way, both endpoints of an *AA-path* are in $V_{\mathcal{G}}(A)$, as well as both endpoints of a *BB-path* are in $V_{\mathcal{G}}(B)$. Furthermore, $AG(A, B)$ can have two extra types of components: each $\mathcal{G}$-adjacency that corresponds to a linear (respect. circular) chromosome is a *linear* (respect. *circular*) *singleton*. Linear singletons are particular cases of AA-paths and BB-paths. When $\mathcal{A} = \mathcal{B} = \emptyset$, the adjacency graph is composed only of clean $\mathcal{G}$-adjacencies and has no singletons. In this case the graph is said to be *clean*. An example of an adjacency graph is given in Fig. 2.

Singletons, AB-paths composed of one single edge, and cycles composed of two edges are said to be *DCJ-sorted*. Longer paths and cycles are said to be *DCJ-unsorted*. The procedure of using DCJ operations to turn $AG(A, B)$ into DCJ-sorted components is called *DCJ-sorting* of A into B. The *DCJ distance* of A and B, denoted by $d_{DCJ}(A, B)$, corresponds to the minimum number of steps required to do a DCJ-sorting of A into B and can be easily obtained:

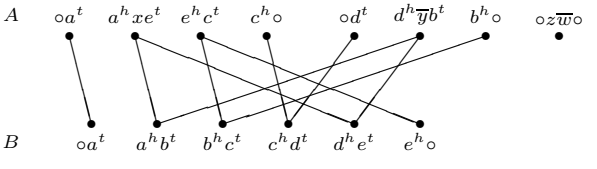


Fig. 2. For genomes $A = \{\circ axec\circ, \circ d\overline{y}b\circ, \circ z\overline{w}\circ\}$ and $B = \{\circ abcde\circ\}$, the adjacency graph contains one cycle, two AA-paths (one is a linear singleton) and two AB-paths

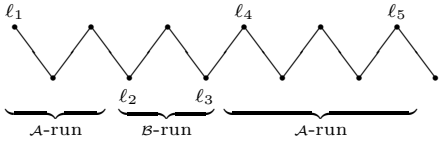


Fig. 3. An AB-path with 3 runs. Only the labels of the $\mathcal{G}$-adjacencies are represented.

Theorem 1 ([1]). *Given a genome A over $\mathcal{G}_A$ and a genome B over $\mathcal{G}_B$, we have $d_{DCJ}(A,B) = n - c - \frac{b}{2}$, where n is the number of markers in $\mathcal{G} = \mathcal{G}_A \cap \mathcal{G}_B$, and c and b are, respectively, the number of cycles and AB-paths in $AG(A,B)$*

Bergeron *et al.* [1] observed that the number of AB-paths in $AG(A,B)$ is even and that an *optimal* DCJ either increases the number of cycles by one, or the number of AB-paths by two (decreasing the DCJ distance by one). In the same way, a *neutral* DCJ does not affect the number of cycles and AB-paths in the graph, while a *counter-optimal* DCJ either decreases the number of cycles by one, or the number of AB-paths by two. The problem of finding an optimal sequence of operations that do a DCJ-sorting of A into B can be solved with a simple greedy linear time algorithm [1].

2.2 Indel Operations and DCJ-Indel Distance

The markers in $\mathcal{A}$ and $\mathcal{B}$ are represented in the adjacency graph as labels and singletons, and, in order to completely sort genome A into genome B, the markers in $\mathcal{A}$ have to be deleted, while the markers in $\mathcal{B}$ have to be inserted. No classical DCJ operation is able to perform an insertion or a deletion. Moreover, no operation is able to delete and insert at the same time (such an event would be a replacement, that is not accepted in the model we consider). An operation is thus either a DCJ, or an *insertion*, or a *deletion*. We refer to insertions and deletions as *indel* operations. A DCJ and an indel operation have the same cost and the *DCJ-indel distance* of A and B, denoted by $d_{DCJ}^{id}(A,B)$, is the minimum number of DCJ and indel operations required to transform A into B.

Given a component C of $AG(A,B)$, we can obtain a string $\ell(C)$ by the concatenation of the labels of the $\mathcal{G}$-adjacencies of C in the order in which they appear. Cycles, AA-paths and BB-paths can be read in any direction, but AB-paths should always be read from A to B. If C is a cycle and has labels in both genomes A and B, we should start to read in a labeled $\mathcal{G}$-adjacency v of A, such that the first labeled vertex before v is a $\mathcal{G}$-adjacency in B; otherwise C has labels in at most one genome and we can start anywhere. Each maximal substring of $\ell(C)$ composed only of markers in $\mathcal{A}$ (respectively in $\mathcal{B}$) is called an *$\mathcal{A}$-run* (respectively a *$\mathcal{B}$-run*). Each $\mathcal{A}$-run or $\mathcal{B}$-run can be simply called *run*. A component composed only of clean $\mathcal{G}$-adjacencies has no run and is said to be *clean*, otherwise the component is *labeled*. We denote by $\Lambda(C)$ the number of runs in a component C. A path can have any number of runs, while a cycle has zero, one, or an even number of runs. Fig. 3 shows an AB-path with 3 runs.

Observe that a $\mathcal{G}$-adjacency with a non-empty label ℓ can be cut in at least two different positions, either before or after ℓ. Since the position of the cut does not change the effect of the DCJ operation on $d_{DCJ}(A, B)$, we can choose to cut at positions that allow the concatenation of the labels of the original $\mathcal{G}$-adjacencies. As a consequence, an $\mathcal{A}$-run can be first *accumulated* with optimal DCJ operations and later deleted at once from genome A. In the same way, a $\mathcal{B}$-run can be first inserted at once as a *cluster* in genome A and later split with optimal DCJ operations [4], as we can see in Fig. 4. This gives directly the following upper bound to the DCJ-indel distance: $d^{id}_{DCJ}(A, B) \leq d_{DCJ}(A, B) + \sum_{C \in AG(A,B)} \Lambda(C)$. However, since a DCJ operation can merge runs in the components of $AG(A, B)$, this upper bound can be improved.

Given two genomes A and B and a component $C \in AG(A, B)$, we denote by $d_{DCJ}(C)$ the minimum number of operations required to do a separate DCJ-sorting in C, applying DCJs only on vertices of C (or vertices that result from DCJs applied on vertices that were in C). It is possible to do a separate DCJ-sorting using only optimal DCJs in any component of $AG(A, B)$ [3], thus $d_{DCJ}(A, B) = \sum_{C \in AG(A,B)} d_{DCJ}(C)$. The optimal DCJs sorting C can merge runs in C and we denote by $\lambda(C)$ the minimum number of runs that can be obtained with a separate DCJ-sorting in C using optimal DCJs:

Proposition 1 ([4]). *Given a component C in $AG(A, B)$, we have $\lambda(C) = \lceil \frac{\Lambda(C)+1}{2} \rceil$, if $\Lambda(C) \geq 1$. Otherwise $\lambda(C) = 0$.*

The number λ allows to define an exact formula for the DCJ-indel distance [4]. If λ_0 and λ_1 are, respectively, the sum of the number λ for the components of the adjacency graph before and after a DCJ ρ, let $\Delta\lambda(\rho) = \lambda_1 - \lambda_0$. Moreover, let $\Delta_{dcj}(\rho)$ be respectively 0, +1 and +2 depending whether ρ is optimal, neutral or counter-optimal, and $\Delta d(\rho) = \Delta_{dcj}(\rho) + \Delta\lambda(\rho)$.

By the definition of λ, any optimal DCJ ρ acting on a single component has $\Delta\lambda(\rho) \geq 0$. Furthermore, if a component C is DCJ-unsorted, it is always possible to find an optimal DCJ acting only on C with $\Delta\lambda(\rho) = 0$. We also know that any neutral or counter-optimal DCJ acting on a single component has $\Delta\lambda \geq -1$,

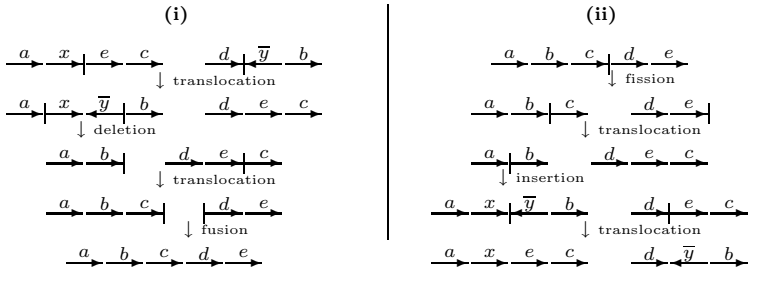


Fig. 4. (i) An optimal scenario sorting $\{\circ axec\circ, \circ d\overline{y}b\circ\}$ into $\{\circ abcde\circ\}$. The first operation (translocation) accumulates x and y, so that they can be deleted at once. (ii) Conversely, while sorting $\{\circ abcde\circ\}$ into $\{\circ axec\circ, \circ d\overline{y}b\circ\}$, we can insert a cluster at once and later split it with a translocation.

Table 1. Path recombinations that have $\Delta d \leq -1$ [4]. Optimal recombinations are in the left, neutral recombinations in the right.

sources	resultants	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_{\mathcal{AB}} + BB_{\mathcal{AB}}$	$AB_\bullet + AB_\bullet$	-2	0	-2
$AA_{\mathcal{A}} + BB_{\mathcal{AB}}$	$AB_\bullet + AB_{\mathcal{AB}}$	-1	0	-1
$BB_{\mathcal{A}} + AA_{\mathcal{AB}}$	$AB_\bullet + AB_{\mathcal{BA}}$	-1	0	-1
$AA_{\mathcal{B}} + BB_{\mathcal{AB}}$	$AB_\bullet + AB_{\mathcal{AB}}$	-1	0	-1
$BB_{\mathcal{B}} + AA_{\mathcal{AB}}$	$AB_\bullet + AB_{\mathcal{AB}}$	-1	0	-1
$AA_{\mathcal{A}} + BB_{\mathcal{A}}$	$AB_\bullet + AB_\bullet$	-1	0	-1
$AA_{\mathcal{B}} + BB_{\mathcal{B}}$	$AB_\bullet + AB_\bullet$	-1	0	-1

sources	resultants	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_{\mathcal{AB}} + AA_{\mathcal{AB}}$	$AA_{\mathcal{A}} + AA_{\mathcal{B}}$	-2	$+1$	-1
$BB_{\mathcal{AB}} + BB_{\mathcal{AB}}$	$BB_{\mathcal{A}} + BB_{\mathcal{B}}$	-2	$+1$	-1
$AA_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_\bullet + AA_{\mathcal{A}}$	-2	$+1$	-1
$AA_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_\bullet + AA_{\mathcal{B}}$	-2	$+1$	-1
$BB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_\bullet + BB_{\mathcal{B}}$	-2	$+1$	-1
$BB_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_\bullet + BB_{\mathcal{A}}$	-2	$+1$	-1
$AB_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_\bullet + AB_\bullet$	-2	$+1$	-1

consequently we have $\Delta d \geq 0$ in these cases [4]. Let $d^{id}_{DCJ}(C) = d_{DCJ}(C) + \lambda(C)$ be the number of steps required to sort separately a component C [4]. This gives a tight upper bound to the distance formula:

Lemma 1 ([4]). *Given two genomes A and B without duplications, we have*

$$d^{id}_{DCJ}(A,B) \leq d_{DCJ}(A,B) + \sum_{C \in AG(A,B)} \lambda(C).$$

Proof. We can sort the components separately with $\sum_{C \in AG(A,B)} d^{id}_{DCJ}(C)$ steps, which corresponds exactly to $d_{DCJ}(A,B) + \sum_{C \in AG(A,B)} \lambda(C)$. □

An exact formula can be derived from the upper bound given by Lemma 1 and a particular type of DCJ operation that acts on two components and is called *recombination*. The two components in which the cuts are applied are called *sources* and those obtained after the joinings are called *resultants* of the recombination. We know that any recombination ρ has $\Delta\lambda(\rho) \geq -2$ [4].

Any recombination applied to a vertex of an AA-path and a vertex of a BB-path is optimal [3]. A recombination applied to vertices of two different AB-paths can be either neutral, when the result is also a pair of AB-paths, or counter-optimal, when the result is a pair composed of an AA-path and a BB-path. All other types of path recombinations are neutral and all recombinations involving at least one cycle are counter-optimal. Any counter-optimal recombination has $\Delta d \geq 0$, thus only path recombinations can have $\Delta d \leq -1$.

Now let $\mathcal{A}$ (respectively $\mathcal{B}$) be a sequence with an odd (≥ 1) number of runs, starting and ending with an $\mathcal{A}$-run (respectively $\mathcal{B}$-run). We can then make any combination of $\mathcal{A}$ and $\mathcal{B}$, such as $\mathcal{AB}$, that is a sequence with an even (≥ 2) number of runs, starting with an $\mathcal{A}$-run and ending with a $\mathcal{B}$-run. An empty sequence (with no run) is represented by ε. Each one of the notations AA_ε, $AA_{\mathcal{A}}$, $AA_{\mathcal{B}}$, $AA_{\mathcal{AB}}$, BB_ε, $BB_{\mathcal{A}}$, $BB_{\mathcal{B}}$, $BB_{\mathcal{AB}}$, AB_ε, $AB_{\mathcal{A}}$, $AB_{\mathcal{B}}$, $AB_{\mathcal{AB}}$ and $AB_{\mathcal{BA}}$ represents a particular type of path (AA, BB or AB) with a particular structure of runs (ε, $\mathcal{A}$, $\mathcal{B}$, $\mathcal{AB}$ or $\mathcal{BA}$). Table 1 gives the recombinations with $\Delta d \leq -1$. We denote by $AB_\bullet$ an AB-path that can not be a source of a recombination in Table 1, such as AB_ε, $AB_{\mathcal{A}}$ and $AB_{\mathcal{B}}$. And in Table 2 we list recombinations with $\Delta d = 0$ that create at least one source of recombinations in Table 1.

Considering that some resultants of recombinations can be used in other recombinations of Table 1, Braga *et al.* [4] identified relevant recombination groups,

Table 2. Recombinations that have $\Delta d = 0$ and create resultants that can be used in recombinations with $\Delta d \leq -1$ [4].

sources	resultants	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_{\mathcal{A}} + AB_{\mathcal{BA}}$	$AB_{\varepsilon} + AA_{\mathcal{AB}}$	-1	$+1$	0
$AA_{\mathcal{B}} + AB_{\mathcal{AB}}$	$AB_{\varepsilon} + AA_{\mathcal{AB}}$	-1	$+1$	0
$BB_{\mathcal{A}} + AB_{\mathcal{AB}}$	$AB_{\varepsilon} + BB_{\mathcal{AB}}$	-1	$+1$	0
$BB_{\mathcal{B}} + AB_{\mathcal{BA}}$	$AB_{\varepsilon} + BB_{\mathcal{AB}}$	-1	$+1$	0

sources	resultants	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_{\mathcal{A}} + BB_{\mathcal{B}}$	$AB_{\varepsilon} + AB_{\mathcal{AB}}$	0	0	0
$AA_{\mathcal{B}} + BB_{\mathcal{A}}$	$AB_{\varepsilon} + AB_{\mathcal{BA}}$	0	0	0
$AB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AA_{\mathcal{A}} + BB_{\mathcal{B}}$	-2	$+2$	0
$AB_{\mathcal{BA}} + AB_{\mathcal{BA}}$	$AA_{\mathcal{B}} + BB_{\mathcal{A}}$	-2	$+2$	0

called P, Q, T, S, M and N-recombinations (see Table 5 in the Appendix). An approach that greedily maximizes the number of P, Q, T, S, M and N-recombinations, in this order, has been proposed, so that the exact formula for the DCJ-indel distance can be computed in linear time [4]:

Theorem 2 ([4]). *Given two genomes A and B without duplications, we have*

$$d_{DCJ}^{id}(A,B) = d_{DCJ}(A,B) + \sum_{C \in AG(A,B)} \lambda(C) - 2P - 3Q - 2T - S - 2M - N,$$

where the number of P, Q, T, S, M and N-recombinations are computed by the algorithm given in [4].

3 DCJ-Indel Sorting

Here we analyze the problem of sorting genome A into B with DCJ and indel operations. Consider a DCJ ρ applied on $\gamma_1\ell_1\gamma_4$ and $\gamma_3\ell_2\gamma_2$, that creates $\gamma_1\ell_1\ell_2\gamma_2$ and $\gamma_3\gamma_4$. We represent such an operation as $\rho = (\{\gamma_1\ell_1|\gamma_4, \gamma_3|\ell_2\gamma_2\} \rightarrow \{\gamma_1\ell_1|\ell_2\gamma_2, \gamma_3|\gamma_4\})$. Furthermore, if ρ is the deletion of ℓ from the $\mathcal{G}$-adjacency $\gamma_1\ell\gamma_2$, we represent this as $\rho = (\gamma_1|\ell|\gamma_2 \rightarrow \gamma_1|\gamma_2)$. In the same way, the insertion of block ℓ in the $\mathcal{G}$-adjacency $\gamma_1\gamma_2$ is represented as $(\gamma_1|\gamma_2 \rightarrow \gamma_1|\ell|\gamma_2)$.

Particular cases happen when we have circular singletons. A DCJ involving two $\mathcal{G}$-adjacencies such that at least one is a circular singleton always result in only one $\mathcal{G}$-adjacency: $(\{|\ell_1|, \gamma_1|\ell_2\gamma_2\} \rightarrow \gamma_1|\ell_1|\ell_2\gamma_2)$ or $(\{|\ell_1|, |\ell_2|\} \rightarrow |\ell_1|\ell_2|)$. Conversely, a DCJ operation with two cuts in one $\mathcal{G}$-adjacency always result in two $\mathcal{G}$-adjacencies, such that at least one is a circular singleton: $(\gamma_1|\ell_1|\ell_2\gamma_2 \rightarrow \{|\ell_1|, \gamma_1|\ell_2\gamma_2\})$ or $(|\ell_1|\ell_2| \rightarrow \{|\ell_1|, |\ell_2|\})$. A deletion of a circular singleton ℓ is represented by $(\ell \rightarrow \emptyset)$ and its insertion is $(\emptyset \rightarrow \ell)$.

Given any operation $\rho = (X \rightarrow Y)$, we define the *inversion* of ρ as $\rho^{-1} = (Y \rightarrow X)$. Observe that the inversion of a deletion is an insertion, and *vice-versa*. We can also extend this notation to a sequence of operations: given a sequence $s = \rho_1\rho_2\ldots\rho_n$, we have $s^{-1} = \rho_n^{-1}\rho_{n-1}^{-1}\ldots\rho_2^{-1}\rho_1^{-1}$.

The approach presented in the previous section allows operations on both A and B, in order to be able to concatenate labels in $\mathcal{G}$-adjacencies of both genomes. Regarding the operations applied on B, this can be seen as a backtracing to find the best moment to do a cluster insertion in A, as shown in Fig. 5.

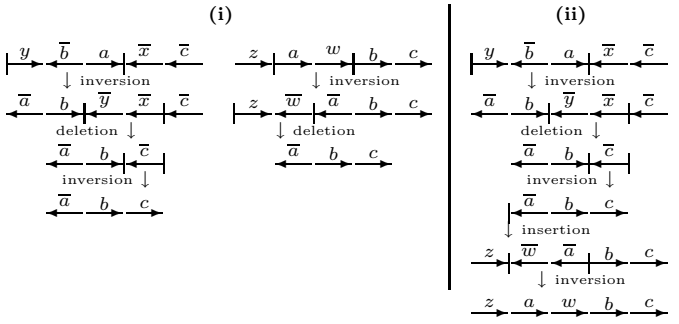


Fig. 5. (i) Two sequences of lengths 3 and 2, sorting $A = \{\circ y\overline{b}a\overline{x}\overline{c}\circ\}$ and $B = \{\circ zawbc\circ\}$ into $\{\circ\overline{a}bc\circ\}$. (ii) A corresponding sequence of length 5 sorting A into B.

Proposition 2. *Given two genomes A and B, any pair of sequences s_1 and s_2 composed of DCJ and indel operations acting on both genomes A and B, transforming respectively A and B into an intermediate genome I, has a corresponding sequence $s_1 s_2^{-1}$ that transforms A into B.*

Proof. Since s_1 sorts A into I and s_2^{-1} sorts I into B, $s_1 s_2^{-1}$ sorts A into B. □

Observe that, according to the presented approach, the operations applied on A are sorting operations with respect to B and the operations applied on B are sorting operations with respect to A. Thus, if s_1 is the sequence of operations applied on A and s_2 is the sequence of operations applied on B, then $s_1 s_2^{-1}$ is an optimal sequence of operations sorting A into B. Indeed, if this was not the case, at least one operation in s_1 or s_2 would be non-sorting, which is a contradiction.

3.1 Sorting with a Minimum Number of DCJ Operations

First we will derive a sorting algorithm directly from the formula given in Theorem 2. In general lines the algorithm constructs incrementally two sequences

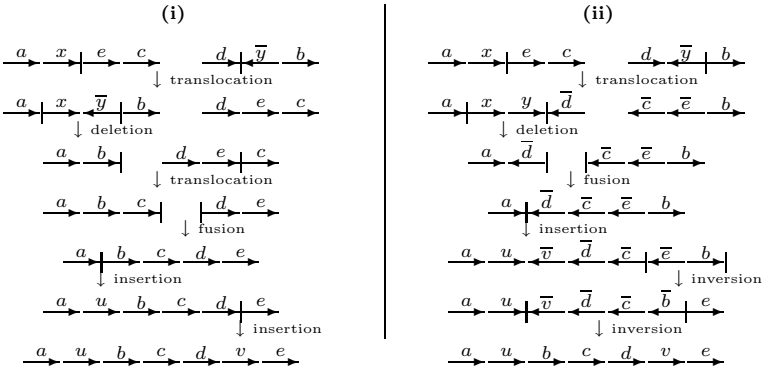


Fig. 6. Two optimal scenarios sorting $\{\circ axec\circ, \circ d\overline{y}b\circ\}$ into $\{\circ aubcdve\circ\}$. (i) Minimizing DCJs gives 3 DCJs and 3 indels. (ii) Minimizing indels gives 4 DCJs and 2 indels.

of operations, s_1 and s_2, such that the operations in s_1 are applied on A and the operations in s_2 are applied on B (although we do not explicitly identify in the pseudo-code which operations are in s_1 and which operations are in s_2). As previously stated, an optimal sequence sorting A into B is given by $s_1 s_2^{-1}$.

Algorithm 1. Sorting genome A into B with minimum number of DCJs

1. Apply all P, Q, T, S, M and N-recombinations, in this order.
2. For each component $C \in AG(A, B)$:
   (a) Split C with **optimal** DCJs (that have $\boldsymbol{\Delta\lambda = 0}$) until only components that have at most 2 runs are obtained and the total number of runs in all new components is equal to $\lambda(C)$.
   (b) Accumulate all runs in the smaller components derived from C with **optimal** DCJ operations (that have $\boldsymbol{\Delta\lambda = 0}$).
   (c) Apply **optimal** DCJ operations (that have $\boldsymbol{\Delta\lambda = 0}$) in the smaller components derived from C until only DCJ-sorted components exist.
   (d) **Delete** all runs in the DCJ-sorted components derived from C.

The given algorithm does the minimum number of DCJs with $\Delta\lambda \leq -1$ (only those that are in recombinations of step 1). All other operations applied are optimal DCJs with $\Delta\lambda = 0$ and indels. As a consequence, this gives a sorting scenario that minimizes the number of DCJs, with respect to indels.

However, the space of solutions of the sorting problem contains scenarios with different compositions and, using the same number of steps, one could look for a scenario with less indels and more DCJ operations. Fig. 6 shows examples of two different approaches: the algorithm given above, that minimizes the number of DCJs, and a second approach, that minimizes the number of indels. The second approach will be presented in the following.

3.2 Sorting with a Minimum Number of Indels

In order to design an algorithm to sort a genome A into a genome B minimizing the number of indels, we need to use the DCJs with lower $\Delta\lambda$, among those with $\Delta d = 0$. Thus, instead of using optimal DCJs with $\Delta\lambda = 0$, as in steps 2(a,b,c) of the previous algorithm, we shall maximize the use of counter-optimal DCJs with $\Delta\lambda = -2$ and neutral DCJs with $\Delta\lambda = -1$.

First we will analyze the operations acting on a single component. In this case, any counter-optimal DCJ has $\Delta\lambda \geq 0$. The same happens with any neutral DCJ acting on a single component C with $\Lambda(C) \leq 3$ [4]. Only neutral DCJs acting on a single component with at least 4 runs can have $\Delta\lambda = -1$:

Proposition 3. *Given a component C with $\Lambda(C) \geq 4$, the best we can get with a neutral DCJ operation ρ acting only on C is $\Delta\Lambda(\rho) = -2$ and $\Delta\lambda(\rho) = -1$.*

Proof. When the component has $\Lambda \geq 4$, the operation ρ can cut after the first and before the last run, and simply invert this segment. In this case we get $\Delta\Lambda(\rho) = -2$, resulting in $\lambda_1 = \lceil \frac{(\Lambda(C)-2)+1}{2} \rceil = \lceil \frac{\Lambda(C)+1}{2} \rceil - 1 = \lambda_0 - 1$. (Since we can apply only two cuts and two joins, it is not possible to do better). □

Table 3. Further neutral and counter-optimal recombinations with $\Delta d = 0$ (in addition to those listed in Table 2). Each operation here has a resultant of the same type of one of its sources.

sources	resultants	$\Delta\lambda$	Δ_{dcj}	Δd
$Y_{\mathcal{AB}} + C_1^*$	C_1^*	-2	$+2$	0
$S_{\mathcal{A}} + C_2^*$	C_2^*	-1	$+1$	0
$S_{\mathcal{B}} + C_3^*$	C_3^*	-1	$+1$	0
$AB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_\varepsilon + AB_{\mathcal{AB}}$	-1	$+1$	0
$AB_{\mathcal{BA}} + AB_{\mathcal{BA}}$	$AB_\varepsilon + AB_{\mathcal{BA}}$	-1	$+1$	0

sources	resultants	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_{\mathcal{A}} + P_1^*$	$AA_\varepsilon + P_1^*$	-1	$+1$	0
$AA_{\mathcal{B}} + P_2^*$	$AA_\varepsilon + P_2^*$	-1	$+1$	0
$BB_{\mathcal{A}} + P_3^*$	$BB_\varepsilon + P_3^*$	-1	$+1$	0
$BB_{\mathcal{B}} + P_4^*$	$BB_\varepsilon + P_4^*$	-1	$+1$	0
$AB_{\mathcal{A}} + P_5^*$	$AB_\varepsilon + P_5^*$	-1	$+1$	0
$AB_{\mathcal{B}} + P_6^*$	$AB_\varepsilon + P_6^*$	-1	$+1$	0

* The components C_1, C_2, C_3, P_1, P_2, P_3, P_4, P_5 and P_6 can be:

C_1: $Y_{\mathcal{AB}}$, $AA_{\mathcal{AB}}$, $AA_{\mathcal{ABA}}$, $AA_{\mathcal{BAB}}$, $BB_{\mathcal{AB}}$, $BB_{\mathcal{ABA}}$, $BB_{\mathcal{BAB}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$, $AB_{\mathcal{ABA}}$ or $AB_{\mathcal{BAB}}$
C_2: $S_{\mathcal{A}}$, $Y_{\mathcal{A}}$, $Y_{\mathcal{AB}}$, $AA_{\mathcal{A}}$, $AA_{\mathcal{AB}}$, $AA_{\mathcal{BAB}}$, $BB_{\mathcal{A}}$, $BB_{\mathcal{AB}}$, $BB_{\mathcal{BAB}}$, $AB_{\mathcal{A}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{BAB}}$
C_3: $S_{\mathcal{B}}$, $Y_{\mathcal{B}}$, $Y_{\mathcal{AB}}$, $AA_{\mathcal{B}}$, $AA_{\mathcal{AB}}$, $AA_{\mathcal{ABA}}$, $BB_{\mathcal{B}}$, $BB_{\mathcal{AB}}$, $BB_{\mathcal{ABA}}$, $AB_{\mathcal{B}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{ABA}}$

P_1: $AA_{\mathcal{AB}}$, $AA_{\mathcal{A}}$ or $AB_{\mathcal{AB}}$
P_2: $AA_{\mathcal{AB}}$, $AA_{\mathcal{B}}$ or $AB_{\mathcal{BA}}$
P_3: $BB_{\mathcal{AB}}$, $BB_{\mathcal{A}}$ or $AB_{\mathcal{BA}}$
P_4: $BB_{\mathcal{AB}}$, $BB_{\mathcal{B}}$ or $AB_{\mathcal{AB}}$
P_5: $AA_{\mathcal{A}}$, $BB_{\mathcal{A}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{A}}$
P_6: $AA_{\mathcal{B}}$, $BB_{\mathcal{B}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{B}}$

Algorithm 2: Sorting genome A into B with minimum number of indels

1. Apply all P, Q, T, S, M and N-recombinations, in this order.
2. While there is a DCJ ρ, such that ρ is either a **counter-optimal** recombination from Table 3, or a **neutral** recombination from Table 2 or 3, or an **optimal** recombination from Table 2, apply ρ. [After this step, there is at most one component with 2 or more runs; the others have at most one run.]
3. For each component $C \in AG(A, B)$:
   (a) While $\Lambda(C) \geq 4$, apply a **neutral** DCJ on C with $\boldsymbol{\Delta\lambda = -1}$ (Prop. 3).
   (b) If $\Lambda(C) = 3$ (C is a path), merge the last and the first runs of C extracting a cycle with all runs (**optimal** DCJ with $\boldsymbol{\Delta\lambda = 0}$).
   (c) Accumulate all runs in the smaller components derived from C with **optimal** DCJ operations that have $\boldsymbol{\Delta\lambda = 0}$.
   (d) Apply **optimal** DCJ operations in the smaller components derived from C until only DCJ-sorted components exist (these DCJs have $\boldsymbol{\Delta\lambda = 0}$).
   (e) **Delete** all runs in the DCJ-sorted components derived from C.

Proposition 3 guarantees that, with neutral DCJs, we can merge runs in any component C with $\Lambda(C) \geq 4$, such that in the end of the process C has only 2 or 3 runs. In order to maximize the use of these neutral DCJs, a good strategy is to first regroup runs efficiently in one component. This can be done using recombinations with $\Delta d = 0$, as those listed in Table 2. Additional recombinations with $\Delta d = 0$ are listed in Table 3, in which $Y_{\mathcal{AB}}$ is a cycle with at least one $\mathcal{A}$-run and one $\mathcal{B}$-run, and $S_{\mathcal{A}}$ and $S_{\mathcal{B}}$ are circular singletons in genomes A and B, respectively. All recombinations in Table 3 regroup all remaining runs in one single component, thus they should be applied before the neutral DCJs from Proposition 3. The same happens with the neutral and optimal recombinations in Table 2. (Instead of using the 2 counter-optimal recombinations of Table 2 we use the 2 neutral recombinations of Table 3 that have the same sources, but regroup all remaining runs in one component.)

From the previous observations we can derive an algorithm to sort a genome A into a genome B minimizing the number of insertions and deletions.

Table 4. Comparing *R. bellii* (1.52 Mbp) with six other species of *Rickettsia*

species	Mbp	d^{id}_{DCJ}	MIN DCJs DCJs+indels	MIN indels DCJs+indels	species	Mbp	d^{id}_{DCJ}	MIN DCJs DCJs+indels	MIN indels DCJs+indels
R. felis	1.55	493	312 + 181	389 + 104	*R. conorii*	1.27	414	261 + 153	313 + 101
R. massiliae	1.36	448	276 + 172	340 + 108	*R. prowazekii*	1.11	314	197 + 117	216 + 98
R. africae	1.28	426	260 + 166	322 + 104	*R. typhi*	1.11	309	195 + 114	212 + 97

3.3 Experiments

We compared the two approaches using a database of seven well annotated genomes of *Rickettsia* bacteria [2]. Six of the species available in [2] are closely related. The exception is *R. bellii*, which shows a high level of rearrangement with respect to the others. We compared *R. bellii* with the other six species, using both approaches, one that minimizes DCJ operations and one that minimizes indels. The results are presented in Table 4.

The purpose of these experiments is to show the contrast of the number of clusters that can be obtained with the two approaches. In the particular case of this dataset, we observed that, while the number of indels varies from 114 to 181 with the algorithm that maximizes indels, we found a much smaller variation (from 97 to 108) with the algorithm that minimizes indels. However, a careful biological analysis of the data would be necessary to verify whether these differences could indicate which approach is better.

4 Final Remarks

In this work we developed algorithms to sort one genome into another one using DCJ and indel operations. We show that, in the space of solutions of this problem, the optimal sorting scenarios can have different compositions with respect to the number of each type of operation. We took a first step in the exploration of this solution space proposing approaches that give two different types of sorting scenarios: one that minimizes the number of DCJ operations (respectively maximizing the number of insertions and deletions) and one that minimizes the number of insertions and deletions (respectively maximizing the number of DCJ operations). However, the characterization of the whole space of solutions remains an open problem.

References

1. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
2. Blanc, G., et al.: Reductive genome evolution from the mother of Rickettsia. PLoS Genetics 3(1), e14 (2007)
3. Braga, M.D.V., Stoye, J.: The solution space of sorting by DCJ. To appear in Journal of Computational Biology (2010)

4. Braga, M.D.V., Willing, E., Stoye, J.: Genomic distance with DCJ and indels. In: Proceedings of WABI (to appear, 2010)
5. El-Mabrouk, N.: Sorting Signed Permutations by Reversals and Insertions/Deletions of Contiguous Segments. Journal of Discrete Algorithms 1(1), 105–122 (2001)
6. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). Journal of the ACM 46, 1–27 (1999)
7. Ouangraoua, A., Bergeron, A.: Parking functions, labeled trees and DCJ sorting scenarios. In: Ciccarelli, F.D., Miklós, I. (eds.) RECOMB-CG 2009. LNCS (LNBI), vol. 5817, pp. 24–35. Springer, Heidelberg (2009)
8. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics 10, 120 (2009); Preliminary version in Crandall, K.A., Lagergren, J. (eds.): WABI 2008. LNCS (LNBI), vol. 5251, pp. 1–13. Springer, Heidelberg (2008)
9. Yancopoulos, S., Friedberg, R.: DCJ path formulation for genome transformations which include insertions, deletions, and duplications. Journal of Computational Biology 16(10), 1311–1338 (2009); Preliminary version in Nelson, C.E., Vialette, S. (eds.): RECOMB-CG 2008. LNCS (LNBI), vol. 5267, pp. 170–183. Springer, Heidelberg (2008)
10. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)

Appendix: P, Q, T, S, M and N-Recombinations

Table 5. The groups of P, Q, T, S, M and N-recombinations, identified by Braga *et al.* [4] with the help of Tables 1 and 2. The column **scr** indicates the contribution of each path in the distance decrease.

	sources	resultants	Δd	scr
P	$AA_{\mathcal{AB}} + BB_{\mathcal{AB}}$	$2AB_\bullet$	-2	-1
Q	$2AA_{\mathcal{AB}} + BB_{\mathcal{A}} + BB_{\mathcal{B}}$	$4AB_\bullet$	-3	$-3/4$
	$2BB_{\mathcal{AB}} + AA_{\mathcal{A}} + AA_{\mathcal{B}}$	$4AB_\bullet$	-3	$-3/4$
T	$AA_{\mathcal{AB}} + BB_{\mathcal{A}} + AB_{\mathcal{AB}}$	$3AB_\bullet$	-2	$-2/3$
	$AA_{\mathcal{AB}} + BB_{\mathcal{B}} + AB_{\mathcal{BA}}$	$3AB_\bullet$	-2	$-2/3$
	$BB_{\mathcal{AB}} + AA_{\mathcal{A}} + AB_{\mathcal{BA}}$	$3AB_\bullet$	-2	$-2/3$
	$BB_{\mathcal{AB}} + AA_{\mathcal{B}} + AB_{\mathcal{AB}}$	$3AB_\bullet$	-2	$-2/3$
	$2AA_{\mathcal{AB}} + BB_{\mathcal{A}}$	$2AB_\bullet + AA_{\mathcal{B}}$	-2	$-2/3$
	$2AA_{\mathcal{AB}} + BB_{\mathcal{B}}$	$2AB_\bullet + AA_{\mathcal{A}}$	-2	$-2/3$
	$2BB_{\mathcal{AB}} + AA_{\mathcal{A}}$	$2AB_\bullet + BB_{\mathcal{B}}$	-2	$-2/3$
	$2BB_{\mathcal{AB}} + AA_{\mathcal{B}}$	$2AB_\bullet + BB_{\mathcal{A}}$	-2	$-2/3$

	sources	resultants	Δd	scr
S	$AA_{\mathcal{A}} + BB_{\mathcal{A}}$	$2AB_\bullet$	-1	$-1/2$
	$AA_{\mathcal{B}} + BB_{\mathcal{B}}$	$2AB_\bullet$	-1	$-1/2$
	$AB_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$2AB_\bullet$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AA_{\mathcal{A}}$	$AB_\bullet + AB_{\mathcal{AB}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + BB_{\mathcal{A}}$	$AB_\bullet + AB_{\mathcal{BA}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AA_{\mathcal{B}}$	$AB_\bullet + AB_{\mathcal{BA}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + BB_{\mathcal{B}}$	$AB_\bullet + AB_{\mathcal{AB}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_\bullet + AA_{\mathcal{A}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_\bullet + AA_{\mathcal{B}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_\bullet + BB_{\mathcal{B}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_\bullet + BB_{\mathcal{A}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + AA_{\mathcal{AB}}$	$AA_{\mathcal{B}} + AA_{\mathcal{A}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + BB_{\mathcal{AB}}$	$BB_{\mathcal{B}} + BB_{\mathcal{A}}$	-1	$-1/2$

	sources	resultants	Δd	scr
M	$2AB_{\mathcal{AB}} + AA_{\mathcal{B}} + BB_{\mathcal{A}}$	$4AB_\bullet$	-2	$-1/2$
	$2AB_{\mathcal{BA}} + AA_{\mathcal{A}} + BB_{\mathcal{B}}$	$4AB_\bullet$	-2	$-1/2$
N	$AB_{\mathcal{AB}} + AA_{\mathcal{B}} + BB_{\mathcal{A}}$	$3AB_\bullet$	-1	$-1/3$
	$AB_{\mathcal{BA}} + AA_{\mathcal{A}} + BB_{\mathcal{B}}$	$3AB_\bullet$	-1	$-1/3$

	sources	resultants	Δd	scr
N	$2AB_{\mathcal{AB}} + AA_{\mathcal{B}}$	$2AB_\bullet + AA_{\mathcal{A}}$	-1	$-1/3$
	$2AB_{\mathcal{AB}} + BB_{\mathcal{A}}$	$2AB_\bullet + BB_{\mathcal{B}}$	-1	$-1/3$
	$2AB_{\mathcal{BA}} + AA_{\mathcal{A}}$	$2AB_\bullet + AA_{\mathcal{B}}$	-1	$-1/3$
	$2AB_{\mathcal{BA}} + BB_{\mathcal{B}}$	$2AB_\bullet + BB_{\mathcal{A}}$	-1	$-1/3$

The Zero Exemplar Distance Problem*

Minghui Jiang

Department of Computer Science, Utah State University, Logan, UT 84322, USA
mjiang@cc.usu.edu

Abstract. Given two genomes with duplicate genes, Zero Exemplar Distance is the problem of deciding whether the two genomes can be reduced to the same genome without duplicate genes by deleting all but one copy of each gene in each genome. Blin, Fertin, Sikora, and Vialette recently proved that Zero Exemplar Distance for monochromosomal genomes is NP-hard even if each gene appears at most two times in each genome, thereby settling an important open question on genome rearrangement in the exemplar model. In this paper, we give a very simple alternative proof of this result. We also study the problem Zero Exemplar Distance for multichromosomal genomes without gene order: from one direction, we show that this problem is NP-hard even if each gene appears at most two times in each genome; from the other direction, we show that this problem admits a polynomial-time algorithm if only one of the two genomes has duplicate genes, and is fixed-parameter tractable if the parameter is the maximum number of chromosomes in each genome.

1 Introduction

Given two genomes with duplicate genes, Genome Rearrangement with Gene Families [11] is the problem of deleting all but one copy of each gene in each genome, so as to minimize some rearrangement distance between the two reduced genomes. The minimum rearrangement distance thus attained is called the *exemplar distance* between the two genomes. For example, each of the following two monochromosomal genomes

$$\begin{aligned} G_1: &\quad -4\ +1\ +2\ +3\ -5\ +1\ +2\ +3\ -6 \\ G_2: &\quad -1\ -4\ +1\ +2\ -5\ +3\ -2\ -6\ +3 \end{aligned}$$

has at most two copies of each gene, and each of the following two reduced genomes

$$\begin{aligned} G_1': &\quad -4\ +1\ +2\ -5\ +3\ -6 \\ G_2': &\quad -4\ +1\ +2\ -5\ +3\ -6 \end{aligned}$$

has exactly one copy of each gene. Recall that in the study of genome rearrangement, a *gene* is usually represented by a signed integer: the absolute value of the

* Supported in part by NSF grant DBI-0743670.

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 74–82, 2010.

integer (the unsigned integer) denotes the gene family to which the gene belongs; the sign of the integer denotes the orientation of the gene in its chromosome. Then a *chromosome* is a sequence of signed integers, and a *genome* is a collection of chromosomes.

Genome Rearrangement with Gene Families is not a single problem but a whole class of related problems, because the choice of rearrangement distance is not unique. This choice becomes irrelevant, however, when we ask the fundamental question: *Is the distance zero?* In the example above, the two reduced genomes G'_1 and G'_2 are identical, thus the exemplar distance between the two original genomes G_1 and G_2 is zero for any reasonable choice of rearrangement distance.

In this paper, we study the most basic version of the problem Genome Rearrangement with Gene Families: Given two sequences of signed integers, Zero Exemplar Distance (for monochromosomal genomes) is the problem of deciding whether the two sequences have a common subsequence including each unsigned integer exactly once in either positive or negative form.

Due to its generic nature, the problem Zero Exemplar Distance has been extensively studied by several groups of researchers [5,4,2] focusing on different rearrangement distances, and, not surprisingly, has acquired several different names. Except for trivial distinctions, Zero Exemplar Distance is essentially the same problem as Zero Exemplar Conserved Interval Distance [5], Exemplar Longest Common Subsequence [4], and Zero Exemplar Breakpoint Distance [2].

It is easy to check that Zero Exemplar Distance can be solved in polynomial time if only one of the two genomes has duplicate genes. On the other hand, if both genomes contain duplicate genes, then even if each gene appears at most three times in each genome, the problem Zero Exemplar Distance is already NP-hard, as shown independently in three papers [5,4,2]. The quest for the exact boundary between polynomial solvability and NP-hardness led to the following open question first raised by Chen et al. in 2006:

Question 1 (Chen, Fowler, Fu, and Zhu, 2006 [5]). Is the problem Zero Exemplar Distance for monochromosomal genomes still NP-hard if each gene appears at most two times in each genome?

This question was finally settled in the affirmative by Blin et al. in 2009:

Theorem 1 (Blin, Fertin, Sikora, and Vialette, 2009 [3]). Zero Exemplar Distance *for monochromosomal genomes is NP-hard even if each gene appears at most two times in each genome.*

In Section 2, we give a very simple alternative proof of this theorem.

Both the previous proof of Theorem 1 [3] and our alternative proof depend crucially on the order of the genes in the chromosomes. One may naturally wonder whether the complexity of Zero Exemplar Distance would change if gene order is not known. Note that genome rearrangement distances such as the syntenic distance [8] can be defined in the absence of gene order.

Now model each chromosome as a set of unsigned integers instead of a sequence of signed integers. Then Zero Exemplar Distance for multichromosomal genomes without gene order is the following problem: Given two collections G_1 and G_2 of subsets of the same ground set S of unsigned integers, decide whether both G_1 and G_2 can be reduced, by deleting elements from subsets and deleting subsets from collections, to the same collection G' of subsets of S such that each unsigned integer in S is contained in exactly one subset in G', i.e., G' is a partition of S. For example,

$$\begin{array}{lllll} G_1: & \{1,2,3\} & \{2,3,4\} & \{4,5\} & \\ G_2: & \{1,2\} & \{2,3,4\} & \{3,4,5\} & \{1,5\} \\ G': & \{1,2\} & \{3\} & \{4,5\} & \end{array}$$

In Section 3, we prove the following theorem:

Theorem 2. Zero Exemplar Distance *for multichromosomal genomes without gene order is NP-hard even if each gene appears at most two times in each genome. On the other hand, this problem admits a polynomial-time algorithm if only one of the two genomes has duplicate genes, and is fixed-parameter tractable if the parameter is the maximum number of chromosomes in each genome.*

As decision problems, Zero Exemplar Distance for monochromosomal genomes and for multichromosomal genomes without gene order are clearly in NP. Thus, following the NP-hardness results in Theorem 1 and Theorem 2, these two decision problems are both NP-complete. Moreover, the NP-hardness results in Theorem 1 and Theorem 2 imply that unless NP = P, the corresponding minimization problems of computing the exemplar distance between two genomes do not admit *any* approximation. We refer to [5,6,4,2,1] for related results.

2 Alternative Proof of Theorem 1

We prove that Zero Exemplar Distance for monochromosomal genomes is NP-hard by a reduction from the well-known NP-complete problem 3SAT [9]. Let (V, E) be a 3SAT instance, where $V = \{v_1, \ldots, v_n\}$ is a set of n boolean variables, $E = \{e_1, \ldots, e_m\}$ is a conjunctive boolean formula of m clauses, and each clause in E is a disjunction of exactly three literals of the variables in V. We will construct two sequences (genomes) G_1 and G_2 over $2n + 6m + 1$ distinct unsigned integers (genes):

variable genes Two genes x_i, y_i for each variable v_i, $1 \le i \le n$;
clause genes Three genes a_j, b_j, c_j for each clause e_j, $1 \le j \le m$;
literal genes Three genes r_j, s_j, t_j for the three literals of each clause e_j, $1 \le j \le m$;
separator gene One gene z.

In our construction, all genes appear in the positive orientation in the two genomes, so we will omit the signs in our description. The two genomes G_1 and G_2 are represented schematically as follows:

$$\begin{aligned} G_1: &\quad \langle v_1\rangle \dots \langle v_n\rangle \; z \; \langle e_1\rangle \dots \langle e_m\rangle \\ G_2: &\quad \langle v_1\rangle \dots \langle v_n\rangle \; z \; \langle e_1\rangle \dots \langle e_m\rangle \end{aligned}$$

For each variable v_i, the variable gadget $\langle v_i\rangle$ consists of one copy of x_i and two copies of y_i in G_1, two copies of x_i and one copy of y_i in G_2, and, for each literal of the variable in the clauses, one copy of the corresponding literal gene (r_j, s_j, or t_j for some clause e_j) in each genome. Let $p_{i,1}, \dots, p_{i,k_i}$ be the literal genes for the positive literals of v_i, and let $q_{i,1}, \dots, q_{i,l_i}$ be the literal genes for the negative literals of v_i. The genes $x_i, y_i, p_{i,1}, \dots, p_{i,k_i}, q_{i,1}, \dots, q_{i,l_i}$ in the variable gadget $\langle v_i\rangle$ are arranged in the following pattern in the two genomes:

$$\begin{aligned} G_1\langle v_i\rangle: &\quad y_i \; p_{i,1} \dots p_{i,k_i} \; x_i \; q_{i,1} \dots q_{i,l_i} \; y_i \\ G_2\langle v_i\rangle: &\quad p_{i,1} \dots p_{i,k_i} \; x_i \, y_i \, x_i \; q_{i,1} \dots q_{i,l_i} \end{aligned}$$

For each clause e_j, the clause gadget $\langle e_j\rangle$ consists of two copies of each clause gene a_j, b_j, c_j and one copy of each literal gene r_j, s_j, t_j. These genes in $\langle e_j\rangle$ are arranged in the following pattern in the two genomes:

$$\begin{aligned} G_1\langle e_j\rangle: &\quad r_j \, a_j \, b_j \, c_j \, s_j \, a_j \, b_j \, c_j \, t_j \\ G_2\langle e_j\rangle: &\quad a_j \, r_j \, b_j \, a_j \, s_j \, c_j \, b_j \, t_j \, c_j \end{aligned}$$

This completes the construction. It is easy to check that each gene appears at most two times in each genome, and that each genome includes exactly $3n + 12m + 1$ genes including duplicates. We give an example:

Example 1. For a 3SAT instance of 4 variables and 2 clauses $e_1 = \{r_1 = v_1, s_1 = \neg v_2, t_1 = \neg v_3\}$ and $e_2 = \{r_2 = \neg v_1, s_2 = v_3, t_2 = v_4\}$, the reduction constructs the following two genomes:

$$\begin{aligned} G_1: &\quad y_1 r_1 x_1 r_2 y_1 \quad y_2 x_2 s_1 y_2 \quad y_3 s_2 x_3 t_1 y_3 \quad y_4 t_2 x_4 y_4 \\ &\quad z \quad r_1 a_1 b_1 c_1 s_1 a_1 b_1 c_1 t_1 \quad r_2 a_2 b_2 c_2 s_2 a_2 b_2 c_2 t_2 \\ G_2: &\quad r_1 x_1 y_1 x_1 r_2 \quad x_2 y_2 x_2 s_1 \quad s_2 x_3 y_3 x_3 t_1 \quad t_2 x_4 y_4 x_4 \\ &\quad z \quad a_1 r_1 b_1 a_1 s_1 c_1 b_1 t_1 c_1 \quad a_2 r_2 b_2 a_2 s_2 c_2 b_2 t_2 c_2 \end{aligned}$$

The assignment $v_1 = \text{true}, v_2 = \text{false}, v_3 = \text{false}, v_4 = \text{true}$ satisfies the 3SAT instance and corresponds to the following common reduced genome:

$$G': \quad r_1 x_1 y_1 \quad y_2 x_2 s_1 \quad y_3 x_3 t_1 \quad t_2 x_4 y_4 \quad z \quad a_1 b_1 c_1 \quad r_2 a_2 s_2 b_2 c_2$$

The reduction clearly runs in polynomial time. It remains to prove the following lemma:

Lemma 1. *The 3SAT instance (V, E) is satisfiable if and only if the two genomes G_1 and G_2 have a common subsequence G' including exactly one copy of each gene.*

We first prove the direct implication. Suppose that the 3SAT instance (V, E) is satisfiable. We will compose a common subsequence G' of the two genomes G_1 and G_2 from a common subsequence of each variable gadget $\langle v_i \rangle$, the separator gene z in the middle, and a common subsequence of each clause gadget $\langle e_j \rangle$. Consider a truth assignment that satisfies the 3SAT instance. For each variable v_i, take the subsequence $p_{i,1} \dots p_{i,k_i}\, x_i y_i$ if v_i is set to true, and take the subsequence $y_i x_i\, q_{i,1} \dots q_{i,l_i}$ if v_i is set to false. For each clause e_j, at least one of its three literals is true; correspondingly, at least one of the three literal genes r_j, s_j, t_j has been taken from some variable gadget $\langle v_i \rangle$. Now take a subsequence from the clause gadget $\langle e_j \rangle$ following one of three cases:

1. If r_j has been taken, then take the subsequence $a_j b_j \underline{s_j} c_j \underline{t_j}$.
2. If s_j has been taken, then take either the subsequence $\underline{r_j} b_j a_j c_j \underline{t_j}$ or the subsequence $\underline{r_j} a_j c_j b_j \underline{t_j}$.
3. If t_j has been taken, then take the subsequence $\underline{r_j} a_j \underline{s_j} b_j c_j$.

Here an underlined literal gene is omitted from the subsequence taken from the clause gadget $\langle e_j \rangle$ if its other copy has already been taken from some variable gadget $\langle v_i \rangle$. The common subsequence G' thus composed clearly includes exactly one copy of each gene.

We next prove the reverse implication. Suppose that the two genomes G_1 and G_2 have a common subsequence G' including exactly one copy of each gene. We will find a satisfying assignment for the 3SAT instance (V, E) as follows. Due to the strategic location of the separator gene z in the two genomes, each literal gene must appear in the common subsequence either before z in both genomes, in some variable gadget $\langle v_i \rangle$, or after z in both genomes, in some clause gadget $\langle e_j \rangle$. The crucial property of the clause gadget $\langle e_j \rangle$ is that it cannot have a common subsequence including exactly one copy of each clause gene a_j, b_j, c_j unless at least one of the three literal genes r_j, s_j, t_j is omitted. A literal gene omitted from the common subsequence of the clause gadget $\langle e_j \rangle$ has to appear in the common subsequence of some variable gadget $\langle v_i \rangle$, where the two variable genes x_i and y_i must appear in the order $x_i y_i$ if the literal is positive and appear in the order $y_i x_i$ if the literal is negative. Now set each variable v_i to true if the two variable genes x_i and y_i appear in the common subsequence G' in the order $x_i y_i$, and set it to false otherwise. Then each clause gets at least one true literal. This completes the proof.

3 Proof of Theorem 2

We prove that Zero Exemplar Distance for multichromosomal genomes without gene order is NP-hard by a reduction again from 3SAT. Let (V, E) be a 3SAT instance, where $V = \{v_1, \dots, v_n\}$ is a set of n boolean variables, $E = \{e_1, \dots, e_m\}$ is a conjunctive boolean formula of m clauses, and each clause in E is a disjunction of exactly three literals of the variables in V. Without loss of generality, assume that no clause in E contains two literals of the same variable in V. We will construct two genomes G_1 and G_2 over $n + 9m$ distinct genes:

variable genes. One gene x_i for each variable v_i, $1 \leq i \leq n$;
clause genes. Six genes $a_j, b_j, c_j, a'_j, b'_j, c'_j$ for each clause e_j, $1 \leq j \leq m$;
literal genes. Three genes r_j, s_j, t_j for the three literals of each clause e_j, $1 \leq j \leq m$.

For each variable v_i, let $p_{i,1}, \ldots, p_{i,k_i}$ be the literal genes for the positive literals of v_i, and let $q_{i,1}, \ldots, q_{i,l_i}$ be the literal genes for the negative literals of v_i. G_1 includes one subset and G_2 includes two subsets of genes including x_i:

$$\begin{aligned} G_1\langle v_i\rangle: &\quad \{p_{i,1}, \ldots, p_{i,k_i}, x_i, q_{i,1}, \ldots, q_{i,l_i}\} \\ G_2\langle v_i\rangle: &\quad \{p_{i,1}, \ldots, p_{i,k_i}, x_i\} \quad \{x_i, q_{i,1}, \ldots, q_{i,l_i}\} \end{aligned}$$

For each clause e_j, G_1 includes six subsets and G_2 includes seven subsets of clause/literal genes:

$$\begin{aligned} G_1\langle e_j\rangle: &\quad \{a_j, b_j\}\,\{b_j, c_j\}\,\{c_j, a_j\} \quad \{a'_j, r_j\}\,\{b'_j, s_j\}\,\{c'_j, t_j\} \\ G_2\langle e_j\rangle: &\quad \{a_j, b_j, c_j\} \quad \{a_j, a'_j, r_j\}\,\{b_j, b'_j, s_j\}\,\{c_j, c'_j, t_j\} \quad \{a'_j\}\,\{b'_j\}\,\{c'_j\} \end{aligned}$$

This completes the construction. It is easy to check that each gene appears at most two times in each genome, G_1 includes exactly $n + 15m$ genes including duplicates, and G_2 includes exactly $2n + 18m$ genes including duplicates. We give an example:

Example 2. For a 3SAT instance of 4 variables and 2 clauses $e_1 = \{r_1 = v_1, s_1 = \neg v_2, t_1 = \neg v_3\}$ and $e_2 = \{r_2 = \neg v_1, s_2 = v_3, t_2 = v_4\}$, the reduction constructs the following two genomes:

$$\begin{aligned} G_1: &\quad \{r_1, x_1, r_2\} \quad \{x_2, s_1\} \quad \{s_2, x_3, t_1\} \quad \{t_2, x_4\} \\ &\quad \{a_1, b_1\}\,\{b_1, c_1\}\,\{c_1, a_1\} \quad \{a'_1, r_1\}\,\{b'_1, s_1\}\,\{c'_1, t_1\} \\ &\quad \{a_2, b_2\}\,\{b_2, c_2\}\,\{c_2, a_2\} \quad \{a'_2, r_2\}\,\{b'_2, s_2\}\,\{c'_2, t_2\} \\ G_2: &\quad \{r_1, x_1\}\,\{x_1, r_2\} \quad \{x_2\}\,\{x_2, s_1\} \quad \{s_2, x_3\}\,\{x_3, t_1\} \quad \{t_2, x_4\}\,\{x_4\} \\ &\quad \{a_1, b_1, c_1\} \quad \{a_1, a'_1, r_1\}\,\{b_1, b'_1, s_1\}\,\{c_1, c'_1, t_1\} \quad \{a'_1\}\,\{b'_1\}\,\{c'_1\} \\ &\quad \{a_2, b_2, c_2\} \quad \{a_2, a'_2, r_2\}\,\{b_2, b'_2, s_2\}\,\{c_2, c'_2, t_2\} \quad \{a'_2\}\,\{b'_2\}\,\{c'_2\} \end{aligned}$$

The assignment $v_1 = \text{true}, v_2 = \text{false}, v_3 = \text{false}, v_4 = \text{true}$ satisfies the 3SAT instance and corresponds to the following common reduced genome:

$$\begin{aligned} G': &\quad \{r_1, x_1\} \quad \{x_2, s_1\} \quad \{x_3, t_1\} \quad \{t_2, x_4\} \\ &\quad \{a_1\}\,\{b_1, c_1\} \quad \{a'_1\}\,\{b'_1\}\,\{c'_1\} \\ &\quad \{c_2\}\,\{a_2, b_2\} \quad \{a'_2, r_2\}\,\{b'_2, s_2\}\,\{c'_2\} \end{aligned}$$

The reduction clearly runs in polynomial time. It remains to prove the following lemma:

Lemma 2. *The 3SAT instance (V, E) is satisfiable if and only if the two genomes G_1 and G_2 have a common reduced genome G' including exactly one copy of each gene.*

We first prove the direct implication. Suppose that the 3SAT instance (V, E) is satisfiable. We will compose a common reduced genome G' of the two genomes G_1 and G_2 as follows. Consider a truth assignment that satisfies the 3SAT instance. For each variable v_i, take the subset $\{p_{i,1}, \ldots, p_{i,k_i}, x_i\}$ if v_i is set to true, and take the subset $\{x_i, q_{i,1}, \ldots, q_{i,l_i}\}$ if v_i is set to false. For each clause e_j, at least one of its three literals is true; correspondingly, at least one of the three literal genes r_j, s_j, t_j has been taken from a subset that contains some variable gene x_i. Now take some subsets of clause/literal genes following one of three cases:

1. If r_j has been taken, then take the subsets $\{a_j\}, \{b_j, c_j\}, \{a'_j\}, \{b'_j, \underline{s_j}\}, \{c'_j, \underline{t_j}\}$.
2. If s_j has been taken, then take the subsets $\{b_j\}, \{c_j, a_j\}, \{a'_j, \underline{r_j}\}, \{\overline{b'_j}\}, \{c'_j, \underline{t_j}\}$.
3. If t_j has been taken, then take the subsets $\{c_j\}, \{a_j, b_j\}, \{a'_j, \underline{r_j}\}, \{b'_j, \underline{s_j}\}, \{\overline{c'_j}\}$.

Here an underlined literal gene is omitted from the subset taken from the clause gadget $\langle e_j \rangle$ if its other copy has already been taken from a subset that contains some variable gene x_i. The reduced genome G' thus composed clearly includes exactly one copy of each gene.

We next prove the reverse implication. Suppose that the two genomes G_1 and G_2 have a common reduced genome G' including exactly one copy of each gene. We will find a satisfying assignment for the 3SAT instance (V, E) as follows. The crucial property of the clause gadget $\langle e_j \rangle$ is that it cannot have a common reduced genome including exactly one copy of each clause gene $a_j, b_j, c_j, a'_j, b'_j, c'_j$ unless at least one of the three literal genes r_j, s_j, t_j is omitted. A literal gene omitted from the clause gadget $\langle e_j \rangle$ has to appear in a subset in G' that contains some variable gene x_i. By the construction of the variable gadgets, this subset contains, besides x_i, either literal genes for positive literals, or literal genes for negative literals. Now set each variable v_i to true if the subset in G' that contains x_i also contains at least one literal gene for a positive literal, and set it to false otherwise. Then each clause gets at least one true literal. This completes the NP-hardness proof.

3.1 Two Algorithms

We present two algorithms for Zero Exemplar Distance for multichromosomal genomes without gene order. Let k_1 and k_2, respectively, be the numbers of chromosomes in G_1 and G_2. Let $A_1, \ldots, A_{k_1}$ be the k_1 chromosomes in G_1. Let $B_1, \ldots, B_{k_2}$ be the k_2 chromosomes in G_2. Let $k = \max\{k_1, k_2\}$. Let n be the total number of genes in G_1 and G_2, i.e., $n = \sum_{i=1}^{k_1} |A_i| + \sum_{j=1}^{k_2} |B_j|$.

We first present a polynomial-time algorithm for a special case of the problem in which only one of the two genomes has duplicate genes. Assume without loss of generality that G_1 has no duplicate genes. Then the problem is simply deciding whether G_1 can be obtained from G_2 by deleting genes from chromosomes and deleting chromosomes altogether, which is possible if and only if each chromosome in G_1 is a subset of a distinct chromosome in G_2. This leads to the following simple algorithm based on matching in bipartite graphs:

Algorithm A1

1. Construct a bipartite graph $G = (V_1 \cup V_2, E)$ with vertices $V_1 = \{A_1, \ldots, A_{k_1}\}$ and $V_2 = \{B_1, \ldots, B_{k_2}\}$, and with an edge between $A_i \in V_1$ and $B_j \in V_2$ if and only if $A_i \subseteq B_j$.
2. Compute a maximum-cardinality matching M in the graph G.
3. If the cardinality of M is equal to k_1, return yes. Otherwise, return no.

The correctness of Algorithm A1 is obvious. We now analyze its time complexity. Step 1 can be easily implemented in $O(n^2)$ time. The resulting bipartite graph has $k_1 + k_2 \leq 2\max\{k_1, k_2\} = 2k$ vertices. Using the standard Hopcroft-Karp algorithm for bipartite matching [10], step 2 can be implemented in $O(k^{5/2})$ time. Thus the overall time complexity is $O(n^2 + k^{5/2})$, which is polynomial.

We next present a fixed-parameter tractable algorithm for this problem without any assumption on the distribution of duplicate genes. Refer to [7] for basic concepts in parameterized complexity theory. The parameter of our algorithm is $k = \max\{k_1, k_2\}$:

Algorithm A2

1. Add $k - k_1$ empty chromosomes $A_{k_1+1}, \ldots, A_k$ to G_1, or add $k - k_2$ empty chromosomes $B_{k_2+1}, \ldots, B_k$ to G_2, such that G_1 and G_2 have the same number k of chromosomes.
2. For each permutation π of $\langle 1, \ldots, k\rangle$, compute $C_\pi = \cup_{i=1}^{k}(A_i \cap B_{\pi(i)})$.
3. If for some permutation π the set C_π includes all the genes, return yes. Otherwise return no.

To see the correctness of Algorithm A2, note that each chromosome of the common reduced genome (if it exists) is obtained from each input genome as a subset of a distinct chromosome. This gives rise to a match between two chromosomes from the two input genomes. All other unmatched chromosomes do not contribute to the common reduced genome and are deleted. To handle the matching and the deletion of the chromosomes in a uniform way, we can think of each chromosome deleted from one genome as matched to a chromosome deleted from the other genome or to an empty chromosome. Thus by padding the two genomes to the same number of chromosomes, we only need to consider perfect matchings as permutations. The overall time complexity of Algorithm A2 is $O(k!\, n^2)$, with $O(n^2)$ time for each of the $k!$ permutations.

We remark that the problem Zero Exemplar Distance for multichromosomal genomes without gene order is unlikely to have a fixed-parameter tractable algorithm if the parameter is the maximum number of genes in any single chromosome. This is because 3SAT remains NP-hard even if for each variable there are at most five clauses that contain its literals [9]. As a result, the number of genes in each chromosome need not be more than some constant in our reduction from 3SAT.

Acknowledgment. The author would like to thank Binhai Zhu for a brief discussion on Question 1 during a visit to University of Texas - Pan American in

May 2008, and thank Guillaume Fertin for communicating the recent result [3]. The alternative proof of Theorem 1 was obtained independently by the author in February 2010 without knowing the recent progress [3].

References

1. Adi, S.S., Braga, M.D.V., Fernandes, C.G., Ferreira, C.E., Martinez, F.V., Sagot, M.-F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsequence. Discrete Applied Mathematics 158, 1315–1324 (2010)
2. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. Journal of Graph Algorithms and Applications 13, 19–53 (2009)
3. Blin, G., Fertin, G., Sikora, F., Vialette, S.: The Exemplar Breakpoint Distance for non-trivial genomes cannot be approximated. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 357–368. Springer, Heidelberg (2009)
4. Bonizzoni, P., Della Vedova, G., Dondi, R., Fertin, G., Rizzi, R., Vialette, S.: Exemplar longest common subsequence. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4, 535–543 (2007)
5. Chen, Z., Fowler, R.H., Fu, B., Zhu, B.: On the inapproximability of the exemplar conserved interval distance problem of genomes. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 245–254. Springer, Heidelberg (2006)
6. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 291–302. Springer, Heidelberg (2006)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
8. Ferretti, V., Nadeau, J.H., Sankoff, D.: Original synteny. In: Hirschberg, D.S., Meyers, G. (eds.) CPM 1996. LNCS, vol. 1075, pp. 159–167. Springer, Heidelberg (1996)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
10. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal on Computing 2, 225–231 (1973)
11. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics 15, 909–917 (1999)

Scaffold Filling under the Breakpoint Distance

Haitao Jiang[1,2], Chunfang Zheng[3], David Sankoff[4], and Binhai Zhu[1]

[1] Department of Computer Science, Montana State University,
Bozeman, MT 59717-3880, USA
htjiang,bhz@cs.montana.edu
[2] School of Computer Science and Technology, Shandong University, Jinan, China
[3] Départment d'informatique et de recherche opérationnelle,
Unive rsité de Montréal, Montréal, Canada H3C 3J7
chunfang313@gmail.com
[4] Department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada KIN 6N5
sankoff@uottawa.ca

Abstract. Motivated by the trend of genome sequencing without completing the sequence of the whole genomes, Muñoz et al. recently studied the problem of filling an incomplete multichromosomal genome (or *scaffold*) I with respect to a complete target genome G such that the resulting genomic distance between I' and G is minimized, where I' is the corresponding filled scaffold. We call this problem the *one-sided* scaffold filling problem. In this paper, we follow Muñoz et al. to investigate the scaffold filling problem under the breakpoint distance for the simplest unichromosomal genomes. When the input genome contains no gene repetition (i.e., is a fragment of a permutation), we show that the *two-sided* scaffold filling problem is polynomially solvable. However, when the input genome contains some genes which appear twice, even the one-sided scaffold filling problem becomes NP-complete. Finally, using the ideas for solving the two-sided scaffold filling problem under the breakpoint distance we show that the two-sided scaffold filling problem under the genomic/rearrangement distance is also polynomially solvable.

1 Introduction

Due to the advancement of genome sequencing technology, it is possible to sequence more organisms for genomic analysis. (Throughout this paper, a multichromosomal genome is represented as sequences of genes, while a unichromosomal genome is just represented as a sequence of genes.) Interesting and somehow contradicting, the cost of finishing genome sequencing has not decreased at the same rate compared with the cost of random sequencing [1]. This means that many genomes released are not completely finished. It would be unsuitable to use these incomplete genomes (scaffolds) for genomic analysis, simply due to the errors they could introduce.

Therefore, a natural problem is to fill the missing genes into scaffolds, with combinatorial algorithms. As one must find a biologically meaningful way of filling scaffolds, it makes sense to make use of some complete genomes (from some close species). Muñoz et al. [10] recently carried out this idea on filling an incomplete multichromosomal scaffold I to have I', such that the genomic distance [13] between I' and a given (complete)

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 83–92, 2010.

genome G is minimized. (The genomic distance is also called *rearrangement* distance, which is the minimum number of allowed rearrangement operations transforming one genome into the other.) We call this the *one-sided* scaffold filling problem. Basically, the one-sided scaffold filling can be solved in polynomial time; in fact, linear time when the breakpoint graph on I and G is constructed [10].

In [10], much effort has been put on several practical issues. For instance, what if the missing genes can only be inserted in certain locations? What if some missing genes in I are not really missing (i.e., they should not appear in G)? However, the corresponding two-sided problem is not tackled in [10].

In this paper, we follow Muñoz et al. to investigate the scaffold filling problem under the breakpoint distance for the simplest unichromosomal genomes. When the input genome contains no gene repetition (i.e., is a fragment of a permutation), we show that the *two-sided* scaffold filling problem is polynomially solvable. However, when the input genome contains some genes which appear twice, even the one-sided scaffold filling problem becomes NP-complete. The latter problem has a close connection with the Minimum Common String Partition (MCSP) problem [2,3,4,7,8,9].

This paper is organized as follows. In Section 2, we give necessary definitions. In Section 3, we present the polynomial time algorithm for the scaffold filling problem. In Section 4, we show the NP-completeness proof for the one-sided scaffold filling problem when gene duplications are allowed. In Section 5, we show how to adapt our ideas in Section 3 to solve the two-sided scaffold filling problem under the rearrangement distance (i.e., for multichromosomal genomes) in polynomial time. In Section 6, we conclude the paper.

2 Preliminaries

We first present some necessary definitions.

Given alphabet Σ, a string S is called a *permutation* if each element in Σ appears exactly once in S. We also use $c(S) = \Sigma$ to denote the set of elements in permutation S. An (unsigned) unichromosomal genome is just a permutation over Σ.

A *scaffold* is an incomplete permutation, i.e., with some missing elements. We use $+$ to denote permutation scaffold filling, e.g., for a permutation A and an element set X such that $c(A) \cap X = \emptyset$, if A^* is a resulting permutation after filling all the elements in X into A, then $A^* = A + X$. Similarly, we use $-$ to denote element elimination from the permutation. Given two permutations A and B, if $c(A) = c(B)$, then A and B are *related*. Given two related permutations A and B, two consecutive elements a_i and a_{i+1} in A form an *adjacency* if they are also consecutive in B (i.e., as $a_i a_{i+1}$ or $a_{i+1} a_i$), otherwise they form a *breakpoint*. The number of breakpoints in A, which is equal to that of B, is the breakpoint distance between A and B, denoted as $bd(A, B)$. Note that our breakpoint definition and the corresponding results all work when the letters (or genes) are possibly *signed*.

The (*two-sided*) scaffold filling problem is defined as follows.

Scaffold Filling under the Minimum Permutation Breakpoint Distance (SF-PBD)
Input: two incomplete permutations A and B and two sets of elements X and Y, where $X = c(B) - c(A)$ and $Y = c(A) - c(B)$.

Question: minimize $bd(A + X, B + Y)$.

In the above definition, when either X or Y is empty, we have the *one-sided* scaffold filling problem. Note that if A and B were related (i.e., $c(A) = c(B)$), then we would have $X = Y = \emptyset$.

In practice, sometimes we need to deal with genomes with orthologous (duplicated) genes. Let $C(S)$ be a multiset to denote all the appearances of all the elements. We still use $+$ to denote string scaffold filling.

Scaffold Filling under the Minimum String Breakpoint Distance (SF-SBD)
Input: two strings A and B and two multisets of elements X and Y where $X = c(B) - c(A)$ and $Y = c(A) - c(B)$.
Question: minimize $bd(A + X, B + Y)$.

For this problem, when some of the genes can appear more than once, we will show that even the one-sided scaffold filling problem is NP-complete. This problem has a close connection with the Minimum Common String Partition problem. We present the details of our results under the breakpoint distance in the next two sections.

3 Polynomial Algorithm for SF-PBD

In this section we present a polynomial algorithm for scaffold filling under the permutation breakpoint distance.

Lemma 1. *Given two incomplete permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. If there is an adjacency $a_i a_{i+1}$ in the two related permutations $A - Y$ and $B - X$, then there exists a scaffold filling such that every two consecutive elements between a_i and a_{i+1} (in $A + X$ and $B + Y$) form an adjacency.*

Proof. To obtain $A+X$ and $B+Y$, we just need to insert the elements in X into A and insert the elements in Y into B respectively. As $a_i a_{i+1}$ is an adjacency in $A - Y$ and $B - X$, we have the full freedom to insert the respective elements in $X + Y$ in between a_i and a_{i+1} such that they all form adjacencies in $A + X$ and $B + Y$. □

Lemma 2. *Given two permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. If there is a breakpoint $b_i b_{i+1}$ in $A - Y$, then in any scaffold filling, there is at least one breakpoint between b_i and b_{i+1} in $A + X$.*

Proof. As shown in the previous lemma, we insert the respective elements in $X + Y$ into $A-Y$ to obtain $A+X$. When we insert some respective elements in between b_i and b_{i+1}, if these inserted elements contain a breakpoint then the lemma is proven. If the inserted elements contain no breakpoint, then they must introduce at least a breakpoint right before b_{i+1} or right after b_i. □

Lemma 3. *Given two permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. If there is a breakpoint $b_i b_{i+1}$ in $A - Y$, then there exists a scaffold filling such that there is only one breakpoint between b_i and b_{i+1} in $A + X$.*

Proof. As shown in the previous lemma, we just insert the respective elements in $X+Y$ in between b_i and b_{i+1}. If the breakpoint in $B-X$ has one letter in agreement with b_i or b_{i+1}, say b_ib', then we just make sure that the inserted elements contain no breakpoint, the only breakpoint hence introduced is right before b_{i+1} (or right after b_i in $A+X$, if the corresponding breakpoint in $B-X$ has the form of $b'b_{i+1}$). □

Theorem 1. *Given two permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. Then $bd(A-Y, B-X) = bd(A+X, B+Y)$ and the corresponding $A+X$ and $B+Y$ can be computed in $O(n^2)$ time.*

Proof. Following the previous lemmas, it is easy to see that $bd(A-Y, B-X) = bd(A+X, B+Y)$. The corresponding algorithm is as follows. First, we identify $A-Y$ and $B-X$, and compute the breakpoint distance $bd(A-Y, B-X)$. Second, we insert the respective elements from Y into $A-Y$ and $B-X$ such that the number of breakpoints (in A and $B-X+Y$) does not change. Then, we insert the respective elements from X into A and $B-X+Y$, still maintaining the number of breakpoints. Eventually, we obtain $A+X$ and $B+Y$ accordingly. The quadratic running time is dominated by finding the correspondence between the identical characters in A and B. □

An example of the above algorithm is as follows.

$$A = acefh, B = adbhge$$

$$X = \{d, b, g\}, Y = \{c, f\}$$

$$A - Y = aeh, B - X = ahe, bd(A - Y, B - X) = 1$$

$$A + X = acdbegfh, B + Y = acdbhfge, bd(A + X, B + Y) = 1.$$

We comment that our algorithm also works when in A and B there are predefined adjacencies one could not break, as long as these adjacencies have no conflict. For example, we have $A = \cdots \boxed{\text{ac}} \cdots \boxed{\text{gh}} \cdots$ and $B = \cdots \boxed{\text{ea}} \cdots \boxed{\text{fg}} \cdots$, with predefined adjacencies in boxes. When we fill e, f into A and c, g into B, the algorithm still works as the predefined adjacencies are not in conflict. However, if $A = \cdots \boxed{\text{bac}} \cdots \boxed{\text{ghf}} \cdots$ and $B = \cdots \boxed{\text{bae}} \cdots \boxed{\text{gfh}} \cdots$, then our result does not hold anymore. In [10], this is related to insert missing genes only in between contigs (i.e., not anywhere), which is a problem needing further study.

4 Hardness of SF-SBD

In this section, we prove that SF-SBD is NP-complete; in fact, even the one-sided scaffold filling problem is NP-complete, when each gene is allowed to appear at most twice.

We make a reduction from the maximum independent set problem on cubic graphs (3-MIS). The main idea of this proof is from the NP-hardness proof by Goldstein *et al.* for 2-MCSP (Minimum Common String Partition with each letter appears at most

twice in an input string) [7]. We try to add more separators and adapt the proof for our purposes. For completeness, we describe the MCSP problem as follows.

Define that two strings A and B are related if each element appears the same number of times in A and B. A partition of a string A is a sequence $P = (P_1, P_2, \ldots, P_m)$ of strings, called the blocks, whose concatenation is equal to A. Given a partition P of a string A and a partition Q of a string B, we say that the pair (P, Q) is a common partition of A and B if Q is a permutation of P. A minimum common string partition is a common partition of two strings A and B with the minimum number of blocks. Two related strings always have a minimum common string partition. Note that the breakpoint distance of two related strings is equal to the size of minimum common string partition minus one. An example is as follows. $A = 01101110111, B = 01110011111$, the three optimal blocks are $011, 01110, 111$.

Given a cubic graph $G = (V, E)$ as an input for 3-MIS, for each vertex $v \in V$, we create two substrings A_v and B_v.

$A_v =$
$d_v x_{v_1} y_{v_2} a_v b_v x_{v_2} y_{v_3} c_v d_v e_v x_{v_3} y_{v_4} b_v e_v z_v f_v g_v x_{v_4} y_{v_5} f_v h_v k_v x_{v_5} y_{v_6} g_v l_v x_{v_6} y_{v_1} h_v$
$B_v =$
$b_v y_{v_1} x_{v_1} c_v d_v y_{v_2} x_{v_2} a_v b_v e_v y_{v_3} x_{v_3} d_v e_v f_v h_v y_{v_4} x_{v_4} f_v g_v l_v y_{v_5} x_{v_5} h_v k_v y_{v_6} x_{v_6} g_v$

In both A_v and B_v, $x_{v_i} y_{v_{(i+1) mod\ 6}}$ for $1 \leq i \leq 5$, $x_{v_6} y_{v_1}$, and $y_{v_i} x_{v_i}$ for $1 \leq i \leq 6$ are separators and are not adjacencies following our definition of breakpoints. A_v and B_v contain 29 and 28 letters respectively, with z_v not appearing in B_v. Among other letters, a_v, c_v, k_v and l_v only appear once in A_v.

For each edge $(u, v) \in E$, we locally modify A_u, B_u, A_v and B_v. Before that a few terms will be needed. The letters a_v and c_v in A_v are called the left sockets of A_v and the letters k_v and l_v in A_v are the right sockets. Initially, all sockets are free.

As in [7], we orient the edges of G in such a way that each vertex has at most two incoming edges and at most two outgoing edges. This can be done as follows: find a maximal set (with respect to inclusion) of edge-disjoint cycles in G, and in each cycle, orient the edges to form a directed cycle. The remaining edges form a forest. For each tree in the forest, choose one of its nodes of degree one to be the root, and orient all edges in the tree away from the root. This orientation will clearly satisfy the desired properties.

Consider an edge $\overrightarrow{(u, v)}$ and a free right socket s_u of A_u and a free left socket s_v of A_v. Define $R = f_u h_u$ if $s_u = k_u$, and $R = g_u$ if $s_u = l_u$; $S = d_v e_v$ if $s_v = c_v$, and $S = b_v$ if $s_v = a_v$. We do the following modifications:

(1) insert S to the immediate right of s_u in A_u; in other words, change Rs_u into $Rs_u S$;

(2) replace $s_v S$ with s_u in A_v; in other words, change $s_v S$ into s_u;

(3) replace s_v with s_u in B_v.

(4) B_u is unchanged.

After this modification process, all relevant sockets will become *non-free* or *used*.

The final instance of SF-SBD is composed of strings A, B and a set Z, where $A = \bigcup_{v \in V} A_v p_v q_v r_v w_v$, $B = \bigcup_{v \in V} B_v r_v p_v w_v q_v$, and $Z = \{z_v | v \in V\}$. More precisely,

A and B can be considered as a concatenation of $A_v p_v q_v r_v w_v$'s and $B_v r_v p_v w_v q_v$'s with orders insignificant. We need to insert the elements in Z into B.

The reduction can be completed with the following lemmas.

Lemma 4. *If there is an edge $(u, v) \in E$, after the above modification, A_u and A_v cannot both contain 6 adjacencies with respect to B.*

Proof. If A_u contains 6 adjacencies with respect to B, they must be

$$\{(a_u b_u), (c_u d_u), (e_u z_u), (z_u f_u), (h_u k_u), (g_u l_u)\}.$$

If $R = g_u$ and $S = b_v$, after the modification $(g_u l_u)$ and $(l_u b_v)$ cannot be an adjacency at the same time. (The claim is also true when $R = g_u$ and $S = d_v e_v$.) Symmetrically, if $R = f_u h_u$ and $S = b_v$, after the modification $(h_u k_u)$ and $(k_u b_v)$ cannot be an adjacency at the same time. (The claim is also true when $R = f_u h_u$ and $S = d_v e_v$.) Therefore, as long as there is an edge between u and v, A_u and A_v cannot both contain 6 adjacencies with respect to B. □

Lemma 5. *Let G be a cubic graph on n vertices. There exists an independent set I of size k in G if and only if there exists a scaffold filling such that there are 5n+k adjacencies between A and $B + Z$.*

Proof. For a vertex $v \in V$, if $v \in I$ then there can be 6 adjacencies in A_v. This can be done as follows: insert z_v between e_v and f_v in B_v, i.e.,
$B_v + \{z_v\} =$
$b_v y_{v_1} x_{v_1} c_v d_v y_{v_2} x_{v_2} a_v b_v e_v y_{v_3} x_{v_3} d_v \underline{e_v z_v f_v} h_v y_{v_4} x_{v_4} f_v g_v l_v y_{v_5} x_{v_5} h_v k_v y_{v_6} x_{v_6} g_v$.

Otherwise, if v is not in I, there are at most 5 adjacencies in A_v, following Lemma 4. All the separators x_{v_i}, y_{v_i} and p_v, q_v, r_v and w_v cannot form an adjacency. Therefore, we have $6k + 5(n - k) = 5n + k$ adjacencies between A and $B + Z$.

The converse can be proved using a similar argument. If there are $5n + k$ adjacencies between A and $B + Z$, following Lemma 4, exactly $6k$ adjacencies are from some A_v where v is in an independent set. Consequently, there is an independent set of G with size k. □

As it is obvious that SF-SBD is in NP, with Lemma 5, we hence have the following theorem.

Theorem 2. *SF-SBD is NP-complete.*

5 Two-Sided Scaffold Filling under the Rearrangement Distance

In this section, we show how to adapt the ideas in Section 3 to solve the two-sided scaffold filling problem under the rearrangement distance, when the genomes are multichromosomal and the genes are signed.

Rearrangement Distance

The *rearrangement distance* or *genomic distance* $D(G_1, G_2)$ is a metric counting the number of genomic rearrangement operations necessary to transform one signed multichromosomal genome G_1 containing n distinct genes into another, G_2. The $+$ or $-$ sign indicates the "reading direction" of a gene, left-to-right or right-to-left.

For a comprehensive repertoire of operations, which we need not elaborate here, Yancopoulos *et al.* [13] showed that D could be calculated efficiently using the *breakpoint graph* [11] of G_1 and G_2 as follows:

1. Replace each positively-signed gene g on a chromosome by the vertex pair g_t, g_h; replace a negative $-g$ by g_h, g_t.
2. Each pair of successive genes in the genome defines one edge connecting the pair of vertices that are adjacent in the vertex order. E.g., if $i\ j\ -k$ are neighboring genes on a chromosome then the two edges they define are $\{i_h, j_t\}$ and $\{j_h, k_h\}$. This leaves two unconnected vertices at the ends of each chromosome. Define an edge incident to each such vertex in genome G_1 and G_2 connecting it to a new vertex, all labelled T_1 in G_1 and T_2 in G_2.
3. Color the edges of G_1 and G_2 blue and red, respectively.
4. Identify (i.e., superimpose) each vertex in G_1 with the identically labelled vertex in G_2.
5. Make a cycle of any path ending in two T_1 or two T_2 vertices, connecting them by a red or blue edge, respectively, while for a path ending in a T_1 and a T_2, collapse them to form one T vertex.
6. Each vertex is now incident to one blue and one red edge. This bicolored graph, the breakpoint graph, decomposes uniquely into κ alternating cycles. If n' is the number of blue edges, $D(G_1, G_2) = n' - \kappa$ (see [13]) and the optimizing rearrangements are rapidly recovered by operations on the graph.

Bundles

Now consider the case where the genes in G_2 are a subset of the genes in G_1. We say some genes are *missing* from G_2. The one-sided scaffold filling problem is to insert the missing genes in G_2, thus forming $\bar{G}_2$, in such a way as to minimize $D(G_1, \bar{G}_2)$. In [10], it was shown that the one-sided scaffold filling problem under the rearrangement distance can be solved in polynomial time, in fact, in linear time once the breakpoint graph is constructed.

We can still construct the breakpoint graph, except that some vertices, called *free ends*, will only be incident to a blue edge and thus paths in the graph can end not only in T vertices but also in free ends. When this happens, step 5 in the breakpoint graph construction cannot be completed, and the decomposition and calculation in step 6 are blocked.

A *bundle* is a subset of the paths in this partial breakpoint graph of G_1 and G_2. (Partial because some paths are not cycles nor do they end in a T.) Each bundle is associated with one or more of the missing genes. The vertices corresponding to each missing gene, its free ends, must be in the same bundle and must be endpoints of one

or two paths. To simplify the exposition, we assume that no bundle consists entirely of blue edges, i.e., no chromosome in G_1 has all its genes absent from G_2. This case is easily handled separately, and does not affect the distance calculations.

To construct a bundle, we initiate it with any path not already in any bundle and ending with a free end. Then if a path containing free end g_t is in a bundle B, then we also include the path with g_h as a free end, and vice versa. There can be zero or two T vertices in a bundle. We now present the details on the two-sided scaffold filling problem under the rearrangement distance. It is not hard to show the following lemma.

Lemma 6. *If genomes G_1 and G_2 both contain the same n genes, and if $m \geq 1$ genes are inserted anywhere into both G_1 and G_2 to get $\bar{G}_1$ and $\bar{G}_2$, then $D(\bar{G}_1, \bar{G}_2) \geq D(G_1, G_2)$.*

In a one-sided scaffold filling problem for G_1 and G_2, suppose there are r cycles in the partial breakpoint graph and suppose this graph determines β bundles. Let $\underline{G}_1$ be the genome formed by deleting from G_1 the genes already missing from G_2.

Lemma 7. *Let κ be the number of cycles in the breakpoint graph of $\underline{G}_1$ and G_2. Then $\beta = \kappa + r$.*

Proof. If a_t and a_h are a pair of free ends in a bundle, incident to blue edges (a_t, x) and (a_h, y), remove a_t and a_h and replace (a_t, x) and (a_h, y) by (x, y). Repeat until there are no more free ends in the bundle. This process converts a bundle into a cycle. Repeated across all bundles it also removes all the missing genes. Therefore the number of cycles in the partial breakpoint graph plus the number of bundles determined by this graph equals the number of cycles in the breakpoint graph of $\underline{G}_1$ and G_2. □

It was shown in [10] how the one-sided scaffold filling problem can be solved by *completing* each bundle separately, i.e., by inserting the missing genes or drawing the red edges between free ends within each bundle. It turns out that we have the following theorem [10].

Theorem 3. *For a bundle with ν paths, there are $\nu + 1$ cycles produced by completing the bundle, while ν genes inserted in G_2.*

The following corollary follows immediately.

Corollary 1. *After completing all the bundles with $(\nu_1, \nu_2, \cdots, \nu_\beta)$ paths, there are $m = \nu_1 + \nu_2 + \cdots + \nu_\beta$ genes inserted and the number of cycles is $m + \beta$.*

Therefore, we have the following main result.

Theorem 4. $D(G_1, \bar{G}_2) = D(\underline{G}_1, G_2)$.

Proof. The breakpoint graph of G_1 and $\bar{G}_2)$ has m more blue edges than the breakpoint graph of $\underline{G}_1$ and G_2, corresponding to the insertion of the m missing genes. But since it had r cycles before bundle completion, the breakpoint graph of G_1 and $\bar{G}_2)$ has $r + m + \beta$ cycles, from Corollary 1. This is m more than the $r + \beta$ cycles in the breakpoint graph of $\underline{G}_1$ and G_2 (Lemma 7). By definition of the distance D, we have $D(G_1, \bar{G}_2) = D(\underline{G}_1, G_2)$ □

For two-sided scaffold filling, we thus have the following exact algorithm for the case when G_1 contains genes $G + X$ and G_2 contains genes $G + Y$, where G, X and Y are disjoint sets of genes.

1. Remove all the genes in set X from G_1 to get G'_1, containing only the genes in G.
2. Apply one-sided scaffold filling to G_2 and G'_1, inserting all the genes in Y into G'_1, producing genome G''_1.
3. Restore genes in set X to G''_1. The insertion points are only constrained by the gene order in G_1. The new genome, G'''_1 contains all the genes in G, X, and Y.
4. Apply one-sided scaffold filling to G'''_1 and G_2, inserting all the genes in X into G_2.

Because the distance between the two genomes reduced to the genes in G is a lower bound for the distance between any two genomes enlarged through gene insertion, by Lemma 6, and because steps 2 and 4 do not increase this distance, by Theorem 4, and because step 3 is as general as possible while respecting the gene order of G_1, the correctness of the algorithm is verified.

As for the one-sided scaffold filling, once the breakpoint graph is constructed, the remaining steps run in linear time.

6 Concluding Remarks

In this paper, we investigate the scaffold filling problems under both the breakpoint and rearrangement distances. A very interesting open problem is when the missing genes can be only inserted in between contigs (i.e., in some predefined locations); our current method cannot generate any result with some performance guarantee. Another problem is dealing with the cases when gene duplications are allowed. Our NP-completeness proof implies that this is closely related to the Minimum Common String Partition problem, for which the existence of an FPT algorithm [5] is not known yet. In [4,8], several special cases were shown to admit exact algorithms, but when using the optimal number of blocks in the final solution as the only parameter, we do not know whether an FPT algorithm exists or not.

Acknowledgments

This research is partially supported by NSF under grant DMS-0918034, by NSF of China under grant 60928006 and by NSERC of Canada. We also thank anonymous reviewers for several useful comments.

References

1. Chain, P., Grafham, D., Fulton, R., FitzGerald, M., Hostetler, J., Muzny, D., Ali, J., et al.: Genome project standards in a new era of sequencing. Science 326, 236–237 (2009)
2. Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: Computing the assignment of orthologous genes via genome rearrangement. In: Proc. of the 3rd Asia-Pacific Bioinformatics Conf. (APBC 2005), pp. 363–378 (2005)

3. Chrobak, M., Kolman, P., Sgall, J.: The greedy algorithm for the minimum common string partition problem. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM 2004 and APPROX 2004. LNCS, vol. 3122, pp. 84–95. Springer, Heidelberg (2004)
4. 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)
5. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
7. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partitioning problem: Hardness and approximations. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 473–484. Springer, Heidelberg (2004); also in: The Electronic Journal of Combinatorics 12, paper R50 (2005)
8. Jiang, H., Zhu, B., Zhu, D., Zhu, H.: Minimum common string partition revisited. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) FAW 2010. LNCS, vol. 6213, pp. 45–52. Springer, Heidelberg (2010)
9. Kaplan, H., Shafrir, N.: The greedy algorithm for edit distance with moves. Inf. Process. Lett. 97(1), 23–27 (2006)
10. Muñoz, A., Zheng, C., Zhu, Q., Albert, V., Rounsley, S., Sankoff, D.: Scaffold filling, contig fusion and gene order comparison. BMC Bioinformatics 11, 304 (2010)
11. Tesler, G.: Efficient algorithms for multichromosomal genome rearrangements. J. Computer and System Sciences 65, 587–609 (2002)
12. Watterson, G., Ewens, W., Hall, T., Morgan, A.: The chromosome inversion problem. J. Theoretical Biology 99, 1–7 (1982)
13. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)

An Efficient Algorithm for Gene/Species Trees Parsimonious Reconciliation with Losses, Duplications and Transfers

Jean-Philippe Doyon[1], Celine Scornavacca[2], K. Yu. Gorbunov[3], Gergely J. Szöllősi[4], Vincent Ranwez[5], and Vincent Berry[1]

[1] LIRMM, CNRS - Univ. Montpellier 2, France
[2] Center for Bioinformatics (ZBIT), Tuebingen Univ., Germany
[3] Kharkevich IITP, Russian Academy of Sciences, Moscow
[4] LBBE, CNRS - Univ. Lyon 1, France
[5] ISEM, CNRS - Univ. Montpellier 2, France

Abstract. Tree reconciliation methods aim at estimating the evolutionary events that cause discrepancy between gene trees and species trees. We provide a discrete computational model that considers duplications, transfers and losses of genes. The model yields a fast and exact algorithm to infer time consistent and most parsimonious reconciliations. Then we study the conditions under which parsimony is able to accurately infer such events. Overall, it performs well even under realistic rates, transfers being in general less accurately recovered than duplications. An implementation is freely available at `http://www.atgc-montpellier.fr/MPR`.

1 Introduction

Duplications, losses and transfers are evolutionary events that shape genomes of eukaryotes and prokaryotes. They result in discrepancies between gene and species trees. Tree reconciliation aims at estimating the course of these events in order to explain the observed incongruences of gene and species trees. A reconciliation defines an embedding of a gene tree G into a species tree S, and thus locates duplications, transfers and losses. Reconciliation methods find applications in various areas such as functional annotation in genomics [3], coevolutionary studies in ecology [10], and studies on poputation areas in biogeography.

Probabilistic models have been proposed to reconcile trees [15,13], but heavy computing times still limit their use to relatively small sets of taxa and small collections of genes. An alternative approach relies on the more tractable combinatorial principle of parsimony [4]. Yet, with the advent of next generation sequencing technologies, that flood molecular biology with new genomes, even combinatorial methods might become too computationally expensive to handle phylogenomic databases, that regularly deal with several dozen thousands of gene families [11]. In this paper, we propose a combinatorial reconciliation method that has the potential to keep pace with new sequencing technologies.

More formally, we consider the *Most Parsimonious Reconciliation* (MPR) problem: given a species tree S, a gene tree G and respective costs for duplication,

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 93–108, 2010.

transfer and loss events (respectively denoted $\mathbb{D}$, $\mathbb{T}$, and $\mathbb{L}$ events), compute a time-consistent reconciliation that has a minimum cost. Time consistency means that $\mathbb{T}$ events happen only horizontally, i.e. between coexisting species, and the cost of a reconciliation is the sum of the costs of the events implied by the embedding of G into S. For instance, when $\mathbb{D}$, $\mathbb{T}$, and $\mathbb{L}$ events have cost $5, 10, 1$ respectively, the reconciliation of Fig. 1 (left) costs 23.

When only $\mathbb{DLS}$ events are considered ($\mathbb{S}$ refers to a speciation), the MPR problem can be solved in linear time w.r.t. the size of G for binary trees [17] and remains tractable when S is polytomous [16]. However, when $\mathbb{T}$ events are considered, the MPR problem is NP-complete, even for reconciling two binary trees [14]. This strong contrast in complexity is explained by the difficulty of managing the chronological constraints among nodes of S that are induced by $\mathbb{T}$ events. When not constraining $\mathbb{T}$ events, time inconsistent scenarios can ensue (see Fig. 1; right), as remarked in [14]. These authors solve a variant of the MPR problem with a fast $O(|S|^2 \cdot |G|)$ algorithm, but that does not handle the time consistency constraints and considers losses a posteriori. A promising approach is to alter the definition of MPR to accept a dated tree S as input [9,1,10,5,15]. Dates for nodes of S can be obtained by relaxed molecular clock techniques working from gene trees and molecular sequences. Relative dates are sufficient for reconciliation, hence they are little limited by the possible absence of fossil records for the studied species [8]. Given a dated tree S, time consistency can be ensured *locally* by only considering $\mathbb{T}$ events whose donor and receiver branches have intersecting time intervals [10]. However, two locally consistent $\mathbb{T}$ events can be *globally* inconsistent, which then needs to be fixed by altering afterwards the position of the proposed $\mathbb{T}$ [10], but this approach does not guarantee to solve MPR exactly. To ensure *global* consistency, branches of S can be subdivided into time slices transversal to all edges. Then, slices are explored one after the other, and only combinations of $\mathbb{T}$ events in a same time slice are considered. This recently led to two exact algorithms, one running in $O((|S| \cdot |G|)^4)$ [7] and one claiming a complexity of $O(|S|^2 \cdot |G|)$ [5,6].

We propose here a formal modelization that leads to a fast exact algorithm solving the time consistent MPR problem for a dated species tree in $O(|S'| \cdot |G|)$,

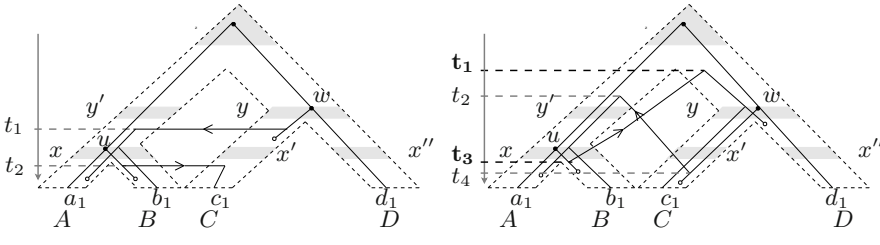


Fig. 1. Two scenarios for a gene tree G (plain lines) along a species tree S (tubes), where the symbol $\circ$ represents loss. (Left) A time consistent scenario. (Right) A scenario that is not time consistent: the transfer from the donor at t_3 (resp. t_4) to a receiver at t_1 (resp. t_2) implies that u predates (resp. follows) w.

where S' is a subdivision of S in at most $O(|S|^2)$ nodes. Then, we rely on an implementation of this fast algorithm to obtain a first insight for the question: *Is parsimony relevant to infer the true evolutionary scenario of a gene family?*

2 Methods

2.1 Basic Definitions and Notations

Let T be a tree with nodes $V(T)$ and branches $E(T)$, and such that only its leaves are labeled. Let $r(T)$, $L(T)$, and $\mathcal{L}(T)$ respectively denote its root node, the set of its leaf nodes, and the set of taxa labelling its leaves. We will adopt the convention that the root is at the top of the tree and the leaves at the bottom.

An edge of T is denoted $(u, v) \in E(T)$, where u is the parent of v. For a node u of T, T_u denotes the subtree of T rooted at u, u_p its parent, (u_p, u) its *parent edge*, and $T_{(u_p,u)}$ denotes the subtree of T rooted at edge (u_p, u). Given a subset of leaves $K \subseteq L(T)$, the *homeomorphic tree* of T connecting K, denoted T_K, is the smallest binary tree induced from T such that $L(T_K) = K$. T is a *dated tree* if it is associated with a *date function* $\theta_T : V(T) \to \mathbb{R}$ such that for any two nodes $x, x' \in V(T)$, if x' is a strict descendant of x then $\theta_T(x') < \theta_T(x)$.

An internal node u of T has one or two children, where $\{u_1\}$ and $\{u_1, u_2\}$ respectively denote its child set. It is important to point out that because T is an unordered tree, the children u_1 and u_2 of u are interchangeable. Given two nodes u, u' of T, u' is said to be a (resp. strict) *descendant* of u if u is on the path from u' to $r(T)$ (resp. and $u \neq u'$). An internal node u of T is said to be *artificial* when it has one and only one child. *Contracting* an artificial node means that the node is removed from the tree and that its two adjacent edges are merged. A tree T' is said to be a *subdivision* of a tree T if the recursive contraction of all artificial nodes of T' yields T.

A *species tree* S is a rooted binary tree such that each element of $\mathcal{L}(S)$ represents an extant species labeling exactly one leaf of S (there is a bijection between $L(S)$ and $\mathcal{L}(S)$). A date function θ_S for S (as defined above) ensures that $\forall x \in L(S)$, $\theta_S(x) = 0$. A *gene tree* G is a rooted binary tree. From now on, we consider a species tree S and a gene tree G such that $\mathcal{L}(G) \subseteq \mathcal{L}(S)$ and where $\mathcal{L} : L(G) \to L(S)$ denotes the function that maps each leaf of G to the unique leaf of S with the same label (leaves of G are labeled with the species from which genes were sampled). To distinguish between G and S, the term *edge* refers to G and the term *branch* refers to S.

We introduce below the concept of a scenario describing the evolution of a gene that starts at node $r(S)$ and evolves along S according to $\mathbb{DTLS}$ events. Such a scenario generates a *completed gene tree* denoted G^o, whose leaf set is formed of contemporary genes (denoted $L_{\mathbb{C}}(G^o)$) but also of lost genes (denoted $L_{\mathbb{L}}(G^o)$), see Fig. 1 and 2. Note that $L(G^o) = L_{\mathbb{C}}(G^o) \cup L_{\mathbb{L}}(G^o)$.

Definition 1. *Given an observed gene tree G and a species tree S, with its time stamp function θ_S, a* $\mathbb{DTLS}$ scenario *for G along S is denoted (G^o, M, θ_{G^o}), where G^o is a completed gene tree, $M : V(G^o) \to V(S)$ maps each node of G^o to*

a node of S, and $\theta_{G^o} : V(G^o) \to [0, \theta_S(r(S))]$ is a date function that associates each node of G^o to a time stamp of S. The scenario associates a $\mathbb{DTLS}$ event to each node $u \in V(G^o) \setminus L_{\mathbb{C}}(G^o)$ as described below (where u_1 and u_2 are the two children of u and $x = M(u)$).

1. *If u is a leaf of $L_{\mathbb{L}}(G^o)$, then it corresponds to an $\mathbb{L}$ event.*
2. *If $M(u_1) = x_1$, and $M(u_2) = x_2$, then u is an $\mathbb{S}$ event happening at x in S'.*
3. *If $M(u_1) = x$ and $M(u_2) = x$, then u is a $\mathbb{D}$ event along the branch (x_p, x).*
4. *If $M(u_1) = x$, $M(u_2) = y$, and y is neither an ancestor nor a descendant of x, then u is a $\mathbb{T}$ event, where (x_p, x) and (y_p, y) respectively correspond to the donor and the receiver branches.*

A $\mathbb{DTLS}$ scenario is said to be consistent *if and only if (1) the homeomorphic gene tree $G^o_{L_{\mathbb{C}}(G^o)}$ is isomorphic to G and (2) for a $\mathbb{T}$ event (i.e. Def. 1 (4)) $[\theta_S(x), \theta_S(x_p)] \cap [\theta_S(y), \theta_S(y_p)] \neq \emptyset$.*

The cost *of such a scenario is denoted $Cost(G^o, M, \theta_{G^o}) = d\delta + t\tau + l\lambda$, where d, t, and l respectively denote the number of $\mathbb{D}$, $\mathbb{T}$, and $\mathbb{L}$ events, and δ, τ, and λ are their respective costs.*

Consider a species tree S with a time stamp function θ_S, an observed gene tree G, the leaf-association function $\mathcal{L} : L(G) \to L(S)$, and costs δ, τ, resp. λ for $\mathbb{D}$, $\mathbb{T}$ resp. $\mathbb{L}$ events. Given these inputs, the optimization problem considered in the present paper, called *MPR*, is to compute a consistent $\mathbb{DTLS}$ scenario (G^o, M, θ_{G^o}) for G along S that minimizes $Cost(G^o, M, \theta_{G^o})$.

2.2 A Tractable Model of Reconciliation

To obtain a tractable model, we discretize time by subdividing the species tree into time slices (similarly as done in [1,13]), then define a limited number of cases for events to happen, that still allows us to infer a most parsimonious scenario.

Definition 2. *(see Fig. 3) Given a species (binary) tree S and a time stamp function $\theta_S : V(S) \to \mathbb{R}$, let S' be the* subdivision *of S constructed as follows: for each node $x \in V(S) \setminus L(S)$ and each branch $(y_p, y) \in E(S)$ s.t. $\theta_S(y_p)$*

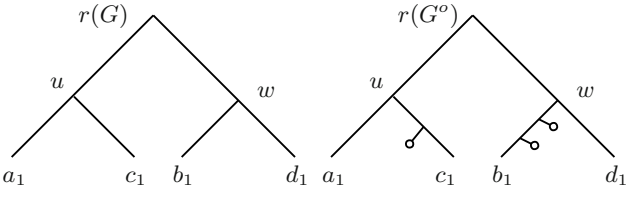


Fig. 2. (Left) An observed gene tree G with four leaves a_1, b_1, c_1, and d_1, respectively belonging to the contemporary species A, B, C, and D (see Fig. 1). (Right) A completed gene tree G^o, with $L(G^o) = L_{\mathbb{C}}(G^o) \cup L_{\mathbb{L}}(G^o)$, where $L_{\mathbb{C}}(G^o) = \{a_1, b_1, c_1, d_1\}$, and $L_{\mathbb{L}}(G^o)$ is formed of the three nodes labelled $\circ$. G is the homeomorphic tree G^o_K, where $K = L_{\mathbb{C}}(G^o)$.

$> \theta_S(x) > \theta_S(y)$, *an* artificial *node is inserted along the branch* (y_p, y) *at time* $\theta_S(x)$. *This subdivision allows us to define a time stamp function* $\theta'_{S'}$ *for* S' *only from its topology: for any* $x \in V(S')$, $\theta'_{S'}(x)$ *is the number of edges separating* x *from one of its descendant leaves (which leaf does not matter as they are all at the same distance from* x*).*

The time stamp of a branch (x_p, x) of S' is denoted $\theta'_{S'}(x_p, x) = \theta'_{S'}(x)$. Moreover, for a time t, let $E_t(S') = \{(x_p, x) \in E(S') : \theta'_{S'}(x_p, x) = t\}$ denote the set of branches of S' located at time t.

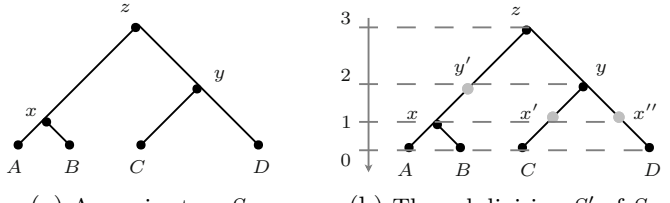
Definition 3. *Consider a gene tree* G *and a species tree* S *with a time stamp function* θ_S, *and let* S' *be the subdivision of* S *and* $\theta'_{S'} : V(S') \to \mathbb{N}$ *be the corresponding time stamp function. A* reconciliation *between* G *and* S *is denoted* α *and maps each edge* $(u_p, u) \in E(G)$ *onto an ordered sequence of branches of* S', *denoted* $\alpha(u_p, u)$, *where* ℓ *denotes its length and* $\alpha_i(u_p, u)$ *its* i*-th element for* $1 \leq i \leq \ell$. *Each branch* $\alpha_i(u_p, u)$, *denoted below* (x_p, x), *respects one and only one of the following constraints (see Fig. 4).*

First, consider the case that (x_p, x) *is the last branch* $\alpha_\ell(u_p, u)$ *of the sequence* $\alpha(u_p, u)$. *If* u *is a leaf of* G, *then* x *is the unique leaf of* S' *that has the same label as* u *(that is* $x = \mathcal{L}(u)$*) (Contemporary taxa mappings). Otherwise, one of the cases below is true.*

- $\{\alpha_1(u, u_1), \alpha_1(u, u_2)\} = \{(x, x_1), (x, x_2)\}$ ($\mathbb{S}$ *event);*
- $\alpha_1(u, u_1)$ *and* $\alpha_1(u, u_2)$ *are both equal to* (x_p, x) ($\mathbb{D}$ *event);*
- $\{\alpha_1(u, u_1), \alpha_1(u, u_2)\} = \{(x_p, x), (x'_p, x')\}$, *where* (x'_p, x') *is any branch of* S' *other than* (x_p, x) *and located at time* $\theta'_{S'}(x_p, x)$ ($\mathbb{T}$ *event).*

If (x_p, x) *is not the last branch* $\alpha_\ell(u_p, u)$ *of the sequence (i.e.* $i < \ell$*), one of the following cases is true.*

- x *is an artificial node of* S' *with a single child* x_1, *and the next branch* $\alpha_{i+1}(u_p, u)$ *is* (x, x_1) ($\varnothing$ *event);*
- x *is not artificial and* $\alpha_{i+1}(u_p, u) \in \{(x, x_1), (x, x_2)\}$ ($\mathbb{SL}$ *event);*
- $\alpha_{i+1}(u_p, u) = (x'_p, x')$ *is any branch of* S' *other than* (x_p, x) *and located at time* $\theta'_{S'}(x_p, x)$ ($\mathbb{TL}$ *event).*



(a) A species tree S (b) The subdivision S' of S

Fig. 3. The species tree S and its subdivision S'. The artificial nodes of S' are represented by gray circles and denoted y', x', and x'', where $\theta'_{S'}(x) = \theta'_{S'}(x') = \theta'_{S'}(x'')$ and $\theta'_{S'}(y) = \theta'_{S'}(y')$.

A reconciliation α between the gene tree G of Fig. 2 (left) and the subdivision S' of Fig. 3b is depicted in Fig 1(left), where the path $\alpha(w, b_1)$ along S' associated to the edge (w, b_1) is $[(y, x'), (y', x), (x, B)]$. Observe that the extended gene tree G^o (see Fig. 2; right) is a by-product of the reconciliation α.

Note that $\mathbb{T}$ events only happen between branches in a same time slice, hence only time-consistent scenarios are generated by this model. We now argue that the model allows to infer most parsimonious scenarios (see Def. 1). First, note that each loss is coupled with either a speciation ($\mathbb{SL}$) or a transfer ($\mathbb{TL}$). Indeed, any *most parsimonious* reconciliation embedding G into S' only needs to use a loss when it meets a speciation node of S' where G goes into only one descending tube, or when leaving a tube due to a transfer, with no part of G remaining in the donor tube. Considering a lone loss as a seventh event in Fig. 4 would lead us to examine reconciliations that are not most parsimonious, as this would only allow us to replace – in a G^o tree generated by the current model – a single $l \in L_{\mathbb{L}}(G^o)$ by a subtree with no extant species (as the structure of G is common to both these completed gene trees). Such a subtree contains at least two losses and is hence less parsimonious than leaving leaf l in the G^o proposed with the current model. Then, any combination of $\mathbb{DTLS}$ events resulting from a scenario (Def 1) can be reproduced by the model of Def. 3, safe for combinations that would obviously not lead to most parsimonious scenarios: a speciation of a gene where its two sons go extinct before reaching the leaves of S'; a gene duplication where at least one of its sons goes extinct; a transfer where the transfered gene lineage goes extinct.

Last, note that all cases considered in Def. 3 (see Fig. 4) allow us to progress either in the time slices of S' or along the edges of G. This is because a $\mathbb{TL}$ case can not be followed by a second one in a most parsimonious scenario (see Prop. 1 in appendix). Thus, the model offers all ingredients for a dynamic programming algorithm that finds a most parsimonious and time consistent scenario, while still running in time polynomial in $|S'|$ and $|G|$. In other words, this model allows to solve the MPR problem exactly and in a tractable way.

Note that since the model places each loss immediately after another event (speciation or transfer), it is not able to generate a most parsimonious scenario $\sigma = (G^o, M, \theta^o_G)$ where a lineage is lost after being alive for several slices in a same tube (without meeting a speciation node). However, it can generate a scenario $\sigma' = (G^o, \bar{M}, \bar{\theta}^o_G)$ that can be seen as a canonical representative of σ: both scenarios share the same G^o and have the same number and localization of $\mathbb{D}$, $\mathbb{T}$, and $\mathbb{L}$ events in S (σ and σ' only differ in the position of some $\mathbb{L}$ in the *subdivided* species tree S').

2.3 An Efficient Algorithm to Solve MPR

In this section, we propose a polynomial time and space algorithm that uses the tractable reconciliation model of Def. 3 to solve the MPR problem (see Algorithm 1).

Consider an edge $(u_p, u) \in E(G)$, a branch $(x_p, x) \in E(S')$, and the time $t = \theta'_{S'}(x_p, x)$. Let $Cost(u, x)$ denote the minimal cost over all reconciliations

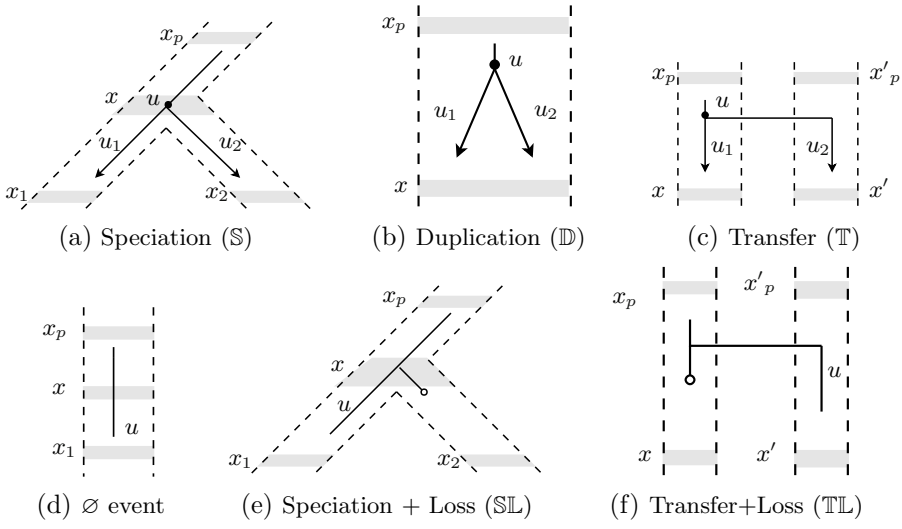


Fig. 4. The six 𝔻𝕋𝕃𝕊 events of Def. 3, where an edge (u_p, u) of G^o is mapped onto a branch (x_p, x) of the sequence $\alpha(u_p, u)$. The extended gene tree G^o is embedded in the subdivision S' of a species tree S, where an edge of G corresponds to a plain line, a branch of S' corresponds to a dotted tube (white zone), and a node of S' corresponds to a gray zone.

between $G_{(u_p,u)}$ and the forest of subtrees of S' rooted with a branch located at time t, and such that (x_p, x) is the first branch in the sequence associated to (u_p, u) (that is $\alpha_1(u_p, u) = (x_p, x)$; see Def. 3). Assuming that the gene tree G and the species tree S are rooted with an artificial branch, $Cost(r(G), r(S'))$ corresponds to the minimal cost over all reconciliations between G and S. The dynamic programming algorithm (see pseudo-code in Algorithm 1) fills the matrix $Cost : V(G) \times V(S') \to \mathbb{N}$ through two embedded loops: one that visits all edges according to a bottom-up traversal of G and one that visits all time stamps of S' in backward time order (i.e. starting from 0). For the edge (u_p, u) and the time stamp t currently considered (respectively in lines 3 and 4), two consecutive loops over all branches $(x_p, x) \in E_t(S')$ compute the minimal cost of mapping (u_p, u) onto (x_p, x) according to the six events 𝕊, 𝔻, 𝕋, ∅, 𝕊𝕃, and 𝕋𝕃 (see Fig. 4). For a branch $(x_p, x) \in E_t(S')$, the first loop (lines 5 to 20) computes the minimal cost among the first five events. 𝕋𝕃 events can be considered separately (lines 21 to 24) as they may never be immediately followed by a second 𝕋𝕃 in a most parsimonious scenario (as implied by Property 1 in Appendix). $Cost(u, x)$ is the minimum of the values computed in these two loops.

The case of Fig. 4c is considered at lines 13 to 15 where the cost of a 𝕋 event starting at (x_p, x) is computed for edge (u_p, u). Assuming that (u, u_1) (resp. (u, u_2)) is the transfered gene lineage, a subroutine called *BestReceiver* computes

Algorithm 1. Computes $Cost(r(G), r(S'))$ according to the $\mathbb{DTL}$ costs, respectively denoted δ, τ, and λ.

1. Construct the subdivision S' of S as described in Def. 2
2. The matrix $Cost : V(G) \times V(S') \to \mathbb{N}$ is initialized as follows: if $u \in L(G)$, $x \in L(S')$, and $\mathcal{L}(u) = x$, then $Cost(u, x) \leftarrow 0$. Else, $Cost(u, x) \leftarrow \infty$.
3. **for all** $(u_p, u) \in E(G)$ according to a bottom-up traversal **do**
4. **for all** $t \in \{0, 1, \ldots, \theta'_{S'}(r(S'))\}$ in backward time order **do**
5. **for all** branch $(x_p, x) \in E_t(S')$ **do**
6. **if** $u \in L(G)$, $x \in L(S')$, and $\mathcal{L}(u) = x$ **then**
7. Skip lines 8 to 20 and go to the next iteration of the loop at line 5 {Base case}
8. $Cost_g \leftarrow \infty$, for each $g \in \{\mathbb{S}, \mathbb{D}, \mathbb{T}, \varnothing, \mathbb{SL}\}$
9. **if** u has two children **then**
10. **if** x has two children **then**
11. $Cost_{\mathbb{S}} \leftarrow \min\{Cost(u_1, x_1) + Cost(u_2, x_2), Cost(u_1, x_2) + Cost(u_2, x_1)\}$
12. $Cost_{\mathbb{D}} \leftarrow Cost(u_1, x) + Cost(u_2, x) + \delta$
13. $(y_p, y) \leftarrow BestReceiver((u, u_1), t, (x_p, x))$
14. $(z_p, z) \leftarrow BestReceiver((u, u_2), t, (x_p, x))$
15. $Cost_{\mathbb{T}} \leftarrow \min\{\ Cost(u_1, x) + Cost(u_2, z), Cost(u_1, y) + Cost(u_2, x)\ \} + \tau$
16. **if** x has a single child **then**
17. $Cost_{\varnothing} \leftarrow Cost(u, x_1)$
18. **if** x has two children **then**
19. $Cost_{\mathbb{SL}} \leftarrow \min\{Cost(u, x_1), Cost(u, x_2)\} + \lambda$
20. $Cost(u, x) \leftarrow \min\{Cost_g : g \in \{\mathbb{S}, \mathbb{D}, \mathbb{T}, \varnothing, \mathbb{SL}\}\}$
21. **for all** branch $(x_p, x) \in E_t(S')$ **do**
22. $(x'_p, x') \leftarrow BestReceiver((u_p, u), t, (x_p, x))$
23. $Cost_{\mathbb{TL}} \leftarrow Cost(u, x') + \tau + \lambda$
24. $Cost(u, x) \leftarrow \min\{\ Cost(u, x),\ Cost_{\mathbb{TL}}\ \}$
25. **return** $Cost(r(G), r(S'))$

the branch (y_p, y) (resp. (z_p, z)) that minimizes $Cost(u_1, y)$ (resp. $Cost(u_2, z)$) over all branches of S' located at the same time t, other than (x_p, x). The same subroutine is used at line 22 for the $\mathbb{TL}$ case of Fig. 4f. A similar optimization to compute the optimal receiver for a transfer was found independently in [13].

Algorithm 1 computes the cost of a most parsimonious reconciliation. Backtracking in the computations of values in the dynamic programming table yields a most parsimonious *reconciliation* (in the sense of Def. 3), which readily allows to obtain a most parsimonious *scenario* (see Def. 1), as we argued in Section. 2.2. This algorithm achieves fast running times, in part due to a factorization of the computations of the best receivers (see Appendix for details).

Theorem 1. *The MPR problem can be solved in* $\Theta(|S'| \cdot |G|)$ *time and space.*

3 Experimental Results

To assess the performance of parsimony, we calculated the most parsimonious reconciliations for a large scale simulated data set that was obtained using a probabilistic model of duplication, transfer, and loss. In our simulations, we started with a single gene at the root of the species tree and generated gene trees according to a Poisson process characterized by rates of duplication, transfer and loss. We compiled two different data sets called ds_1 and ds_2, aiming both to simulate a relatively large phylogenetic time scale (a bacterial or archean phylum) with realistic loss rates as well as to explore a wide range of duplication and transfer rates. For further details on ds_1 and ds_2, see the Appendix.

For each data set, we used a single cost per event corresponding to the inverse of the average rate of this event during the simulation process (i.e., for ds_1 $\delta = 1/0.18$). According to those costs and for each pair of gene and species trees, we used Algorithm 1 to compute one of the most parsimonious reconciliations denoted α_p.

Note that the real reconciliation α_R may contain the record of events that cannot be recovered by a reconciliation for G, since no traces of them exist. For instance, subtrees whose leaves are all lost, $\mathbb{D}$ events followed straightaway by an $\mathbb{L}$ event, or several $\mathbb{TL}$ events in a row. Thus, we post-processed the $\mathbb{DTL}$ events of α_R, removing hidden parts of α_R of the above kinds, but potentially leaving other unrecoverable parts.This leads to obtain a reconciliation α'_R.

We first study under which conditions the parsimony criterion can correctly estimate the $\mathbb{DTL}$ events that lead to an observed gene tree G. This can be simply achieved by comparing the costs of the real scenario and that of a most parsimonious one. As soon as the two costs strongly differ, the parsimony is no longer a reasonable approach. Recall that the cost of a reconciliation α can be computed as $Cost(\alpha) = d\delta + t\tau + l\lambda$, where d, t and l are the number of $\mathbb{D}$, resp. $\mathbb{T}$, resp. $\mathbb{L}$ implied by α. The relative over cost of α'_R in terms of parsimony score compared to that of a most parsimonious one is defined below:

$$OverCost(\alpha'_R, \alpha_P) = \frac{Cost(\alpha'_R) - Cost(\alpha_P)}{Cost(\alpha_P)}.$$

Since several most parsimonious scenarios can exist, that $Cost(\alpha'_R) = Cost(\alpha_P)$ does not imply $\alpha_P = \alpha'_R$. Fig. 5 shows the extent of this over cost depending on the duplication and transfer rates and tree heights. We can see that the over cost is really small for all combinations of duplication and transfer rates we investigated, but does increase with the height of the gene trees. This can be related to hidden events that we failed to identify and remove from α'_R.

We now proceed to investigate quantitatively whether parsimony is able to correctly infer the position of $\mathbb{DTL}$ events.

Recall that a reconciliation α of a gene tree G defines $\mathbb{DTL}$ events associated to internal nodes and edges of G. As the position of duplication and transfer events in G^o allow to locate losses, we only focus below on $\mathbb{D}$ and $\mathbb{T}$ events. Let $\mathbb{D}(\alpha) \subseteq V(G) \setminus L(G)$ denote the subset of internal nodes of G that correspond to a $\mathbb{D}$ event and $\mathbb{T}(\alpha) \subseteq E(G)$ the subset of edges of G that correspond to a $\mathbb{T}$ event. It is important to point out that $\mathbb{D}(\alpha)$ and $\mathbb{T}(\alpha)$ alone do not resolve where

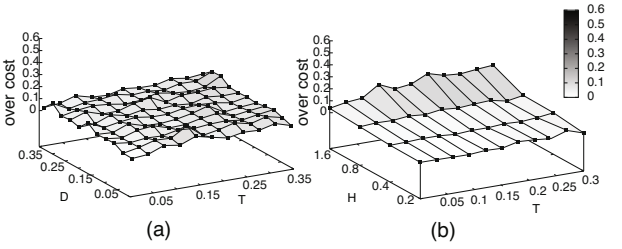


Fig. 5. Over cost of simulated scenarios compared to that of most parsimonious ones for combinations of heights, transfer and duplication rates, i.e. ds_1 (a) and ds_2 (b). High values show cases where parsimony criterion is inadequate.

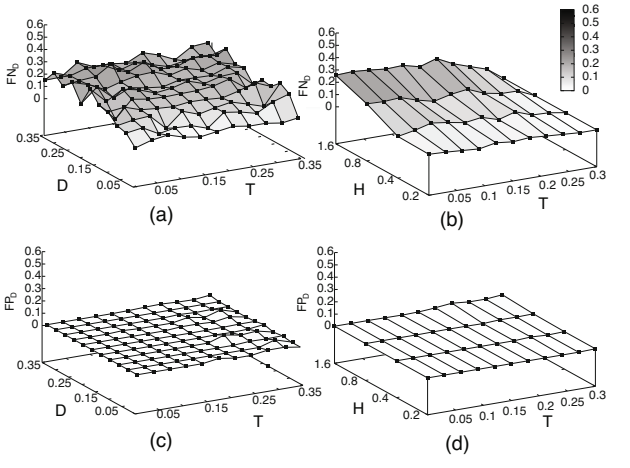


Fig. 6. Accuracy of parsimony to estimate reconciliations: ratios of false negative (a,b) and false positive (c,d) to estimate and localize $\mathbb{D}$ events, for combinations of heights, transfer and duplication rates, i.e. ds_1 (a,c) and ds_2 (b,d)

in S the event has taken place, hence are not sufficient to determine whether a $\mathbb{DTL}$ event is common to two reconciliations. Let $\mathbb{D}_S(\alpha)$ denote the set of pairs $(u,(x_p,x)) \in \mathbb{D}(\alpha) \times E(S)$ such that α places u on the branch (x_p,x) of S. Let $\mathbb{T}_S(\alpha)$ denote the triplet set $((u_p,u),(x_p,x),(y_p,y)) \in \mathbb{T}(\alpha) \times E(S)^2$ such that (u_p,u) is a $\mathbb{T}$ event from the donor (x_p,x) to the receiver (y_p,y) branches in S.

Given a most parsimonious reconciliation α_P, its accuracy to retrieve the $\mathbb{D}$ and $\mathbb{T}$ events of the real (simulated) reconciliation α'_R is evaluated by the ratios of false positive and false negative events defined as follows:

$$FP_{\mathbb{E}}(\alpha'_R,\alpha_P) = \frac{|\mathbb{E}_S(\alpha_P) - \mathbb{E}_S(\alpha'_R)|}{|\mathbb{E}_S(\alpha_P)|}$$
$$FN_{\mathbb{E}}(\alpha'_R,\alpha_P) = \frac{|\mathbb{E}_S(\alpha'_R) - \mathbb{E}_S(\alpha_P)|}{|\mathbb{E}_S(\alpha'_R)|},$$

where $\mathbb{E} = \mathbb{D}, \mathbb{T}$. Figures 6 and 7 show those ratios for various combinations of $\mathbb{D}$, $\mathbb{T}$ rates and tree heights.

In Fig. 6, we can see that $FP_{\mathbb{D}}$ is close to zero for all combinations of duplication and transfer rates: almost all parsimonious duplications are correct (i.e., present in α'_R). The high values of $FN_{\mathbb{D}}$ can be explained by several reasons. First, α'_R can contain hidden events that cannot be detected by reconciliation approaches. Second, fixing $\delta = \tau$ causes the misidentification of some $\mathbb{D}$ events replaced by $\mathbb{T}$ events in the inference. This would also explain the high ratio of false positive transfers with such rates (see Fig. 7). Finally, this can be due to the wrong most parsimonious reconciliation proposed among the several possible ones. This also explains the quite high level of false negatives for $\mathbb{T}$ events.

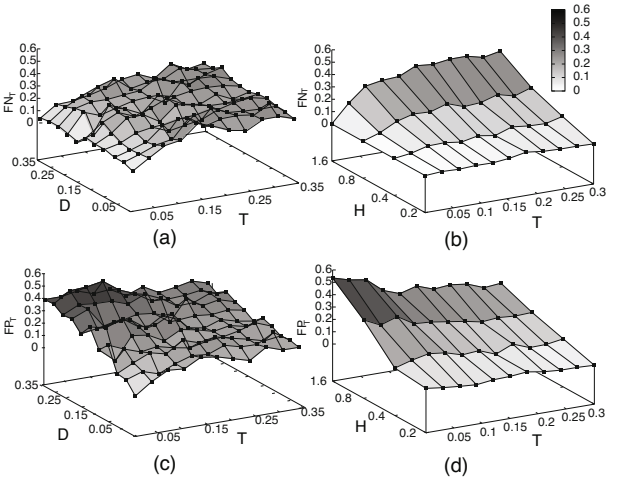


Fig. 7. Accuracy of parsimony to estimate reconciliations: ratios of false negative (a,b) and false positive (c,d) to estimate and localize $\mathbb{T}$ events, for combinations of heights, transfer and duplication rates, i.e. ds_1 (a,c) and ds_2 (b,d)

4 Conclusion

We presented a new model for reconciling gene and species trees. This model leads to a fast and exact algorithm to compute a time consistent and most parsimonious reconciliation while accounting for duplications, losses and transfers. Simulations showed that the parsimony criterion performs satisfactorily under realistic conditions at the phylum level. At the inter-phlyum level, transfers are more difficult to recover and the existence of several most-parsimonious reconciliations might be a decisive factor there. This needs further scrutiny. Moreover, running times are on average 1.09s (resp. 1.38s) for low (resp. high) rates of events for trees on 100 species. This clearly scales the reconciliation approach up to the phylogenomic stage, where several tens of thousand genes are considered.

Many things remain to be done, among others to allow for multifurcating gene and species trees and to measure the accuracy of the reconciliation approach for orthology prediction (where the localization of events is not needed, increasing the accuracy of the method w.r.t. our results) compared to other methods.

References

1. Conow, C., Fielder, D., Ovadia, Y., Libeskind-Hadas, R.: Jane: a new tool for the cophylogeny reconstruction problem. Algorithms Mol. Biol. 5, 16 (2010)
2. Csuros, M., Miklos, I.: Streamlining and Large Ancestral Genomes in Archaea Inferred with a Phylogenetic Birth-and-Death Model. Mol. Biol. Evol. 26(9), 2087–2095 (2009)
3. Gabaldon, T.: Computational approaches for the prediction of protein function in the mitochondrion. Am J. Physiol. Cell. Physiol. 291(6), C1121–C1128 (2006)
4. Goodman, M., Czelusniak, J., Moore, G.W., Herrera, R.A., Matsuda, G.: Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences. Syst. Zool. 28, 132–163 (1979)
5. Gorbunov, K.Y., Lyubetsky, V.A.: Reconstructing genes evolution along a species tree. Mol. Biol (Mosk.) 43, 946–958 (2009)
6. Gorbunov, K.Y., Lyubetsky, V.A.: An algorithm of reconciliation of gene and species trees and inferring gene duplications, losses and horizontal transfers. Information processes 10(2), 140–144 (2010) (in Russian)
7. Libeskind-Hadas, R., Charleston, M.A.: On the computational complexity of the reticulate cophylogeny reconstruction problem. JCB 16(1), 105–117 (2009)
8. Loader, S.P., Pisani, D., Cotton, J.A., Gower, D.J., Day, J.J., Wilkinson, M.: Relative time scales reveal multiple origins of parallel disjunct distributions of african caecilian amphibians. Biol. Lett., 505–508 (October 2007)
9. Merkle, D., Middendorf, M.: Reconstruction of the cophylogenetic history of related phylogenetic trees with divergence timing information. Theory Biosci. 123(4), 277–299 (2005)
10. Merkle, D., Middendorf, M., Wieseke, N.: A parameter-adaptive dynamic programming approach for inferring cophylogenies. BMC Bioinformatics 11(suppl. 1), S60 (2010)
11. Penel, S., Arigon, A.M., Dufayard, J.F., Sertier, A.S., Daubin, V., Duret, L., Gouy, M., Perriere, G.: Databases of homologous gene families for comparative genomics. BMC Bioinformatics 10(suppl. 6), S3 (2009)
12. Rambaut, A.: Phylogen: phylogenetic tree simulator package (2002)
13. Tofigh, A.: Using Trees to Capture Reticulate Evolution, Lateral Gene Transfers and Cancer Progression. PhD thesis, KTH Royal Institute of Technology, Sweden (2009)
14. Tofigh, A., Hallett, M., Lagergren, J.: Simultaneous identification of duplications and lateral gene transfers. In: IEEE/ACM TCBB, p. 99 (2010)
15. Tofigh, A., Sjöstrand, J., Sennblad, B., Arvestad, L., Lagergren, J.: Detecting LGTs using a novel probabilistic model integrating duplications, lgts, losses, rate variation, and sequence evolution (manuscript)
16. Vernot, B., Stolzer, M., Goldman, A., Durand, D.: Reconciliation with non-binary species trees. J. Comput. Biol. 15, 981–1006 (2008)
17. Zhang, L.: On a mirkin-muchnik-smith conjecture for comparing molecular phylogenies. Journal of Computational Biology 4(2), 177–187 (1997)

A Some Proofs

Property 1. Consider a parsimonious reconciliation α between G and S, an edge (u_p, u) of G and a time t. The sequence $\alpha(u_p, u)$ contains at most two branches of S' located at time t. If there are two such branches denoted $\alpha_i(u_p, u)$ and $\alpha_j(u_p, u)$, then they are adjacent in the sequence $\alpha(u_p, u)$ (i.e. $|i - j| = 1$).

Proof. The adjacency of the two branches follows immediately from Def. 3 (relying on the fact that both happen at time t).

Assume that α contains two $\mathbb{TL}$ events for (u_p, u) described as follows: there are three adjacent branches $\alpha_i(u_p, u)$, $\alpha_{i+1}(u_p, u)$ and $\alpha_{i+2}(u_p, u)$ in $E_t(S')$, which respectively corresponds (according to Def. 3) to the donor of the first $\mathbb{TL}$ event, the receiver (resp. donor) of the first (resp. second) $\mathbb{TL}$ event, and the receiver of the second $\mathbb{TL}$ event.

As the cost of a single $\mathbb{TL}$ event between $\alpha_i(u_p, u)$ and $\alpha_{i+2}(u_p, u)$ is smaller than the cost for the previous two $\mathbb{TL}$ events, α is not a parsimonious reconciliation. □

Proof of the Complexity of the Algorithm

Proof of the Time Complexity. We claim the algorithm runs in $O(n'm)$ where n' is the size of the subdivides species tree S' and m is the size of G.

The loop over the edges of G (line 3) runs for $\Theta(m)$ iterations. The loop over the times t of S' (line 4) together with the two loops over branches $E_t(S')$ in sequence (line 5 and 21) run for $\Theta(n')$ iterations. Thus, lines 6 to 20 and lines 22 to 24 are run $\Theta(n'm)$ time globally. For the nodes $u \in V(G)$ and $x \in V(S')$ currently visited, we now have to prove that $Cost(u, x)$ can be computed in constant time, which is obviously the case for the cost associated to the $\mathbb{S}$, $\mathbb{D}$, $\varnothing$, and $\mathbb{SL}$ events (see lines 11, 12, 16, and 18, respectively). We prove below how the cost associated to a $\mathbb{T}$ event (lines 13 to 15) can be computed in constant time, considering that both genes are conserved (we omit the case for a $\mathbb{TL}$ combination at lines 22 to 24, as it is solved using the same optimization idea).

Consider a $\mathbb{T}$ event from a donor $(x_p, x) \in E_t(S')$, assuming w.l.o.g. that (u, u_1) is conserved in the lineage (x_p, x) while (u, u_2) is transfered. The algorithm needs to compute the optimal receiver (i.e. that leading to a minimum cost) for (u, u_2) in $E_t(S') \setminus \{(x_p, x)\}$. As currently stated, i.e. in the most readable form, Algorithm 1 allows to compute the best receiver in $\Theta(|E_t(S')|)$ time by a simple loop over the branch set $E_t(S')$ (line 14; subroutine BestReceiver). However, slightly modifying the statement of the algorithm allows to compute the best receiver in constant time at line 14 (and similarly lines 13 and 22). To achieve this, immediately before the loop over the branch set $E_t(S')$ (line 5), add another loop on $E_t(S')$ to find the first and second optimal receivers for (u, u_2) in $E_t(S')$ and denote (x'_p, x') and (x''_p, x'') these respective receivers. Second, when a donor $(x_p, x) \in E_t(S')$ is visited during the loop at line 5, the optimal receiver for (u, u_2) in $E_t(S') \setminus \{(x_p, x)\}$ is the first optimal receiver if $(x_p, x) \neq (x'_p, x')$, and the second one otherwise. Hence line 13 now requires constant time, while adding the additional loop mentioned above doesn't cost more

than the already existing loop of line 5. As a result, the overall time complexity of the algorithm is in $\Theta(n'm)$. Note that [13] independently uses a similar idea to obtain a fast reconciliation algorithm.

Proof of the Space Complexity. The size of the whole matrix $Cost(u, x)$ is in $\Theta(n'm)$, all other variables used in the algorithm are constant in size, and the space complexity is then immediate. □

Sketch of the Proof for the Correctness of the Algorithm

Given a tree T, define the height of a node $u \in V(T)$, denoted $h(u)$, as the length of the unique path from u to $r(T)$, and the height of T, denoted $h(T)$, is the maximal height over all its nodes.

Consider the edge (u_p, u) and the time stamp t examined at an iteration of the main loops (respectively in lines 3 and 4) in the algorithm. For any branch $(x_p, x) \in E_t(S')$, we now explain how the two loops compute $Cost(u, x)$ by considering all six events seperately. First, for the $\mathbb{S}$ and $\mathbb{D}$ events (lines 11 and 12 resp.), the consistency of the corresponding cost is ensured because for any child $u' \in \{u_1, u_2\}$ and any branch (x'_p, x') of S', $Cost(u', x')$ is previously computed during the bottom-up traversal of G. Second, for the $\varnothing$ and $\mathbb{SL}$ events (lines 17 and 19 resp.), the optimality is verified because for any branch $(x'_p, x') \in E_{t-1}(S')$, $Cost(u, x')$ is computed during the iteration for the time $(t-1)$ of S'.

The cases for the $\mathbb{T}$ and $\mathbb{TL}$ events use the optimal receivers for (u_p, u) and its two descendant edges, all located at time t. The bottom-up traversal of G implies that $Cost(u', x')$ is computed for all children $u' \in \{u_1, u_2\}$ and all branches ${(x'}_p, x') \in E_t(S')$. Thus, for a donor $(x_p, x) \in E_t(S')$, BestReceiver $(({u'}_p, u'), t, (x_p, x))$ computes (in linear time in the size of $E_t(S')$) the best receiver for the transfered edge ${(u'}_p, u')$. For the two descendant edges (u, u_1) and (u, u_2), the best receiver are respectively computed at lines line 13 and 14. For a $\mathbb{T}$ event (line 15) with (x_p, x) as the donor, the consistency of the corresponding cost is ensured following the same reasons as for a $\mathbb{D}$ event together with the availability of these two best receivers.

Considering a $\mathbb{TL}$ event with (x_p, x) as the donor, Property 1 implies that the minimal cost of mapping (u_p, u) onto an optimal receiver $({x'}_p, x') \in E_t(S') \setminus \{(x_p, x)\}$ corresponds to any of the five events considered in the third loop (line 5). Thus, when BestReceiver computes such an optimal receiver (line 22), its optimality is ensured together with that for the cost of a $\mathbb{TL}$ event (line 23) and the final cost (line 24).

This conclude the sketch to prove the correctness of Algorithm 1. Moreover, it is important to point out that a scenario in which a node $u \in V(G)$ has its two descendant edges (u, u_1) and (u, u_2) both transfered is implicitly considered by our combinatorial model of reconciliations. Indeed, given $u \in V(G)$ and a branch $(x_p, x) \in E_t(S')$ that is the last one of the sequence $\alpha(u_p, u)$, assume that this association corresponds to a $\mathbb{T}$ event for u, where u_1 is conserved by the donor (x_p, x) and u_2 is given to a receiver (see $\mathbb{T}$ event in Def. 3). Given that the first branch $\alpha_1(u, u_1)$ equals (x_p, x) in the sequence associated to u_1, a

reconciliation allows the next branch of this sequence (i.e. $\alpha_2(u, u_1)$) to be any branch in $E_t(S') \setminus \{(x_p, x)\}$ (see $\mathbb{TL}$ event in Def. 3).

B Simulated Data Sets

B.1 Simulated Species Trees

We generated a sets of 10 random ultrametric species trees with 100 species using a standard birth death process with PhyloGen [12] (the ratio of birth to death rate was 1.25). All species trees were normalized to a common height h, with time measured from the leaves of the species tree at $t = 0$ to its root at $t = h$. The time order of the internal nodes (speciation events), and hence S, was uniquely determined by the branch lengths of the tree.

B.2 Simulated $\mathbb{DTL}$ Scenarios

Starting with a single gene at time $t = h$ at the root of S, we generated evolutionary scenarios according to a Poisson process characterized by rates of duplication, transfer and loss. At time t, each extant gene underwent duplication with rate r_δ or loss at rate r_λ. Transfers to each branch of the species tree at time t occurred at rate r_τ, with the donor gene drawn uniformly from genes extant at time t except the branch considered.

Instances of the above Poisson process correspond to a completed gene tree G^o and a simulated reconciliation, denoted α_R, that includes a complete record of the $\mathbb{DTLS}$ events that gave rise to it. The gene tree G, obtained from G^o by removing the extinct subtrees of G^o, is used as the input to the parsimony algorithm.

Csűrös and Miklós recently provided estimates of the relative magnitude of duplication, transfer and loss rates in the domain of Archaea. For our purposes, there results can be summarized by the average ratio of 23% duplication, 1% gain, and 76% loss, and an approximate loss rate of 1.5 (assuming a tree with unit height). As many transfer scenarios do not leave behind a clear signal in the phylogenetic profile of a gene family, the gain rate can potentially underestimates the rate of transfer and overestimates the rate of duplication.

To explore a wide as possible set of parameters we chose two different ways of varying the rates of duplication, transfer, and loss.

In the first data set, denoted ds_1, we chose a fixed loss rate of $r_\lambda = 0.7$ (with tree height $h = 1$) and varied values of both r_δ and r_τ in the interval $[0.01, 0.35]$, choosing 11 values of each parameter, resulting in 11×11 sets of rates. This choice of parameters aims to simulate a relatively large phylogenetic time scale, corresponding to, e.g. a bacterial or archean phylum, with realistic loss rates, while making no assumption about the ratio of transfer and loss events, and only requiring $r_\delta + r_\tau \leq r_\lambda$. We generated 5 gene trees per species tree and per parameter set (6,050 in total).

In the second data set, denoted by ds_2, we chose to fix the ratio of $r_\delta + r_\tau$ to r_λ as follows: $r_\lambda/(r_\delta + r_\tau + r_\lambda) = 0.7$ (motivated by the results of Csűrös

and Miklós [2]). This choice of parameters aims at investigating the accuracy of parsimony on different phylogenetic scales, using 4 different tree heights $h = 0.2, 0.4, 0.8$ and 1.6. We varied the transfer rate $r_\tau \in [0, 0.3]$ in 11 steps (with consequently $r_\delta = 0.3 - r_\tau$). We generated 20 gene trees, per species tree and per rate parameter set (8,800 in total).

C Complementary Experimental Results

In some context, such as sequence orthology prediction, only the tagging of the nodes of G is important. Thus another way to account for errors is to compare the tagging inferred by a parsimony reconciliation with the tagging due to the real scenario. Fig. 8 shows error ratios when false positive and negative are judged on the fact that the internal nodes of the gene tree are assigned to the correct event they represent in the simulated scenario (i.e. one of $\mathbb{DTLS}$). It can be noted that both error levels for transfers decrease remarkably when accounting for transfers in this way (compare with Fig. 7).

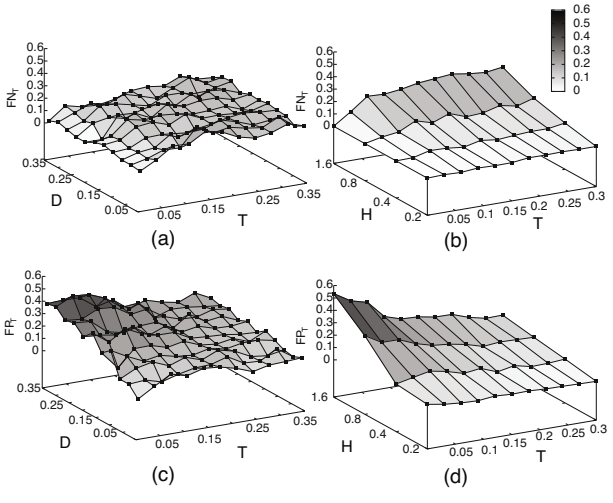


Fig. 8. Ratios of false negative (a-b) and false positive (c-d) for $\mathbb{T}$ events, for various combinations of heights, transfer and duplication rates, i.e. ds_1 (a-c) and ds_2 (b-d) when considering a transfer to be common to $\alpha_{R'}$ and α_P as soon as the same branch of G is transferred, i.e., without looking at place where the receiver is in S

Detecting Highways of Horizontal Gene Transfer

Mukul S. Bansal[1], J. Peter Gogarten[2], and Ron Shamir[1]

[1] The Blavatnik School of Computer Science, Tel-Aviv University, Israel
{bansal,rshamir}@tau.ac.il
[2] Department of Molecular and Cell Biology, University of Connecticut, Storrs, USA
gogarten@uconn.edu

Abstract. In a horizontal gene transfer (HGT) event a gene is transferred between two species that do not share an ancestor-descendant relationship. Typically, no more than a few genes are horizontally transferred between any two species. However, several studies identified pairs of species between which many different genes were horizontally transferred. Such a pair is said to be linked by a *highway of gene sharing*. We present a method for inferring such highways. Our method is based on the fact that the evolutionary histories of horizontally transferred genes disagree with the corresponding species phylogeny. Specifically, given a set of gene trees and a trusted rooted species tree, each gene tree is first decomposed into its constituent quartet trees and the quartets that are inconsistent with the species tree are identified. Our method finds a pair of species such that a highway between them explains the largest (normalized) fraction of inconsistent quartets. For a problem on n species, our method requires $O(n^4)$ time, which is optimal with respect to the quartets input size. An application of our method to a dataset of 1128 genes from 11 cyanobacterial species, as well as to simulated datasets, illustrates the efficacy of our method.

1 Introduction

Horizontal gene transfer (HGT) (also called lateral gene transfer) is an evolutionary process in which genes are transferred between two organisms that do not share an ancestor-descendant relationship. HGT plays an important role in bacterial evolution by allowing them to transfer genes across species boundaries. This transfer of genes between divergent organisms first became a research focus when the transfer of antibiotic resistance genes was discovered [1,2]. Microbiologists soon realized that the sharing of genes between unrelated species resulted in evolutionary patterns very different from those found in multi-cellular animals. Since then, the problem of detecting horizontally transferred genes has been extensively studied; see, for example, [3] for a review.

An important problem in understanding microbial evolution is to infer the HGT events (i.e., the donor and recipient of each HGT) that occurred during the evolution of a set of species. This problem is generally solved in a comparative-genomics framework by employing a parsimony criterion, based on the observation that the evolutionary history of horizontally transferred genes does not agree with the evolutionary history of the corresponding set of species. This is illustrated in Fig. 1(b). More formally, given a gene tree and a species tree, the *HGT inference problem* is to find the minimum number of HGT events that can explain the incongruence of the gene tree with the species tree.

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 109–120, 2010.

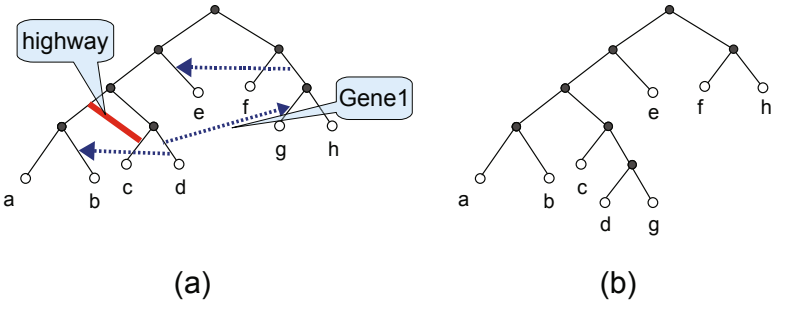


Fig. 1. Horizontal gene transfers and highways. (a) A species tree depicting three HGT events (dotted arcs) and a highway (bold red horizontal edge). The highway represents many individual HGT events all occurring between the same two (present-day or ancestral) species. (b) The corresponding gene tree for Gene1. Because of the HGT of Gene1 from species *d* into species *g*, the copy of that gene in *g* is most closely related to the one in *d*. Therefore, in the tree for Gene1, the species *g* appears next to *d*. (Here we assume that Gene1 was not transferred on the highway.)

The HGT inference problem is known to be NP-hard [4,5] and, along with some of its variants, has been extensively studied [5,6,7,8,9,10,11,12].

In general, one expects at most a few genes to have been horizontally transferred between any given pair of species. However, Beiko et al. [9] demonstrated that some pairs of species portray a multitude of horizontal gene transfer events. Such pairs are said to be connected by a *highway of gene sharing* [9]. Highways of gene sharing point towards major events in evolutionary history; well corroborated examples of this phenomenon are the uptake of endosymbionts into the eukaryotic host, and the many genes transferred from the symbiont to the hosts nuclear genome [13]. Recent proposals for evolutionary events that may be reflected in highways of gene sharing are the role of Chlamydiae in establishing the primary plastid in the Archaeplastida (red and green algae, plants and glaucocystophytes) [14], and the evolution of double membrane bacteria through an endosymbiosis between clostridia and actinobacteria [15]. Detecting these highways of gene sharing is thus an important biological problem and is crucial for inferring past symbiotic associations that shaped the evolution of organisms.

Given a rooted species tree, any two species (nodes) in it that are not related by an ancestor-descendant relationship define a *horizontal edge* connecting those two nodes. Any HGT event must take place along a horizontal edge in one of its two directions (see Fig. 1(a)). A horizontal edge along which an unusually large number of HGT events have taken place (say 10% of the genes) will be called a *highway of gene sharing* or simply a *highway*. The only existing method for detecting highways is the one employed originally by Beiko et al. [9]. That method takes as input a species tree and a set of gene (protein) trees, and computes, for each gene tree, the HGT events affecting that gene on the species tree. This is done by solving the HGT inference problem for each gene tree. The HGT events that are inferred in the HGT scenarios for a significant fraction of the gene trees are postulated as the highways. However, this approach suffers from several serious drawbacks. First, the HGT inference problem is NP-hard, and thus, difficult to solve exactly (and must often be solved using heuristics). Second, there

may be multiple (in fact, exponentially many) alternative optimal solutions to the HGT inference problem [10]. And third, when the rate of HGT is relatively high, there is little reason to expect that the number of HGT events should be parsimonious; i.e., the HGT inference problem, even if solved exactly and yielding only one optimal solution, may not infer the actual HGT events. In this work we propose an alternative approach to detecting highways that does not rely on inferring individual HGT events. Moreover, our formulation allows exact solution of the problem in polynomial time. Our method thus avoids all of the aforementioned pitfalls.

As in [9], the input to our method is a trusted rooted species tree for some set of species, and a set of gene trees on genes taken from those species. Since it is often difficult to accurately root gene trees, we assume that the input gene trees are unrooted. Our method is based on the observation that highways, by definition, affect the topologies of many gene trees. Thus, the idea is to combine the phylogenetic signals for HGT events from all the gene trees and use the combined signal to infer the highways, thereby avoiding the need to infer individual HGT events. We achieve this by employing a quartet decomposition of the gene trees. In particular, our method decomposes each gene tree into its constituent set of quartet trees and combines the quartet trees from all the gene trees to obtain a single weighted set of quartet trees. The intuition is that quartet trees that disagree with the species tree may indicate HGT events and thus the collective evidence from all quartet trees could pinpoint possible highways. The combined set of quartet trees is then analyzed against the given species tree to infer the highways of gene sharing. Decomposing the gene trees into quartet trees allows us to cleanly merge the phylogenetic signals for HGT events from all the different gene trees into a single summary signal, from which exact and efficient inference of the highways is possible.

To find highways, our method iteratively finds a horizontal edge that explains the largest fraction of inconsistent quartet trees. Essentially, for each (weighted) quartet tree inconsistent with the species tree, we identify the horizontal edges that can explain it by an HGT event (in either direction) along them. The horizontal edge that explains the most (normalized) inconsistency is proposed as a highway. (Normalization is needed since the structure of the species tree and the location of the horizontal edge in it influence the number of inconsistent quartet trees that edge may explain.) We give a dynamic programming algorithm that, given the weighted set of quartet trees, finds the best highway in $O(n^4)$ time. Since there may be $\Omega(n^4)$ input quartet trees, our algorithm is asymptotically optimal with respect to that input. In contrast, a naïve enumeration algorithm would require $O(n^6)$ time. Our efficient algorithms allow our method to be applied to fairly large datasets; for example, we can analyze a dataset of 1000 gene trees with 200 taxa within a day on a personal computer. We demonstrate the utility of our method on simulated data as well as on a dataset of 1128 genes from 11 cyanobacterial species [16], where its results match prior biological observations. For lack of space, proofs and some algorithmic details are omitted from this manuscript.

2 Basic Definitions and Preliminaries

Given a rooted or unrooted tree T, we denote its node set, edge set, and leaf set by $V(T)$, $E(T)$, and $Le(T)$ respectively. For the remainder of this paragraph, let T denote

a rooted tree. Given T, the root node of T is denoted by $rt(T)$. Given a node $v \in V(T)$, we denote the parent of v by $pa_T(v)$, its set of children by $Ch_T(v)$, and the subtree of T rooted at v by $T(v)$. We define $\leq_T$ to be the partial order on $V(T)$ where $u \leq_T v$ if v is a node on the path between $rt(T)$ and u. Given a non-empty subset $L \subseteq Le(T)$, we denote by $lca_T(L)$ the least common ancestor (LCA) of all the leaves in L in tree T. Given a rooted tree T, a *horizontal edge* on T is a pair of nodes $\{u, v\}$, where $u, v \in V(T)$, such that $u, v \neq rt(T)$, $u \not\leq v$, $v \not\leq u$, and $pa_T(u) \neq pa_T(v)$. We denote by $H(T)$ the set of all horizontal edges on T. Horizontal edges represent potential horizontal gene transfer events; the (directed) horizontal edge (u, v) represents the HGT event that transfers genetic information from the edge $(pa_T(u), u)$ to $(pa_T(v), v)$. Thus, the horizontal edge $\{u, v\}$ represents the HGT events (u, v) and (v, u). Also note that, while any particular HGT event is directional, we address the problem in which horizontal edges are undirected because highways can be responsible for transfer of genetic material in both directions. Throughout this work the term tree refers to a binary tree.

Our formulation and solution to the highway detection problem rely on the concept of quartets and quartet trees. A *quartet* is a four-element subset of some leaf set and a *quartet tree* is an unrooted tree whose leaf set is a quartet. The quartet tree with leaf set $\{a, b, c, d\}$ is denoted by $ab|cd$ if the path from a to b does not intersect the path from c to d. Given a rooted or unrooted tree T, let X be a subset of $Le(T)$ and let $T[X]$ denote the minimal subtree of T having X as its leaf set. We define the *restriction* of T to X, denoted $T|X$, to be the unrooted tree obtained from $T[X]$ by suppressing all degree-two nodes (including the root, if T is rooted). We say that a quartet tree Q is *consistent* with a tree T if $Q = T|\,Le(Q)$, otherwise Q is *inconsistent* with T. Observe that, given any T and any quartet $X = \{a, b, c, d\}$ from $Le(T)$, X induces exactly one quartet tree in T, that is, the quartet tree $T|X$. Also observe that this quartet tree must have one of three possible topologies: $ab|cd$, $ac|bd$, or $ad|bc$.

3 Detecting Highways

Our goal is to detect the highways of gene sharing in the evolutionary history of a set of species $\mathbb{S}$. To that end, we are given a set of unrooted gene trees $\{T_1, \ldots, T_m\}$, and a rooted species tree S showing the evolutionary history of $\mathbb{S}$. Thus, $Le(S) = \mathbb{S}$, and $Le(T_i) \subseteq \mathbb{S}$ for $1 \leq i \leq m$. The idea is to infer the highways by inspecting the differences in the topologies of the gene trees compared to the species tree. The *highway detection problem* can thus be stated as follows: Given a species tree S and a collection of gene trees, find the horizontal edges on S that correspond to highways.

Throughout this manuscript, S denotes the given species tree, and n denotes the number of species in the analysis, i.e., $n = |\,Le(S)|$.

Our solution to the highway detection problem is based on decomposing each input gene tree T into its constituent set of $\binom{|\,Le(T)|}{4}$ quartet trees. To understand the intuition behind using quartet trees, consider the scenario depicted in Fig. 2. The tree on the left is a species tree on six species, along with two HGT events of two different genes. Consider the HGT event (C, E) that transfers Gene1. This HGT event causes the topology of the corresponding gene tree to deviate from the topology of the species tree. Essentially, according to the standard subtree transfer model of horizontal gene transfer (see,

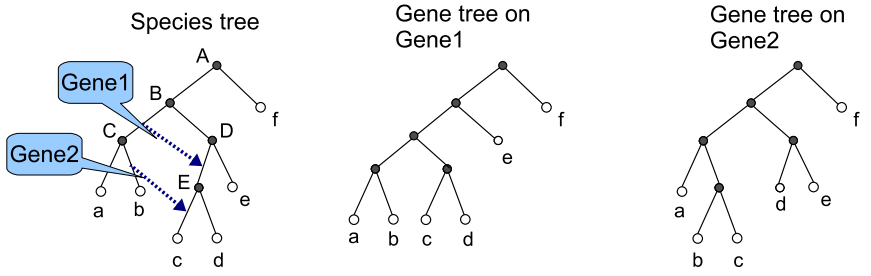


Fig. 2. The tree on the left is a species tree showing the evolutionary history of a set of six species. Two HGT events (C, E) and (b, c), shown by the dotted arcs, are also depicted on this species tree. The two other trees show the evolutionary histories of Gene1 and Gene2.

e.g., [17,9,8]), this HGT event causes the subtree rooted at node E to be pruned and then regrafted along the edge (B, C), as shown in the figure. Let us decompose both trees into their constituent set of quartet trees: Each tree generates $\binom{6}{4} = 15$ quartet trees. Note that four of the fifteen quartets induce different quartet trees in the two trees; in the gene tree, these appear as $ac|ef$, $ad|ef$, $bc|ef$ and $bd|ef$. In general, different HGT events produce gene trees with different sets of inconsistent quartet trees. Thus, given the species tree, and the set of the four inconsistent quartet trees from the gene tree on Gene1, we could have inferred the HGT event (C, E) that affected Gene1.

3.1 The Method in Detail

Our method proceeds iteratively, inferring one highway per iteration, as follows.

Step 1: Decompose each input gene tree T into its constituent set of $\binom{|Le(T)|}{4}$ quartet trees, and combine the quartet trees from the different gene trees into a single weighted set, Φ, of quartet trees.
Step 2: Remove from Φ all those quartet trees that are consistent with S.
Step 3: Compute the HGT score of each edge in $H(S)$. This HGT score for an edge is computed based on Φ, and is explained in detail below.
Step 4: Select the highest scoring horizontal edge as a highway.
Step 5: Remove from Φ all those quartet trees that are explained by the proposed highway, and go to Step 3 to start the next iteration.

The (raw) HGT score of a horizontal edge is simply the total weight of the quartet trees from Φ that are explained by a HGT along that edge (in either direction). Thus, this raw score of a horizontal edge captures the number of quartet trees from the input gene trees that support horizontal gene transfer along that edge. However, not all horizontal gene transfers affect the same number of quartet trees. Consider the example shown in Fig. 2. As seen previously, the HGT event (C, E) causes four of the quartet trees in the corresponding gene tree to become inconsistent. Consider the HGT event (b, c) that transfers Gene2. This HGT event causes ten of the quartet trees in the gene tree built on Gene2 (shown on the right in Fig. 2) to become inconsistent; these are $ad|bc$, $ae|bc$, $af|bc$, $ac|de$, $ac|df$, $ac|ef$, $bc|de$, $bc|df$, $bc|ef$ and $de|cf$. Thus, considering only the

raw scores of the horizontal edges would lead to overestimation of the quantity of HGT along certain horizontal edges and underestimation of this quantity for other horizontal edges, leading to incorrect inference of highways.

To overcome this bias we modify the score of each horizontal edge by dividing its raw score by a normalization factor: The maximum number of distinct quartet trees that could be explained by a horizontal gene transfer (in either direction) along that edge. More precisely, let Ψ be the set of all possible quartet trees on the leaf set $Le(S)$. Given a horizontal edge $\{u, v\}$, let Q_1 denote the set of quartet trees in Ψ that become consistent due to the HGT event (u, v), and let Q_2 denote the set of quartet trees in Ψ that become consistent due to the HGT event (v, u). The normalization factor for $\{u, v\}$ is defined to be $|Q_1 \cup Q_2|$. After normalization, the HGT scores of all horizontal edges can be directly compared to one another.

The number of iterations in the method can either be fixed at the beginning or, preferably, be decided on the fly, based on the distribution of the horizontal edge scores computed in the current iteration.

4 The Highway Scoring Problem

This iterative quartet based method involves four computational steps: (i) Computing the initial set of weighted quartet trees from the gene trees, (ii) removing the quartet trees that are consistent with S, (iii) computing the (normalized) HGT score of each edge in $H(S)$, and (iv) identifying and removing those quartet trees that are explained by the proposed highway. It is relatively straightforward to show (details omitted for brevity) that step (i) can be executed in $O(mn^4)$ time, where m is the number of input gene trees, and steps (ii) and (iv) can be executed in $O(n^4)$ time. The main computational challenge here is (iii), i.e., computing the (normalized) HGT score of each horizontal edge. In this section we focus on this main problem.

Given a rooted species tree S and a set Φ of weighted quartet trees (that are inconsistent with S) on the leaf set $Le(S)$, the *Highway Scoring (HS) problem* is to find the (normalized) HGT score of each edge in $H(S)$.

The naïve way to solve the HS problem would be to consider each edge in $H(S)$ one-at-a-time and to check which of the quartet trees from Φ are explained by that edge. Checking whether a quartet tree is explained by a horizonal edge can be accomplished in $O(1)$ time. Since there are $\Theta(n^2)$ candidate horizontal edges and $O(n^4)$ quartet trees in Φ, the complexity of computing just the raw score of each horizontal edge is still $O(n^6)$. In this section we show that the HS problem can be solved in $O(n^4)$ time. The time complexity of our algorithm is thus optimal.

Recall that each horizontal edge actually represents two HGT events. We denote the set of all these HGT events on S by $\overrightarrow{H}(S)$. Thus, for any horizontal edge $\{u, v\} \in H(S)$, there are two HGT events (u, v) and (v, u) in $\overrightarrow{H}(S)$.

Given a horizontal edge $\{u, v\}$, if Q_1 and Q_2 denote the sets of quartet trees that are explained by the HGT events (u, v) and (v, u) respectively, then, the raw score of $\{u, v\}$ is $|Q_1 \cup Q_2|$, which is $|Q_1| + |Q_2| - |Q_1 \cap Q_2|$. First, in Section 4.1, we show how to compute the raw score of each horizontal event (i.e., how to compute $|Q_1|$ and $|Q_2|$), and then, in Section 4.2, we show how to compute $|Q_1 \cap Q_2|$ and thus obtain the raw

scores of horizontal edges. In Section 4.2, we also show how to reuse these algorithms to compute the normalization factor for each horizontal edge.

4.1 Computing the Raw Scores of HGT Events

For any given quartet tree $Q \in \Phi$, there may be several HGT events from $\overrightarrow{H}(S)$ that could explain Q; we denote this set of HGT events by $\overrightarrow{H}(S, Q)$. Since S is fixed, throughout the remainder of this work we will abbreviate $H(S)$, $\overrightarrow{H}(S)$ and $\overrightarrow{H}(S, Q)$ to H, $\overrightarrow{H}$ and $\overrightarrow{H}(Q)$ respectively. Our algorithm relies on an efficient characterization of the HGT events that can explain a given quartet. This characterization appears in the next lemma; but first, we need some additional definitions and notation.

Notation and Definitions. We denote the raw score of an HGT event $(u, v) \in \overrightarrow{H}$ by $RS(u, v)$. Given any two nodes $p, q \in V(S)$, let $p \rightarrow q$ denote the path between them in S, and let $V(p \rightarrow q)$ denote the set of nodes on this path (including p and q). A *subtree-path (SP) pair* on S is a pair $\langle S(v), p \rightarrow q\rangle$, where $v, p, q \in V(S)$, such that the subtree $S(v)$ and the path $p \rightarrow q$ are node disjoint and none of the nodes in $p \rightarrow q$ is an ancestor or descendant of v. Given an SP pair $\sigma = \langle S(v), p \rightarrow q\rangle$, the set of all HGT events (u, v) from $\overrightarrow{H}$ such that $u \in S(v)$ and $v \in V(p \rightarrow q)$ is denoted by $\overrightarrow{H}(\sigma)$. Similarly, a *subtree-complement-path (SCP) pair* on S is a pair $\langle S(v), p \rightarrow q\rangle$, where $v, p, q \in V(S)$, such that $V(p \rightarrow q) \subseteq V(S(v))$. We define $\overline{V}(S(v))$ to be the set $[V(S) \setminus V(S(v))] \cup \{v\}$. Given an SCP pair $\sigma = \langle S(v), p \rightarrow q\rangle$, the set of all HGT events (u, v) from $\overrightarrow{H}$ such that $u \in \overline{V}(S(v))$ and $v \in V(p \rightarrow q)$ is denoted, as before, by $\overrightarrow{H}(\sigma)$. If σ is an SCP pair, then we say that $S(v)$ is the *subtree-complement* of σ, and it refers to the subtree of S induced by $\overline{V}(S(v))$.

Lemma 1. *Given any quartet tree $Q \in \Phi$, there exist four SP/SCP pairs, denoted $\sigma_1, \sigma_2, \sigma_3, \sigma_4$, such that $\overrightarrow{H}(Q) = \overrightarrow{H}(\sigma_1) \cup \overrightarrow{H}(\sigma_2) \cup \overrightarrow{H}(\sigma_3) \cup \overrightarrow{H}(\sigma_4)$. Moreover, the four sets $\overrightarrow{H}(\sigma_1), \overrightarrow{H}(\sigma_2), \overrightarrow{H}(\sigma_3)$ and $\overrightarrow{H}(\sigma_4)$ are pairwise disjoint.*

In fact, after an initial $O(|\,Le(S)|)$ preprocessing step, we can compute the four SP/SCP pairs for any given quartet tree in $O(1)$ time. Our algorithm performs a nested tree traversal of S. Before we begin this nested tree traversal we (i) perform a pre-processing step, which precomputes certain values on the tree S, and (ii) perform a tree decoration step during which we decorate the nodes of S with information about the four SP/SCP pairs for each quartet tree in Φ. Next we describe these two steps in detail.

The preprocessing step. The first step in the algorithm is to preprocess the tree S so that, given any two nodes from $V(S)$, we can compute their LCA within $O(1)$ time [18]. This preprocessing step also allows us to label the nodes of S in such a way that given any two nodes $u, v \in V(S)$ we can check if $v \in V(S(u))$ in $O(1)$ time. We also associate with each $v \in V(S)$ a counter, denoted by $counter_v$, initialized to zero, and a set $path_v$ initialized to be empty.

Decorating the tree. For each quartet tree $Q \in \Phi$, we identify the four SP/SCP pairs $\sigma_1 = \langle S(v_1), p_1 \rightarrow q_1\rangle$, $\sigma_2 = \langle S(v_2), p_2 \rightarrow q_2\rangle$, $\sigma_3 = \langle S(v_3), p_3 \rightarrow q_3\rangle$, and $\sigma_4 = \langle S(v_4), p_4 \rightarrow q_4\rangle$. One of the end points of the path in each of these SP/SCP

pairs must be a leaf node (see proof of Lemma 1). By convention, we let the q_is, for $i \in \{1,2,3,4\}$, denote these leaf nodes. We mark these four paths on S as follows: For each $i \in \{1,2,3,4\}$, if σ_i is an SP pair then add the triple (Q, v_i, SP) to the sets $path_{q_i}$ and $path_{pa(p_i)}$; if σ_i is an SCP pair, add the triple (Q, v_i, SCP) to the sets $path_{q_i}$ and $path_{pa(p_i)}$. Here SP/SCP is included as a binary label to indicate the type of the pair.

The tree decoration step, described above, marks the endpoints of the four paths in the SP/SCP pairs of any quartet. Our algorithm performs a post-order traversal of S and, at each node v, calls the procedure *Augment*(v) described below. This procedure marks the corresponding subtrees/subtree-complements for all the paths that appear in the set $path_v$, and computes a value val_u at each $u \in V(S) \setminus \{rt(S)\}$. This value val_u is the weight of all quartet trees Q from Φ such that (i) $(Q, x, \Gamma) \in path_v$ and (ii) if Γ is SP then $u \in V(S(x))$, and, if Γ is SCP then $u \in \overline{V}(S(x))$. The reason for computing these val_u's becomes clear in the context of Lemma 2.

Procedure *Augment*(v) $\quad \{v \in V(S)\}$
1: **for** each $x \in V(S)$ **do**
2: Set $counter_x$ to 0.
3: **for** each triple $(Q, y, \Gamma) \in path_v$ **do**
4: **if** Γ is SP **then**
5: Increment $counter_y$ by the weight of Q.
6: **if** Γ is SCP **then**
7: Increment $counter_{rt(S)}$ by the weight of Q and, decrement $counter_{y_1}$ and $counter_{y_2}$ by the weight of Q, where $\{y_1, y_2\} = Ch(y)$.
8: **for** each $u \in V(S) \setminus \{rt(S)\}$ **do**
9: Set val_u to $\sum_{x \in V(rt(S) \to u)} counter_x$.

Our algorithm is based on the following key lemma.

Lemma 2. *Suppose S has been decorated and procedure Augment(v) has been executed for some $v \in V(S)$. Consider any $(u, v) \in \overrightarrow{H}$.*

1. *If $v \in Le(S)$, then $RS(u, v) = val_u$.*
2. *If $v \notin Le(S)$, then $RS(u, v) = RS(u, v_1) + RS(u, v_2) - val_u$, where $v_1, v_2 \in Ch(v)$.*

Nested tree traversal. Once the pre-processing and tree decoration steps have been executed, the algorithm performs a nested tree traversal of S and computes the raw score of each HGT event from $\overrightarrow{H}$ according to Lemma 2. More formally, the algorithm proceeds as follows:

Algorithm *ComputeScores*
1: **for** each $v \in V(S)$ in a post-order traversal of S **do**
2: Perform procedure *Augment*(v).
3: **for** each $u \in V(S) \setminus \{rt(S)\}$ **do**
4: **if** (u, v) is a valid HGT event, i.e., $(u, v) \in \overrightarrow{H}$, **then**
5: **if** $v \in Le(S)$ **then**
6: Set $RS(u, v)$ to be val_u.
7: **else**
8: Set $RS(u, v)$ to be $RS(u, v_1) + RS(u, v_2) - val_u$, where $v_1, v_2 \in Ch(v)$.

The preprocessing step, tree decoration step, and Algorithm *ComputeScores* require $O(n)$, $O(\Phi)$, and $O(n^2 + |\Phi|)$ time respectively. Thus, we have the following lemma.

Lemma 3. *The raw scores of all HGT events in* $\overrightarrow{H}$ *can be computed in* $O(n^2 + |\Phi|)$ *time.*

4.2 Raw Scores of Horizontal Edges and Normalization Factors

Our goal now is to compute the raw score of each horizontal edge in H. For any edge $\{u, v\} \in H$, let its raw score be denoted by $RS\{u, v\}$. Observe that $RS\{u, v\} = RS(u, v) + RS(v, u) - common\{u, v\}$, where $common\{u, v\}$ is the total weight of the quartet trees that are counted in both $RS(u, v)$ and $RS(v, u)$. A variant of the algorithm described above enables us to compute the value $common\{u, v\}$ for each horizontal edge $\{u, v\} \in H$ in $O(n^2 + |\Phi|)$ time. Thus, we can compute the raw score of each horizontal edge in $O(n^2 + |\Phi|)$ time.

Recall that the normalization factor for a horizontal edge is simply the maximum number of distinct quartet trees that could be explained by that edge. Thus, we can reuse the algorithm that computes the raw scores of horizontal edges by running it on a set that contains all the possible $3 \times \binom{n}{4}$ quartet trees, each with weight 1. Thus, we have the following theorem.

Theorem 1. *The highway scoring problem can be solved in* $O(n^4)$ *time.*

5 Experimental Analysis

Cyanobacterial dataset. We first applied our method to a dataset of 1128 genes from 11 cyanobacterial species [16]. The existence of a highway on this set of species was postulated in [16,19] and thus this dataset serves for method validation. Each of the 1128 gene trees had at least nine of the 11 species (see [16] for further details). As the trusted species tree, shown in Fig. 3, we used the rooted tree constructed on the 16S ribosomal RNA sequence from these species [20]. To account for uncertainty in the topologies of the gene trees, for each gene tree we used only those quartet trees that were present in at least 80% of the bootstrap replicates of that gene tree [16]. Our final weighted set had 799 different quartet trees with a total weight of 214,729. The total number of inconsistent quartet trees was 469 and their total weight was 23,042. There were 118 candidate horizontal edges. Fig. 4(a) shows the histogram of the normalized scores for these horizontal edges in the first application of the algorithm. The highest scoring edge is extremely well separated from the next candidate in terms of the scores (Fig. 4(a)). It is marked in Fig. 3. A priori, it is surprising that this highway connects two different genera that are distinguished by different light harvesting machineries, but the high rate of transfer between marine *Synechococcus* and *Prochlorococcus* has been previously observed and discussed [16,19]. The discovered highway thus matches perfectly with prior biological observations.

We performed further analysis of this dataset with the aim of discovering other novel highways. In the second iteration, our method proposes the second highway shown in Fig. 3. Though the normalized score of this highway is much smaller (179.4) than that of the first highway (508.6), it is significantly higher than the scores of the other

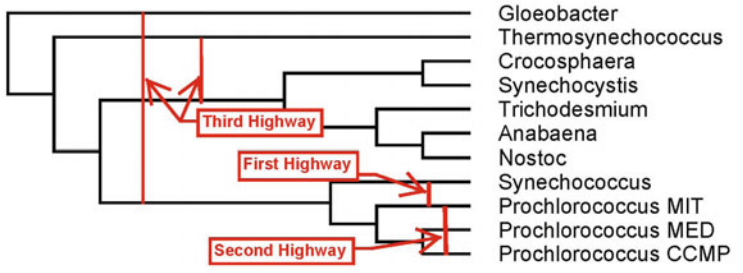
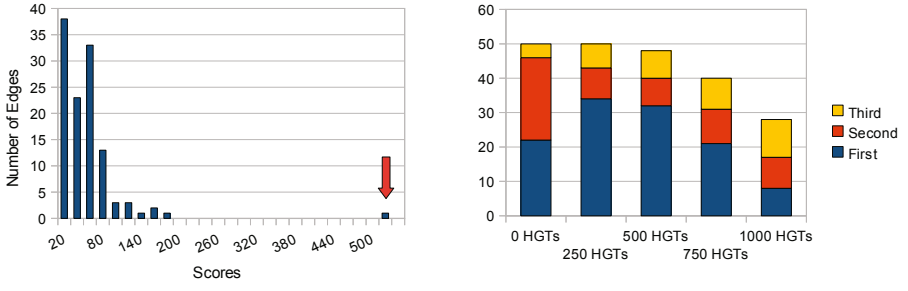


Fig. 3. The 16SrRNA tree on the 11 cyanobacterial species, with detected highways marked



(a) (b)

Fig. 4. Highway detection statistics. (a) histogram of edge scores for the first highway on the cyanobacterial dataset. (b) Simulation results: the number of times (out of 50) an implanted highway edge is detected in simulated datasets with varying levels of noise.

edges (only two other edges have scores above 100). Like the first, this second highway also represents transfer between the small marine cyanobacteria, likely mediated by cyanophage. Further analysis also suggests the presence of a third highway (normalized score: 157.2, second-highest score: 97.7) along one of two possible horizontal edges, shown in Fig. 3. These two horizontal edges produce the same unrooted tree and are hence indistinguishable in our quartet-based model.

Simulated datasets. We used simulations to test the effect of HGT abundance on the ability to infer highways. Each simulated dataset consisted of a randomly generated species tree on 25 taxa and 1000 gene trees. For the experiment, we randomly chose a highway on the species tree, and randomly assigned 10% of the 1000 genes as having been transferred along this highway, with equal probability for each transfer direction. Next, we simulated varying levels of "noise" on the species tree in the form of random HGT events, each affecting a gene sampled at random without replacement (including the genes that were transferred on the chosen highway). We simulated noise at five different levels: 0 HGTs (i.e., no noise), 250 HGTs, 500 HGTs, 750 HGTs and 1000 HGTs. For each noise level, we created 50 different datasets (different species tree, highway, and random HGTs) and measured the number of times (out of 50) that the implanted highway is reported as one of the top three highest scoring edges by our method. As shown in Fig. 4(b), our method tends to identify the planted highway, even

in datasets with high levels of noise; for instance, when there are 750 random HGTs (7.5 times the number of highway transfers) only 20% of the implanted highways were not included among the top three edges. By 1000 HGTs, performance has deteriorated. Interestingly, even when there is no noise in the data, the method does not always identify the implanted highway as its top-scoring edge. This is probably because of the way we normalize the scores. Our normalization factor is independent of direction, while the actual HGT events that take place along the highway are directed. This can cause some biases, which can make the normalized score of some nearby horizontal edges slightly higher than the score of the actual highway. Still, as the experiment demonstrates, even with relatively high levels of noise our algorithm usually brings to the top the correct highway, and further analysis of the top candidates can reveal the true highway.

6 Discussion

In this paper we addressed the problem of inferring highways of gene sharing, a fundamental problem in understanding the effects and dynamics of horizontal gene transfer, and crucial to inferring past symbiotic associations that shaped the evolution of organisms. Our new systematic approach and efficient algorithms for the highway detection problem facilitate accurate and in-depth analysis of relatively large datasets. The method detects the fingerprints of highways by looking at combined data from all the input gene trees summarized as quartet tree counts. We thus avoid the computational burden and uncertainty of inferring individual HGT events for each gene. Our experimental results demonstrate that our method is effective at detecting highways and is robust to noise in the data. We were able to identify the established highway in the cyanobacterial dataset, and our analysis identified two additional putative highways. As the experiments on the simulated data indicate, even in the presence of substantial noise our method reports the true highway among the few top-scoring edges.

While we demonstrate the effectiveness of our method, it still has some limitations. For example, if the dataset contains two highways that are closely related to one another then the method may only detect one of them (since many of the inconsistent quartet trees from one highway may also support the other highway). More generally, while the normalized scoring of the horizontal edges that we propose takes care of the variation in the number of candidate quartets of different edges, perhaps a better normalization could highlight the correct highways even more strongly. Similarly, the quartic running time is quite high and may be limiting for very large datasets. Further testing of the method in both simulations and on real datasets is also needed, and it might be instructive to compare it to alternative non-quartet-based methods. Finally, a statistical analysis of highway and HGT score distributions could provide more quantifiable significance, which we still lack.

Acknowledgements. MSB was supported in part by a postdoctoral fellowship from the Edmond J. Safra Bioinformatics program at Tel-Aviv University. JPG was supported in part by NSF grant DEB 0830024, the Edmond J. Safra Bioinformatics Program, and a fellowship from the Fulbright Program. RS was supported in part by the Israel Science Foundation (Grant 802/08) and by the Raymond and Beverly Sackler Chair in Bioinformatics.

References

1. Ochiai, K., Yamanaka, T., Kimura, K., Sawada, O.: Inheritance of drug resistance (and its transfer) between Shigella strains and Between Shigella and E.coli strains. Hihon Iji Shimpor 1861, 34–46 (1959) (in Japanese)
2. Gray, G., Fitch, W.: Evolution of antibiotic resistance genes: the DNA sequence of a kanamycin resistance gene from Staphylococcus aureus. Mol. Biol. Evol. 1(1), 57–66 (1983)
3. Zhaxybayeva, O.: Detection and Quantitative Assessment of Horizontal Gene Transfer. In: Horizontal gene Transfer: Genomes in Flux. Methods in Molecular Biology, vol. 532, pp. 195–213. Humana Press (2009)
4. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Annals of combinatorics 8(4), 409–423 (2005)
5. Hallett, M.T., Lagergren, J.: Efficient algorithms for lateral gene transfer problems. In: RECOMB, pp. 149–156 (2001)
6. Boc, A., Makarenkov, V.: New efficient algorithm for detection of horizontal gene transfer events. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 190–201. Springer, Heidelberg (2003)
7. Nakhleh, L., Warnow, T., Linder, C.R.: Reconstructing reticulate evolution in species: theory and practice. In: RECOMB, pp. 337–346 (2004)
8. Nakhleh, L., Ruths, D.A., Wang, L.S.: Riata-hgt: A fast and accurate heuristic for reconstructing horizontal gene transfer. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 84–93. Springer, Heidelberg (2005)
9. Beiko, R.G., Harlow, T.J., Ragan, M.A.: Highways of gene sharing in prokaryotes. Proc. Natl. Acad. Sci. USA 102(40), 14332–14337 (2005)
10. Than, C., Ruths, D.A., Innan, H., Nakhleh, L.: Confounding factors in hgt detection: Statistical error, coalescent effects, and multiple solutions. Journal of Computational Biology 14(4), 517–535 (2007)
11. Jin, G., Nakhleh, L., Snir, S., Tuller, T.: Parsimony score of phylogenetic networks: Hardness results and a linear-time heuristic. IEEE/ACM Trans. Comput. Biology Bioinform. 6(3), 495–505 (2009)
12. Boc, A., Philippe, H., Makarenkov, V.: Inferring and Validating Horizontal Gene Transfer Events Using Bipartition Dissimilarity. Syst. Biol. 59(2), 195–211 (2010)
13. Gary, M.W.: Origin and evolution of organelle genomes. Curr. Opin. Genet. Dev. 3, 884–890 (1993)
14. Huang, J., Gogarten, J.: Did an ancient chlamydial endosymbiosis facilitate the establishment of primary plastids? Genome Biology 8(6), R99 (2007)
15. Lake, J.A.: Evidence for an early prokaryotic endosymbiosis. Nature 460, 967–971 (2009)
16. Zhaxybayeva, O., Gogarten, J.P., Charlebois, R.L., Doolittle, W.F., Papke, R.T.: Phylogenetic analyses of cyanobacterial genomes: Quantification of horizontal gene transfer events. Genome Research 16(9), 1099–1108 (2006)
17. Hein, J.: Reconstructing evolution of sequences subject to recombination using parsimony. Mathematical Biosciences 98(2), 185–200 (1990)
18. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
19. Zhaxybayeva, O., Doolittle, W.F., Papke, R.T., Gogarten, J.P.: Intertwined Evolutionary Histories of Marine *Synechococcus* and *Prochlorococcus marinus*. Genome Biol. Evol. 1, 325–339 (2009)
20. Fournier, G.P., Gogarten, J.P.: Rooting the Ribosomal Tree of Life. Mol. Biol. Evol. (2010), (Epub ahead of print) msq057

On Exploring Genome Rearrangement Phylogenetic Patterns

Andrew Wei Xu

School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL)
EPFL IC LCBB, Station 14
CH-1015 Lausanne, Switzerland
wei.xu@epfl.ch

Abstract. The study of genome rearrangement is much harder than the corresponding problems on DNA and protein sequences, because of the occurrences of numerous combinatorial structures. By explicitly exploring these combinatorial structures, the recently developed adequate subgraph theory shows that a family of these structures, adequate subgraphs, are informative in finding the optimal solutions to the rearrangement median problem. Its extension gives rise to the tree scoring method GASTS, which provides quick and accurate estimation of the number of rearrangement events, for any given topology. With a similar motivation, this paper discusses and provides solid but somewhat initial results, on combinatorial structures that are informative in phylogenetic inference. These structures, called rearrangement phylogenetic patterns, provide more insights than algorithmic approaches, and may provide statistical significance for inferred phylogenies and lead to efficient and robust phylogenetic inference methods on large sets of taxa.

We explore rearrangement phylogenetic patterns with respect to both the breakpoint distance and the DCJ distance. The latter has a simple formulation and well approximates other edit distances. On four genomes, we prove that a contrasting shared adjacency, where a gene forms one adjacency in two genomes and a different adjacency in the other two genomes, is a rearrangement phylogenetic pattern. Phylogenetic inferences based on the numbers of this pattern, are very accurate and robust against short internal edges, tested on 55,000 datasets simulated by random inversions. Further analysis shows that the numbers of this pattern well explain the variations in the number of rearrangement events over different topologies.

1 Introduction

Genome rearrangement information, revealed in comparison of gene orders from related species, has been used for phylogenetic study for decades [11,8,3]. These methods, under the parsimony framework, make inferences of tree topologies and gene orders on internal nodes simultaneously. Such a detailed consideration gives extremely accurate phylogenetic inferences and provides good estimation

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 121–136, 2010.

of ancestral genome architectures [16,9,13,1]. However, due to the occurrences of numerous combinatorial structures, these problems are generally very difficult to solve [4,10,5,14].

We recently developed the *adequate subgraph theory* [17,15] in studying the rearrangement median problem. The theory explicitly explores the combinatorial structures and proves that a family of these structures, *adequate subgraphs*, are informative in finding optimal solutions. This theory allows us to solve the rearrangement median problem using a decomposition approach: detect the occurrence of an adequate subgraph, decompose the original problem into two smaller subproblems, and repeat this iteratively. In the case that no known adequate subgraphs can be detected, a branch-and-bound method or heuristics can be used to find optimal or heuristic solutions, respectively. An extension of this theory gives rise to a tree scoring method, *GASTS*, which minimizes the number of rearrangement events needed to explain a given data for a given tree topology [16] using a local optimization approach. GASTS scores very quickly and accurately, with typical errors within 0.1%, on trees with up to thousands of genomes and with thousands of genes in each genome.[1] To infer the phylogeny, we just need to find the topology with the smallest GASTS score.

The goal of this paper is to explore *rearrangement phylogenetic patterns* which are the combinatorial structures containing phylogenetic information. Under the parsimony framework, a combinatorial structure is said to be a rearrangement phylogenetic pattern, if we can prove that this structure always gives smaller scores (numbers of rearrangements or summarized distances over the tree) on one fixed topology but not on the others. In other words, we say a rearrangement phylogenetic pattern differentiates tree topologies. The topology with the smaller score is called the *preferred topology*.

We want to investigate whether and to what degree, we can make phylogenetic inferences by only inspecting the rearrangement phylogenetic patterns presented in the given data. Long-term goals of this topic are to analyze the probabilistic properties of these rearrangement phylogenetic patterns, to derive statistical tests for the significance of inferred results, and then to design efficient and robust phylogenetic inference methods for large numbers of taxa.

This paper focuses on problems with four signed genomes. Although only genomes with circular chromosomes are discussed, our results also can be applied to genomes with multiple linear chromosomes. We are aware that, given the computational difficulties and the numerous combinatorial structures in the rearrangement problems, this paper only presents some initial results. Hence it is not the goal of this paper to design better methods than the full parsimony methods, although our new methods do have very good accuracies.

In the rest of the section, we briefly introduce common pairwise distance measures, as they are the basis of most phylogenetic methods discussed in this paper. In Section 2, we introduce the full parsimony tree scoring method GASTS, two distance based methods and four-point metric, and phylogenetic invariants

[1] A study of the performance of GASTS can be found at http://sites.google.com/site/andrewweixu/gasts.

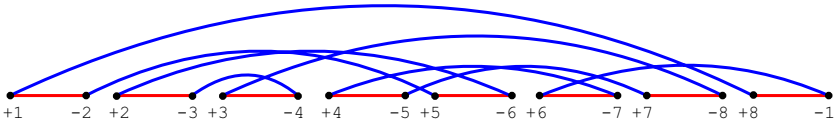


Fig. 1. The breakpoint graph of two circular genomes (1,-8,-3,4,7,5,2,6) and (1,2,3,4,5,6,7,8), in blue and red colors respectively. There are 8 genes in each genome, and there is one color-alternating cycle and no common adjacency in the breakpoint graph.

methods. In Section 3, we give results on what structures are and are not rearrangement phylogenetic patterns, and then introduce two phylogenetic scoring functions and their inference methods. In Section 4, we present comparison results of various phylogenetic inference methods.

1.1 Pairwise Distance Measures

In measuring the genomic distance between two genomes, there are two types of measures: observational distances, such as the breakpoint distance; edit distances, which are the smallest numbers of operations needed to transform one genome into the other given a set of allowed operations, such as the inversion distance [7], its generalization—the HP distance [6] and a further generalization—the DCJ distance [18,2]. This paper focus on the breakpoint distance and the DCJ distance, as the latter well approximates the other edit distances.

Breakpoint graph. The breakpoint graph is an important graph tool to study a pair of genomes. For each gene g, a pair of vertices $-g$ and $+g$ are used to represent its two endpoints, also called *extremities*. For each genome, *adjacencies*, which are pairs of extremities from neighboring genes, are represented by edges connecting the corresponding vertices; each genome is assigned a different color, and all edges from each genome are given that color. The breakpoint graph naturally decomposes into a set of *color-alternating cycles*, and we use c to denote their total number. Fig. 1 shows the breakpoint graph for the two genomes $(1, -8, -3, 4, 7, 5, 2, 6)$ and $(1, 2, 3, 4, 5, 6, 7, 8)$.

The breakpoint distance. A breakpoint occurs when an adjacency exists in one genome but not in the other. The number of breakpoints between two genomes is their breakpoint distance. The breakpoint distance has an intuitive explanation. To transform one genome into the other, we can cut the first genome into small fragments, rearrange and paste them back into the second genome. And the breakpoint distance is just the minimum number of cuts we need in this process. If n denotes the number of genes in each genome and a denotes the number of adjacencies shared by both genomes, then the breakpoint distance is simply $d_{BP} = n - a$.

The DCJ distance. The DCJ operation was first introduced in [18] and further studied in [2]. The DCJ distance provides a general rearrangement framework, including all common rearrangement operations, and a simple mathematical formula: $d_{DCJ} = n - c$.

2 Full Parsimony Methods, Distance Based Methods and Phylogeny Invariants

In this section, we briefly introduce three types of existing phylogenetic inference methods. The first type of methods are the full parsimony methods, which infer phylogeny and gene orders of internal nodes simultaneously, under either the breakpoint distance or some edit distance. In this paper, we consider the DCJ distance, as there exists a very quick and accurate tree scoring method, GASTS. Phylogenetic methods based on GASTS have been shown to be very accurate [16]. The second type of methods are distance-based methods. Given a data with n extant species and a distance measure, we can easily convert these sequences or gene orders into a n by n distance matrix with $\frac{n(n-1)}{2}$ independent variables, and reconstruct the phylogeny from this matrix. This kind of method has a charm of simplicity. To another end, the third type of methods, phylogeny invariants, explicitly explore various patterns and their occurrence frequencies in the data. Given an evolution model, algebraic relations (invariants) of the probabilities of these patterns can be derived, and are used to make phylogenetic inferences.

2.1 Full Parsimony Methods

The full parsimony methods are computationally challenging. But due to the recent development of the adequate subgraph theory and its extension, the scoring method GASTS can quickly find accurate estimation of the minimum numbers of rearrangements on thousands of genomes with thousands of genes in each genome. To infer the correct phylogeny for a given quartet problem with four taxa: A, B, C and D, we just need to compute the tree scores S_{GASTS} (we call them *phylogenetic score functions* in this paper) for the three topologies $AB|CD$, $AC|BD$ and $AD|BC$ and return the topology $\hat{T}_{\text{GASTS}}$ with the smallest GASTS score.

2.2 Distance Based Methods and Four-Point Metric

Assume the correct topology of the quartet problem is depicted by Fig. 2. If the distance between any two species is equal to the length of their path on the tree, e.g. $d_{AD} = v_1 + v_5 + v_4$, then the following *four point metric* holds:

$$d_{AB} + d_{CD} < d_{AC} + d_{BD} = d_{AD} + d_{BC}. \tag{1}$$

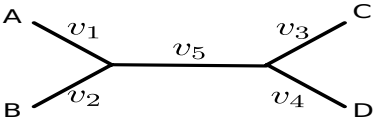


Fig. 2. The underlying tree for the four species A, B, C and D

This relation can be easily verified and it can be used to test whether the distance is additive. A weak version of this relation not requiring the equality can be used to make phylogenetic inference. The distance sum can be thought of as the sum of the distances between sibling species on the tree. Denote $d_{AB|CD} = d_{AB} + d_{CD}$, $d_{AC|BD} = d_{AC} + d_{BD}$ and $d_{AD|BC} = d_{AD} + d_{BC}$ as the distance sums for the corresponding three topologies. The inferred topology is the one with the smallest distance sum, as the others use the internal edge twice and hence have larger values.

Furthermore, the length of the internal edge can be estimated by:

$$\hat{v}_5 = \frac{1}{2}\Big[\max\{d_{AB}+d_{CD}, d_{AC}+d_{BD}, d_{AD}+d_{CB}\} - \min\{d_{AB}+d_{CD}, d_{AC}+d_{BD}, d_{AD}+d_{CB}\}\Big]. \quad (2)$$

The estimated internal length also indicates the significance of the inferred phylogeny: a small internal edge is likely to arise from noisy data and a large internal edge is likely to show the true phylogeny. In reconstructing large trees, this information is used to resolve conflicting conclusions on internal edges or subtrees.

We apply both the breakpoint distance d_{BP} and the DCJ distance d_{DCJ} for genome rearrangement data. For a topology $T = AB|CD$, the two phylogenetic score functions $S_{\mathrm{BP},T}$ and $S_{\mathrm{DCJ},T}$ denote $d_{\mathrm{BP},AB} + d_{\mathrm{BP},CD}$ and $d_{\mathrm{DCJ},AB} + d_{\mathrm{DCJ},CD}$ respectively. Phylogenetic inferences $\hat{T}_{\mathrm{BP}}$ and $\hat{T}_{\mathrm{DCJ}}$ give the topologies which minimize $S_{\mathrm{BP},T}$ and $S_{\mathrm{DCJ},T}$. In many situations, it is more convenient to compute the number of shared adjacencies a and the number of color-alternating cycles c. We use S_a and S_c to denote the corresponding phylogenetic score functions; the inferred phylogeny is the one maximizing S_a or S_c.

Under the breakpoint distance, an adjacency shared by three or all four genomes contributes once or twice to the S_a scores of all three topologies. So it does not provide phylogenetic information, and it is sufficient to only count the number of adjacencies shared by exactly two of the four genomes. Then, for each topology, say $T = AB|CD$, $S_{a,T}$ is redefined as the number of adjacencies shared only by A and B or C and D. Alekseyev and Pevzner [1] suggested using S_a to infer phylogeny.

2.3 Phylogeny Invariants

Phylogeny invariant methods directly examine patterns and their occurrence frequencies presented in the data. These methods are easier to understand on sequence data. Given 4 DNA sequences A, B, C and D, the pattern presented at site i can be, the first two nucleotides of A_i, B_i, C_i and D_i take the same value, say G, and the last two nucleotides take another value, say T. Given a topology (on which, edge lengths are unknown and irrelevant), a set of algebraic relations (*invariants*) about the expected frequencies can be derived. To infer phylogeny, we find the topology that best fits these invariants in a statistical sense.

On 4 DNA or protein sequences, we may observe $4^4 = 256$ or $20^4 = 1.6 \times 10^5$ different configurations. By the following encoding method, these configurations

can be reduced to 15 patterns[2]. For a site i, we assign x to A_i, y to the first character different from x among B_i to D_i, z to the first character different from x and y, and so on. The 15 patterns on four sequences are:

$$xxxx$$
$$xyxx, xxyx, xxxy, xyyy$$
$$xxyy, xyxy, xyyx$$
$$xxyz, xyxz, xyzx, xyyz, xyzy, xyzz$$
$$xyzw.$$

Sankoff and Blanchette [12] explored how to apply invariants methods on gene order data. The characters they used are the adjacencies in the gene orders: the successors for any gene extremity g (an endpoint of a gene, represented by a vertex in the breakpoint graph) on different gene orders. Here g plays a similar role as a site i in sequence data, and its successors in the gene orders have a similar role as nucleotides or amino acids. Sankoff and Blanchette first derived stochastic probabilities for a gene h to take position i after k random inversions. From these probabilities, they found 11 phylogenetic invariants in the case of five species.

Phylogeny invariants methods and our new methods are similar, in the way that both methods explore directly the patterns presented in the data. But phylogeny invariants methods are probabilistic methods, which concern about expected probabilities and statistics tests. Our new methods are under the parsimony framework.

3 Rearrangement Phylogenetic Patterns

In this section, we explore rearrangement phylogenetic patterns on four genomes. Under the parsimony framework, given a distance measure, a rearrangement phylogenetic pattern is a pattern that always gives smaller scores on one fixed topology but not on the others. In other words, a rearrangement phylogenetic pattern differentiates different topologies. We consider rearrangement phylogenetic patterns with regard to both the breakpoint distance and the DCJ distance, with the latter closely related to the inversion distance and its generalization, the HP distance.

Genome rearrangement problems have far more combinatorial structures than the corresponding problems on sequences, thus they are much harder to study, especially for the ones concerning three or more genomes [4,10,5,14]. This paper initiates the discussion on rearrangement phylogenetic patterns, with the emphasis on small sized patterns. As a learnt experience from adequate subgraphs for the median problem, combinatorial structures of small sizes alone may provide sufficient information on solving the problem. Furthermore, rearrangement phylogenetic patterns of small sizes will be amenable for further probabilistic analysis, which will provide statistical significance for phylogenetic inference.

[2] This number is given by the Bell number, $B(n)$, with the leading terms 1, 2, 5, 15, 52, 203 and the recurrence equation $B(n+1) = \sum_{k=0}^{n} \binom{n}{k} B(n)$.

In Subsection 3.1 we present some negative results, which exclude some combinatorial structures from being rearrangement phylogenetic patterns. Surprisingly, adjacencies shared by two or more genomes, which constitute the basis of S_{BP} or S_a, are not rearrangement phylogenetic patterns. This conclusion is also verified by the observations in Section 4. In Subsection 3.2, we introduce a class of rearrangement phylogenetic patterns and prove that they differentiate different topologies with regard to both the breakpoint distance and the DCJ distance. Then we introduce two related phylogenetic score functions and their phylogenetic inference methods.

3.1 Patterns Which Are Not Rearrangement Phylogenetic Patterns

Theorem 1. *Out of the 15 patterns discussed in Subsection 2.3, the following 6 patterns are not rearrangement phylogenetic patterns with regard to either the breakpoint distance or the DCJ distance: xxxx, xyxx, xxyx, xxxy, xyyy, and xyzw.*

Proof. Under the breakpoint distance (or the DCJ distance), the first, the next four and the last patterns contribute 5, 4 and 2 pairwise shared common adjacencies (or same numbers of color-alternating cycles), respectively. Hence these patterns do not differentiate the three topologies. □

This theorem tells us that an adjacency shared by 1, 3 or 4 genomes does not contain any phylogenetic information. The next theorem states that an adjacency shared only by 2 genomes, described by the six patterns $xxyz$, $xyxz$, $xyzx$, $xyyz$, $xyzy$, and $xyzz$, does not contain phylogenetic information either, with regard to the breakpoint distance or the DCJ distance.

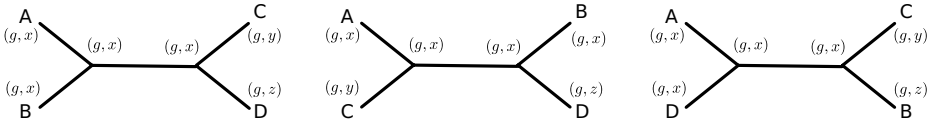


Fig. 3. A counter example showing an adjacency shared by two genomes is not a rearrangement phylogenetic pattern. The optimal configurations for the three topologies are shown next to the nodes.

Theorem 2. *A pairwise shared adjacency is not a rearrangement phylogenetic pattern.*

Proof. We prove this by showing two counter examples: a simple one and a complicated one.

Counter Example I. In Fig. 3 genome A and B share a common adjacency (g, x), genome C has the adjacency (g, y), and genome D has (g, z), where g, x, y and z are gene extremities. In the optimal solutions, the two internal nodes both contain (g, x). Hence all three topologies contribute 3 pairwise shared adjacencies or 3 color-alternating cycles.

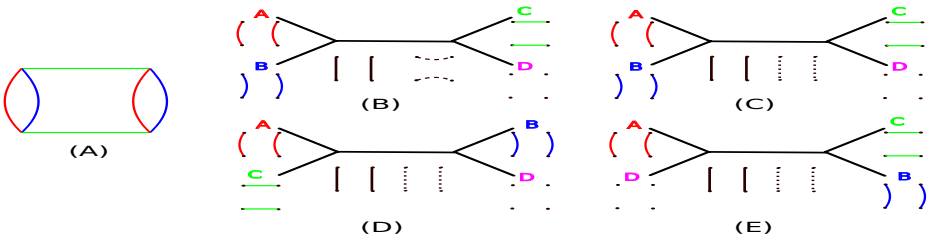


Fig. 4. Another counter example showing adjacencies shared by two genomes are not rearrangement phylogenetic patterns. Genome A and B share two adjacencies and Genome C form two different adjacencies on the same four extremities (A). (B), (C), (D) and (E) show optimal configurations of the internal nodes for the three topologies $AB|CD$, $AC|BD$ and $AD|BC$. Note that $AB|CD$ has two optimal configurations. The configurations are shown next to the nodes. Solid black edges represent adjacencies for the first internal node and dashed black edges represent adjacencies for the second internal node.

Counter Example II. In Fig. 4 (b) genome A and B share two common adjacencies and genome C has two different adjacencies on these four gene extremities. Subfigures (B)–(E) show the configurations taken by the two internal nodes, where the optimality of these configurations can be easily proved (either by the adequate subgraph theory or similar techniques applicable for the breakpoint distance). Note that for the first topology $AB|CD$, there are two locally optimal configurations, and at least one of them is part of a global optimal configuration. All three topologies contribute 6 pairwise shared adjacencies and 7 color-alternating cycles.

Therefore, adjacencies shared by only two genomes are not rearrangement phylogenetic patterns. □

3.2 Rearrangement Phylogenetic Patterns and Contrasting Shared Adjacencies

A *contrasting shared adjacency*, is when a gene extremity g forms one adjacency on two genomes and another adjacency on the other two genomes. Related to the patterns discussed in Subsection 2.3, a contrasting shared adjacency corresponds to $xxyy, xyxy, xyyx$.

In the two counter examples used to prove Theorem 2, the two internal nodes take the same configuration in the optimal configurations. When this happens on all three topologies, the two scores S_a and S_c will always be the same. For a pattern to be phylogenetic, the internal nodes have to be different in the optimal configurations. The next theorem shows that a contrasting shared adjacency has such a property.

Theorem 3. *If a contrasting shared adjacency forms an adjacency* (g, x) *on genome A and B and* (g, y) *on genome C and D, then we have the following conclusions:*

1. *the configurations of the two internal node on the topology $AB|CD$ are different;*
2. *the contrasting shared adjacency differentiates the three topologies and $AB|CD$ is the preferred one, with regard to both the breakpoint distance and the DCJ distance;*
3. *the contrasting shared adjacency is a rearrangement phylogenetic pattern.*

Proof. Fig. 5 shows the optimal configurations of the internal nodes for all three topologies. The optimality can be easily shown by the adequate subgraph theory (or similar techniques for the breakpoint distance) and so is the fact that the two nodes on $AB|CD$ take different configurations. Furthermore, on $AB|CD$ there are 4 pairwise shared adjacencies and color-alternating cycles; while on the other two topologies, there are only 3 pairwise shared adjacencies and color-alternating cycles. Hence a contrasting shared adjacency is a rearrangement phylogenetic pattern and $AB|CD$ is the preferred topology with regard to both the breakpoint distance and the DCJ distance. □

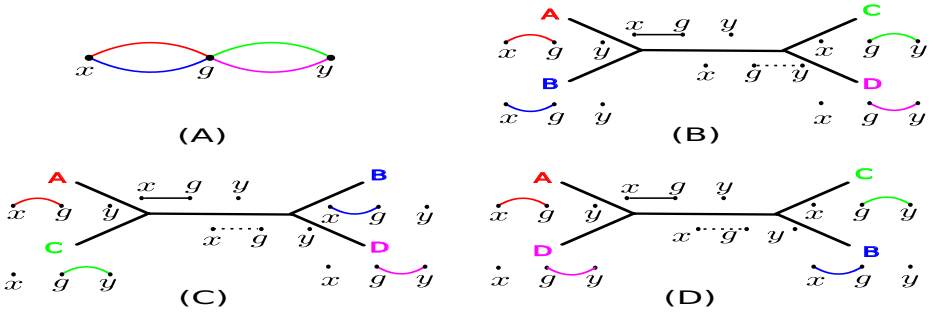


Fig. 5. Contrasting shared adjacency, where Genome A and B share one adjacency (g, x) and Genome C and D share another adjacency (g, y). (B), (C) and (D) show optimal configurations of the internal nodes for the three topologies $AB|CD$, $AC|BD$ and $AD|BC$. The configurations are shown next to the nodes. Solid black edges represent adjacencies for the first internal node and dashed black edges represent adjacencies for the second internal node.

3.3 Multi-paths, Multi-cycles and Two New Phylogenetic Score Functions

A multi-path (a multi-cycle) is a path (a cycle) only consisting of multi-edges.The size of a multi-path (a cycle) is the number of multi-edges it contains, which is denoted by l. We say a multi-path (a multi-cycle) is consistent with a topology $T = AB|CD$, if its adjacencies are pairwise shared by either genomes A, B or genomes C, D. In the breakpoint distance based method, a multi-path (a multi-cycle) of size l contributes l units toward $S_{\mathrm{BP},T}$.

We have shown, in proving Theorem 3, two contrasting adjacencies (g, x) and (g, y) together force the two internal nodes to take different configurations. In

counting how many units of phylogenetic information a multi-path (a multi-cycle) contains, we argue that it should be equal to the maximum number of non-overlapping[3] contrasting shared adjacencies contained in that path (cycle), which is $\lfloor \frac{l}{2} \rfloor$ units of phylogenetic information[4].

We define the phylogenetic score function $S_{\mathrm{CA},T}$ as the maximum number of non-overlapping contrasting shared adjacencies consistent with the topology T. This can be explicitly expressed as:

$$S_{\mathrm{CA},T} = \sum_{p \text{ consistent with } T} \left\lfloor \frac{|p|}{2} \right\rfloor, \tag{3}$$

where p is a multi-path or multi-cycle and $|p|$ is its size. And the corresponding phylogenetic inference $\hat{T}_{\mathrm{CA}}$ is the topology T with the maximum $S_{\mathrm{CA},T}$.

Under the DCJ distance, we have the following result for multi-cycles. This result means, a multi-cycle of size $l = 2k$ (always an even number) only contributes $k-1$ units of phylogenetic information, instead of k.

Theorem 4. *A multi-cycle of size $l = 2k$ contributes $4k+1$ color-alternating cycles on its consistent topology and contributes $3k+2$ color-alternating cycles on other two topologies.*

As the difference in the number of cycles is $k-1$, it will be better to assign only $k-1$ units of phylogenetic information to the multi-cycle.

Remark 1. For a multi-cycle with $2k$ multi-edges, if we treat these multi-edges as simple edges, the DCJ distance defined on this cycle is also $k-1$. Although the two $k-1$s coincide, they come from two different problems. If the distance measure is additive, the distance on the wrong topology will count the internal edge twice, and the extra counting explains the difference in the total distances.

With this modification, we have a new phylogenetic score function $S_{\mathrm{MCA},T}$:

$$S_{\mathrm{MCA},T} = \sum_{\text{multi-path } p \text{ consistent with } T} \left\lfloor \frac{|p|}{2} \right\rfloor + \sum_{\text{multi-cycle } p \text{ consistent with } T} \frac{|p|}{2} - 1. \tag{4}$$

And the corresponding phylogenetic inference $\hat{T}_{\mathrm{MCA}}$ is just the topology with the largest S_{MCA}.

4 Testing the Accuracies of Phylogenetic Inference Methods on Simulated Data

We generated various groups of simulation data to compare the accuracies of various phylogenetic inference methods. Simulated genomes were generated according to the tree shown in Fig. 2 but with only two parameters: $e_1 = v_5$ and

[3] Non-edge-overlapping, to be more accurate.

[4] The notation $\lfloor x \rfloor$ denotes the largest integer which is no larger than x.

$e_2 = v_1 = v_2 = v_3 = v_4$. The edge lengths e_1 and e_2 denote the number of random inversions applied. In the first group of data, e_1 and e_2 varied among 5, 10, 20, 30 and 40. In the second group of data, to study the effect of short internal edges, we let e_1 take very short lengths: 1, 2, 3, 4 and 5, and let e_2 take values among 5, 10, 20, 30, 40, 50 and 60. We generated 1,000 datasets for each parameter combination; thus we generated a total of 55,000 simulation datasets. We used genomes of single circular chromosome containing 200 genes, so that we finished the whole test very quickly: for each dataset, all phylogenetic inferences finished within a couple of seconds.

4.1 Comparing Accuracies of Various Inference Methods

We compared the accuracies of five phylogenetic inference methods ($\hat{T}_{\text{MCA}}$, $\hat{T}_{\text{CA}}$, $\hat{T}_{\text{GASTS}}$, $\hat{T}_{\text{BP}}$ and $\hat{T}_{\text{DCJ}}$) on the first group of data. We used a strict criteria to calculate accuracies: a method makes a correct inference only when the true topology is given as the unique result. For example, if the inference of a method contains two topologies, even with the true one included, we still treat this inference as wrong. Under this strict criteria, accuracies would appear low, but the comparison results are not affected.

Table 1 shows the comparison results for $e_1 = 5$, 10 and 20. Results for $e_1 = 30$ and 40 are not shown, as the accuracies were all 100%. Overall, these methods were very accurate, except when the internal edge was small and the outer edges were large. With no surprise, the full parsimony method $\hat{T}_{\text{GASTS}}$ had the best accuracy, leading the second best method by up to 7 percentage points. The two new phylogenetic inference methods $\hat{T}_{\text{MCA}}$ and $\hat{T}_{\text{CA}}$ had the second best accuracies, better than the next best method by up to another 7 percentage points. Among the two, $\hat{T}_{\text{MCA}}$ was slightly better than $\hat{T}_{\text{CA}}$. This shows that special modification regarding the DCJ distance measure has a marginal advantage. The next best method was $\hat{T}_{\text{BP}}$, leading $\hat{T}_{\text{DCJ}}$ by nearly 5 percentage points. However, there was a trend that their difference decreases as the internal edge length increases.

4.2 Tests on Challenging Cases with Short Internal Edges and Long Outer Edges

Fig. 6 shows the comparison results on cases with e_1 =1,2,3 and 5, e_2 =5, 10, 20, 30, 40, 50 and 60. Results for $e_1 = 4$ are not shown, as they are similar to the ones for $e_1 = 5$. As the cases with short internal edges and long outer edges are very difficult, it is not surprising to see accuracies as low as 40%. On the contrary, it is a little surprising to see the two new methods $\hat{T}_{\text{MCA}}$ and $\hat{T}_{\text{CA}}$ had nearly 50% accuracies on the extremely difficult cases where the outer edges were 60 times the length of the internal edge. As the chance to make a correct inference by chance is no larger than 33% (as ties are not regarded as correct), these results were fairly good.

Another obvious observation is, the accuracies increase quickly with the length of the internal edge length. When $e_1 = 1$, the accuracies started from 90%

Table 1. Comparison of 5 phylogenetic inference methods on the first group of datasets. The internal edge length e_1 and the outer edge length e_2 took values among 5, 10, 20, 30 and 40 random inversion. Accuracies are measured strictly where any tie is treated as incorrect.

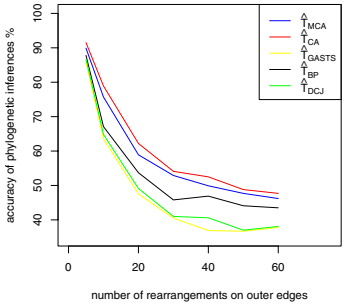
e_1	e_2	$\hat{T}_{\mathrm{MCA}}$	$\hat{T}_{\mathrm{CA}}$	$\hat{T}_{\mathrm{GASTS}}$	$\hat{T}_{\mathrm{BP}}$	$\hat{T}_{\mathrm{DCJ}}$
5	5	100	100	100	99.9	99.7
5	10	100	99.9	100	99.6	98.8
5	20	98.1	97.0	98.5	94.9	90.6
5	30	88.0	88.2	93.0	81.5	78.6
5	40	80.2	79.9	87.7	72.4	68.8
10	5	100	100	100	100	100
10	10	100	100	100	100	100
10	20	100	99.9	100	99.9	99.2
10	30	99.0	98.5	100	97.7	95.8
10	40	94.8	94.1	99.1	93.2	88.7
20	20	100	100	100	100	100
20	30	100	100	100	100	100
20	40	99.6	99.6	100	99.3	99.6

($e_2 = 5$) and quickly dropped to 40%–50% ($e_2 = 60$); when $e_1 = 5$, they increase to 100% ($e_2 = 5$) or 55%–75% ($e_2 = 60$).

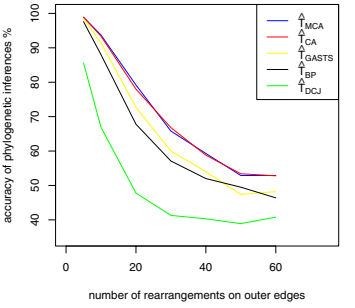
The two new methods $\hat{T}_{\mathrm{MCA}}$ and $\hat{T}_{\mathrm{CA}}$ had the best overall performance. They remained the best, until e_1 increased to 4 and 5, where they were only second to the full parsimony method $\hat{T}_{\mathrm{GASTS}}$. $\hat{T}_{\mathrm{GASTS}}$ performed poorly when $e_1 = 1$, fairly when $e_1 = 2$, and well when $e_1 \geq 3$. $\hat{T}_{\mathrm{BP}}$ had decent performance, better than $\hat{T}_{\mathrm{DCJ}}$. We can explain the fact that the three adjacency-related methods had the best performance for $e_1 = 1$ by the following argument. The single inversion on the internal edge leaves two breakpoints. The trace of these breakpoints may be erased by random inversions on the four outer edges. In order to correctly reconstruct this inversion, DCJ based methods need both breakpoints to remain. Adjacency-related methods can work even if only one breakpoint remains.

4.3 How Do the Phylogenetic Score Functions Correlate to the Number of Rearrangements

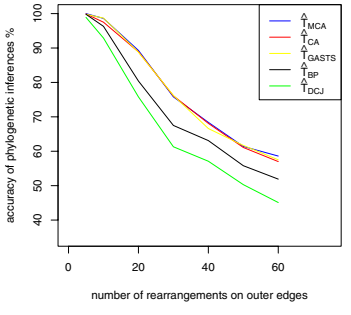
On the first group of datasets, we further investigated how the phylogenetic score functions correlate to the number of rearrangements. The number of rearrangements on the true topology can be easily calculated during simulation, however this is not feasible on the other two topologies. As GASTS can very accurately estimates the number of rearrangements on a given topology, we use the GASTS scores to approximate the real numbers of rearrangements. We then calculated how the other phylogenetic score functions deviated from the GASTS scores. We denote the three topologies as T_i with $i = 1$, 2 and 3, where T_1 is the true topology. We calculated the deviations for S_{MCA}, S_{CA}, S_{BP}, $\frac{S_{\mathrm{BP}}}{2}$, S_{DCJ} and $\frac{S_{\mathrm{DCJ}}}{2}$. The two fractional scores are considered, because of the factor of 2 in Equation 2.



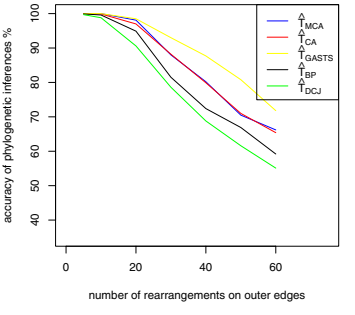
(a) 1 rearrangement on the internal edge



(b) 2 rearrangements on the internal edge



(c) 3 rearrangements on the internal edge



(d) 5 rearrangements on the internal edge

Fig. 6. Comparison of 5 phylogenetic inference methods on datasets with very small internal edges. The length e_1 of these edges varied among 1, 2, 3, 4 and 5 random inversions. Accuracies are measured strictly where any tie is treated as incorrect.

Let P represent any of the above 6 phylogenetic score functions. The deviation diff(P) calculates the discrepancy between the difference in S_{P} and the difference in S_{GASTS} over different topologies on the same datasets. It is given by the following formula:

$$\begin{aligned}\mathrm{diff}(P) = \frac{1}{2000}\sum_{i=1}^{1000}\{&|\,(S_{P,T_1} - S_{P,T_2}) - (S_{\mathrm{GASS},T_1} - S_{\mathrm{GASS},T_2})\,| \\ &+|\,(S_{P,T_1} - S_{P,T_3}) - (S_{\mathrm{GASS},T_1} - S_{\mathrm{GASS},T_3})\,|\}. \end{aligned} \quad (5)$$

Table 2 shows the results. The two score functions S_{BP} and S_{DCJ} deviated significantly from the GASTS scores; but their fractional versions had much better performance. $\frac{S_{\mathrm{DCJ}}}{2}$ had very small deviations, only second to S_{MCA}. S_{MCA} had an excellent performance, with very small deviations from the GASTS scores,

Table 2. Deviations between six phylogenetic score functions and the GASTS score. The deviation is defined by Equation 5. As the GASTS score well approximates the number of rearrangements for any given tree, these deviations show how well these six phylogenetic score functions reflect the variation of the number of rearrangements over different topologies. The deviations for S_{MCA} are in bold font, if they are smaller than 2. e_1 and e_2 are the lengths of internal and outer edges, respectively.

e_1	e_2	diff(S_{MCA})	diff(S_{CA})	diff(S_{BP})	diff($\frac{S_{\mathrm{BP}}}{2}$)	diff(S_{DCJ})	diff($\frac{S_{\mathrm{DCJ}}}{2}$)
5	5	**0.1435**	3.1985	12.465	3.8337	4.8845	0.4315
5	10	**0.3970**	2.2015	10.664	3.089	4.8190	0.8970
5	20	**1.0155**	1.5345	8.4615	2.6680	5.454	1.7460
5	30	**1.5490**	1.6580	7.2310	2.6550	6.1460	2.4405
5	40	2.1055	2.1435	6.6230	2.8687	7.0365	2.9890
10	5	**0.2410**	6.0200	24.127	7.2652	9.6765	0.5550
10	10	**0.6345**	3.8970	20.876	5.7495	9.4910	1.0505
10	20	**1.6520**	1.8600	14.825	3.4027	8.9575	1.8160
20	5	**0.4670**	10.687	44.850	12.969	19.125	0.7430
20	10	**1.2020**	6.2245	38.051	9.7300	18.644	1.2835
20	20	3.1775	2.4450	26.761	4.8037	17.483	2.0010
30	5	**0.6205**	13.855	62.395	17.191	28.235	0.8870
30	10	**1.7490**	7.9415	53.082	12.777	27.477	1.3840
40	5	**0.9020**	16.229	77.793	20.449	37.253	1.0150
40	10	2.4585	8.7070	65.984	14.852	36.190	1.5275

especially when the total number of events were small. This shows that S_{MCA} can well explain the difference in the number of rearrangements over different topologies. And the reason that its deviations increased is: the gap between S_{MCA} and S_{GASTS} was mainly caused by large rearrangement phylogenetic patterns, which occurred more frequently when the number of events got larger; these large rearrangement phylogenetic patterns are not considered in our paper. The closely related score function S_{CA} had much worse performance and this justifies our special consideration on multi-cycles.

5 Conclusion and Future Work

In this paper, we explore rearrangement phylogenetic patterns for the genome rearrangement quartet problem. A rearrangement phylogenetic pattern is a combinatorial structure, which contains phylogenetic information, and by examining their occurrences, phylogenetic inferences can be made. As the first solid study of this subject, we prove what are or are not genome rearrangement phylogenetic patterns, with regard to the breakpoint distance and the DCJ distance and under the parsimony framework. We define two phylogenetic score functions as the numbers of the observed rearrangement phylogenetic patterns, and based on them we design two phylogenetic inference methods. Tested on simulated data, these new methods demonstrated good accuracies, only second to the full parsimony method, and remarkable robustness when the internal edge of the tree is

extremely short. All these observations imply that, rearrangement phylogenetic patterns indeed carry a significant amount of phylogenetic information and this is a promising alternative approach to study phylogenetic problems.

There are many open problems to explore. On the quartet problem, discovering of more rearrangement phylogenetic patterns of larger sizes or the patterns specialized for linear chromosomes will certainly increase the power for phylogenetic inferences; analyzing the probabilistic properties of these patterns will allow us to develop statistic tests for the significance of the inferences. The topic of discovering rearrangement phylogenetic patterns can go beyond four genomes. The question of how to use these rearrangement phylogenetic patterns and their probabilistic properties to design efficient and accurate methods for phylogenetic problems with large numbers of taxa, will be very interesting but challenging.

Acknowledgments

The author would like to thank the anonymous referees for their helpful suggestions on writing this paper.

References

1. Alekseyev, M.A., Pevzner, P.: Breakpoint Graphs and Ancestral Genome Reconstructions. Genome Res. 19, 943–957 (2009)
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
3. Bourque, G., Pevzner, P.: Genome-scale evolution: Reconstructing gene orders in the ancestral species. Genome Res. 12, 26–36 (2002)
4. Bryant, D.: The complexity of the breakpoint median problem. TR CRM-2579. Centre de recherches mathématiques, Université de Montréal (1998)
5. Caprara, A.: The reversal median problem. INFORMS J. Comput. 15, 93–113 (2003)
6. Hannenhalli, S., Pevzner, P.: Transforming men into mice (polynomial algorithm for genomic distance problem. In: Proc. 43rd IEEE Symp. on Foudations of Computer Science FOCS 1995, pp. 581–592. IEEE Computer Soc., Los Alamitos (1995)
7. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. JACM 46, 1–27 (1999)
8. Moret, B., Siepel, A., Tang, J., Liu, T.: Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, p. 521. Springer, Heidelberg (2002)
9. Murphy, W.J., Larkin, D.M., van der Wind, A.E., Bourque, G., Tesler, G., Auvil, L., Beever, J.E., Chowdhary, B.P., Galibert, F., Gatzke, L., Hitte, C., Meyers, S.N., Milan, D., Ostrander, E.A., Pape, G., Parker, H.G., Raudsepp, T., Rogatcheva, M.B., Schook, L.B., Skow, L.C., Welge, M., Womack, J.E., O'Brien, S.J., Pevzner, P.A., Lewin, H.A.: Dynamics of Mammalian Chromosome Evolution Inferred from Multispecies Comparative Maps. Science 309(5734), 613–617 (2005)
10. Pe'er, I., Shamir, R.: The median problems for breakpoints are np-complete. In: Burkhardt, H., Neumann, B. (eds.) ECCV 1998. LNCS, vol. 1407. Springer, Heidelberg (1998)

11. Sankoff, D., Blanchette, M.: Multiple genome rearrangement and breakpoint phylogeny. J. Comput. Biol. 5, 555–570 (1998)
12. Sankoff, D., Blanchette, M.: Phylogenetic invariants for genome rearrangements. Journal of computational biology: a journal of computational molecular cell biology 6(3-4), 431–445 (1999)
13. Sankoff, D., Zheng, C., Wall, P.K., DePamphilis, C., Leebens-Mack, J., Albert, V.A.: Towards improved reconstruction of ancestral gene order in angiosperm phylogeny. Journal of Computational Biology: A Journal of Computational Molecular Cell Biology 16(10), 1353–1367 (2009)
14. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS (LNBI), vol. 5251, pp. 1–13. Springer, Heidelberg (2008)
15. Xu, A.W.: DCJ median problems on linear multichromosomal genomes: Graph representation and fast exact solutions. In: Ciccarelli, F.D., Miklós, I. (eds.) RECOMB-CG 2009. LNCS, vol. 5817, pp. 70–83. Springer, Heidelberg (2009)
16. Xu, A.W., Moret, B.M.: Genome rearrangement analysis on thousands of taxa with thousands of synteny blocks (submitted, 2010)
17. Xu, A.W., Sankoff, D.: Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS (LNBI), vol. 5251, pp. 25–37. Springer, Heidelberg (2008)
18. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)

Fast and Accurate Phylogenetic Reconstruction from High-Resolution Whole-Genome Data and a Novel Robustness Estimator

Yu Lin*, Vaibhav Rajan*, and Bernard M.E. Moret

Laboratory for Computational Biology and Bioinformatics, EPFL,
EPFL-IC-LCBB INJ230, Station 14, CH-1015 Lausanne, Switzerland
{yu.lin,vaibhav.rajan,bernard.moret}@epfl.ch

Abstract. The rapid accumulation of whole-genome data has renewed interest in the study of genomic rearrangements. Comparative genomics, evolutionary biology, and cancer research all require models and algorithms to elucidate the mechanisms, history, and consequences of these rearrangements. However, even simple models lead to NP-hard problems, particularly in the area of phylogenetic analysis. Current approaches are limited to small collections of genomes and low-resolution data (typically a few hundred syntenic blocks). Moreover, whereas phylogenetic analyses from sequence data are deemed incomplete unless bootstrapping scores (a measure of confidence) are given for each tree edge, no equivalent to bootstrapping exists for rearrangement-based phylogenetic analysis.

We describe a fast and accurate algorithm for rearrangement analysis that scales up, in both time and accuracy, to modern high-resolution genomic data. We also describe a novel approach to estimate the robustness of results—an equivalent to the bootstrapping analysis used in sequence-based phylogenetic reconstruction. We present the results of extensive testing on both simulated and real data showing that our algorithm returns very accurate results, while scaling linearly with the size of the genomes and cubically with their number. We also present extensive experimental results showing that our approach to robustness testing provides excellent estimates of confidence, which, moreover, can be tuned to trade off thresholds between false positives and false negatives. Together, these two novel approaches enable us to attack heretofore intractable problems, such as phylogenetic inference for high-resolution vertebrate genomes, as we demonstrate on a set of six vertebrate genomes with 8,380 syntenic blocks.
Availability: a copy of the software is available on demand.

1 Introduction

Genomic rearrangements have been studied by biologists since their discovery in the late 1910s (see [25]). In 1987 Day and Sankoff [7] proposed two major problems about rearrangements: the *edit distance*—given two genomes and a model of rearrangements, find the shortest sequence of rearrangements that transforms one input genome into the other; and the *median*—given three genomes, construct a fourth genome that minimizes

* Who contributed equally to this work.

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 137–148, 2010.

the sum of its pairwise distances to the other three. The edit distance, which underlies most distance-based phylogenetic reconstruction methods, is computable in linear time for most models, while the median, which underlies maximum parsimony and most Bayesian and maximum likelihood reconstruction methods, is NP-hard for most models [10]. Phylogenetic reconstruction from rearrangement data attracted attention, as rearrangements are "rare genomic events" [21] and thus might help resolve difficult questions about ancient branching patterns in evolution, but the computational complexity of parsimonious approaches precluded widespread application of the approach. However, as whole-genome data multiplies everywhere, phylogenetic reconstruction based on rearrangements is becoming a necessity for work in comparative genomics.

In earlier work [13], we described a method for estimating very precisely the true evolutionary distance between two genomes under rearrangements in the absence of duplications and losses, when every genome shares the same collection of genes. While the limitation to identical gene content would be severe, whole-genome data is typically given in terms of syntenic blocks rather than genes; and syntenic blocks can be selected to obey these limitations: they are shared among the genomes and are not duplicated. We show in this paper how our true distance estimator can be used to reconstruct highly accurate phylogenetic trees from high-resolution genomic data using simple distance methods.

A major drawback of phylogenetic reconstruction based on rearrangements has been the lack of any way to assess the robustness of the reconstructed edges. In phylogenetic analysis from sequence data, such an assessment is *de rigueur* and is carried out through a *bootstrapping* process, formally proposed in 1985 by Felsenstein [8], in which replicates of the multiple sequence alignment are produced by random sampling with replacement of the columns, a tree is built from each replicate, and each edge of the original tree assigned a score that is simply the fraction of replicate trees that contain this same edge. While the interpretation of bootstrapping scores remains uncertain (see, e.g., [24]), the practice of providing such scores is nearly universal. However, the same bootstrapping process cannot be applied to rearrangement data, since a rearrangement is a single character. (Jackknifing, in which some taxa are dropped, remains viable, but the relatively small number of taxa in most analyses, together with the fact that dropping certain taxa can lead to poor reconstruction due to insufficient taxon sampling [33], severely limit the power of the approach.)

In this paper, we propose an entirely new approach to the assessment of the robustness of a reconstruction, in which we take advantage of the additive nature of true evolutionary distances. This approach is fast, uses exactly the same reconstruction algorithm as produced the original tree, allows the user to trade off sensitivity for specificity (not possible in the conventional approach to phylogenetic bootstrapping), and produces support values that correlate well with error measures.

2 Background

Rearrangement data was used in phylogenetic analysis 80 years ago by Sturtevant and Dobzhansky [26]. Sankoff and Blanchette [4] introduced the first algorithmic approach to the reconstruction of a phylogenetic tree from rearrangement data, BPAnalysis. The

algorithm seeks the tree and internal genomes which together minimize the total number of *breakpoints*—adjacencies present in one genome, but absent in the other. Moret *et al.* [17] reimplemented this approach in their GRAPPA tool and extended it to *inversion distances*—inversions are the best documented of the hypothesized mechanisms of genomic rearrangements. This work focused on unichromosomal genomes; to handle multichromosomal genomes, Bourque and Pevzner [5] proposed MGR, based on GRAPPA's distance computations. Whereas BPAnalysis and GRAPPA search all trees and report the one with the best score (an approach that limits GRAPPA to trees of 15 taxa unless combined with the DCM approach [28], in which case it scales up to 1,000 taxa), MGR uses a heuristic sequential addition method to grow the tree one species at a time. The heuristic approach trades accuracy for scalability, yet MGR does not scale well—in particular, it cannot be used to infer a phylogeny from modern high-resolution data, as even just a few such genomes may require days or weeks of computation. Yet to date MGR (and its more recent derivative MGRA [1]) had remained the only tool available for the analysis of multichromosomal genomic rearrangements. All such parsimony-based approaches must produce good approximations to the NP-hard problem of computing the rearrangement median of three genomes, which limits their scalability.

Distance-based methods, in contrast, run in time polynomial in the number and size of genomes. Moreover, methods like Neighbor-Joining (NJ) [22] provably return the true tree when given true evolutionary distances. Their speed has long been a major attraction, but the distances that can be computed with sequence data are often far from the true evolutionary distances, particularly on datasets with markedly divergent genomes. Pairwise distances are often computed as edit distances, that is, as minimum-cost distances under the assumed model of evolution. However, even with detailed models, such an edit distance typically underestimates the true distance and that underestimation worsens as the true distance grows. The result is poor trees (see, e.g., [16] for examples from rearrangement data). The true evolutionary distance—the actual number of evolutionary events between the two genomes—is at once an abstraction and impossible to measure; however, it can be estimated using statistical techniques, something that has long been done with sequence data (see, e.g., [27]).

Distance estimators have been used for rearrangement data. For unichromosomal genomes under inversions, transpositions, and inverted transpositions, Wang and Warnow [30] showed how to estimate a true evolutionary distance from the breakpoint distance (the number of disrupted gene adjacencies), later deriving exact formulas [29]. For unichromosomal genomes evolving under inversions only, an experimental approach was used by our group to derive the EDE estimate from the inversion edit distance [16], yielding greatly increased accuracy in tree estimation under both distance and parsimony methods.

For multichromosomal genomes, all of the rearrangement operations can be modeled by a single operation called "Double-Cut-and-Join" (DCJ) [3, 32]. One DCJ operation makes two cuts, which can be in the same chromosome or in two different chromosomes, producing four cut ends, and then rejoins the four cut ends (in three possible ways) to mimic different kinds of rearrangements. We have described a statistical method, using exact formulas, to estimate the true evolutionary distance between two

genomes under the DCJ model [13], an estimator that we recently refined to include gene duplication and loss events [14].

Nonparametric bootstrapping has become the standard method used in phylogenetic analysis to assess the robustness of the reconstruction. In phylogenetic reconstruction, bootstrapping provides *support values* for branches (or clades) in the estimated tree. The classical bootstrapping method for sequence data by Felsenstein [8] samples columns with replacement from a multiple sequence alignment to create a new alignment called the bootstrap replicate. Each replicate thus contains the same number of species and the same number of columns per species, but some columns in the original alignment may be duplicated and others omitted in the sampling process. The result is a shift in column bias, with some replicates possibly losing many significant characters and others losing possibly confusing ones. From each replicate a tree (the bootstrap tree) can be reconstructed using any of the available reconstruction techniques. The support value of a branch in the inferred tree is the proportion of the bootstrap trees that contain this branch. For sequence data, experiments suggest that the derived support values are good estimators of accuracy [9].

The agreement among the bootstrap trees is subject to very complex processes and cannot be directly understood as a statistical measure. Yet practitioners have found it very useful—and the absence of any equivalent measure for reconstructions based on rearrangement data has been a major problem. Small-scale attempts have been made to use a simple jackknife (leave-one-out), but the relatively small number of taxa in typical datasets limits the power of the test; Tang and his group have recently completed a study of a more elaborate strategy that leaves out a combination of both selected genomes and selected genes [23]. For the most part, however, current phylogenetic work with rearrangement data has focused on comparing its results with trees produced from sequence data for the same organisms.

3 Phylogenetic Reconstruction and Accuracy Testing

For reconstruction, we use the distance-based *Neighbor Joining (NJ)* method. Given a matrix of pairwise distances between taxa, NJ reconstructs the phylogeny (including the internal branch lengths) by iteratively joining a closest pair of leaves according to a suitable metric, replacing the two leaves by a "cherry" (the pair of leaves connected to an internal node), computing distances from the cherry to all other leaves, and iterating until only three leaves remain. When the distance matrix is additive, NJ guarantees the reconstruction of the true tree [22].

We study the accuracy of the reconstructed trees and their internal branch lengths through extensive simulations—conducted by generating several trees, simulating evolution on these trees, and using the leaf permutations as inputs to the reconstruction method. The reconstructed trees are compared with the "true" trees to test the accuracy of the method.

We use the Robinson-Foulds (RF) metric [20] to measure the topological accuracy of inferred trees. Every edge e in a leaf-labeled tree defines a bipartition on the leaves: removing e disconnects the tree and thus partitions the set of leaves. If T is the true tree, and T' is the inferred tree, then the false positives are the bipartitions of T' not

present in T, and the false negatives are the bipartitions of T not in T'. Divide each count by $n-3$, the number of internal edges in a binary tree on n leaves: the results are the false positive and false negative rates. The RF distance between two binary trees is the average of the number of false negatives and false positives; the RF error rate is the average of the false negative and false positive rates.

The accuracy of branch length estimation is measured by the average branch length error for each inferred tree: $\Sigma|e_i - t_i|/\Sigma t_i$ where e_i and t_i are the edge lengths of edge i in the inferred tree and true tree, respectively, and the summation is over all edges of the trees.

4 Bootstrapping for Rearrangement-Based Phylogenetic Reconstruction

The classical bootstrapping method cannot be applied directly to rearrangement data because the entire permutation (genome) is a single character in the space of permutations. However, bootstrapping is just one of many possible tests of robustness or repeatability; we therefore design such a test for distance-based reconstructions that takes advantage of the unique characteristics of rearrangement data. The rationale behind our bootstrapping method is to introduce perturbations in the leaf genomes and to reconstruct bootstrap trees from the perturbed leaf genomes. (Note that sampling characters with replacement from leaf sequences, as is done with DNA sequence data, also results in "perturbed" sequences.)

Our method is based on the same two properties used in sequence-based bootstrapping. First, the perturbed data must remain a valid input for phylogenetic reconstruction. Therefore we perturb the leaf genomes by applying random DCJ operations, thus ensuring that the new genomes remain valid inputs; the number of DCJ operations applied to each genome is chosen from a Gaussian distribution, so that the level of perturbation can vary from leaf to leaf. Second, the expected pairwise distance after perturbation must remain close to the distance between the corresponding pair of leaves before perturbation. In our case, if x DCJ operations are applied to leaf i to yield leaf i' and y DCJ operations are applied to leaf j to yield leaf j' (where i and j are in the input dataset and i' and j' in the replicate), and if x and y are not too large, then the expected distance between i' and j' is approximately $(x+y)$ larger than the distance between i and j. To enforce this second property, we produce a distance matrix in which the distance between i' and j' is $d(i',j') - (x+y)$, where $d(i',j')$ is the estimated true distance between i' and j'.

Our procedure produces new distance matrices on the full set of taxa; on each such matrix, it applies NJ to produce a bootstrap tree; and from these trees, it computes support values as in the sequence-based bootstrap.

5 Experimental Design

5.1 Testing Phylogenetic Reconstruction

Our simulation studies follow the standard procedure in phylogenetic reconstruction (see, e.g., [11]): we generate model trees under various parameter settings, then use each

model tree to produce a number of "true trees" on which we evolve artificial genomes from the root down to the leaves (by performing randomly chosen DCJ operations on the current genome) to obtain datasets of leaf genomes for which we know the complete history. We then reconstruct trees and branch lengths for each dataset by computing a distance matrix using our DCJ-based true distance estimator and then using this matrix as input to NJ. We then compute Robinson-Foulds distances and error rates as well as branch-length errors.

A model tree consists of a rooted tree topology and corresponding branch lengths. The trees are generated by a three-step process. We first generate birth-death trees using the tree generator in the software R [19] (with a birth rate of 0.001 and a death rate of 0), which simulates the development of a phylogenetic tree under a uniform, time-homogeneous birth-death process. The branch lengths in this tree are ultrametric, so, in the second step, the branch lengths are modified to eliminate the ultrametricity. Choosing a parameter c, for each branch we sample a number s uniformly from the interval $[-c,+c]$ and multiply the original branch length by e^s (for the experiments in this paper, we set $c = 2$). Thus, each branch length is multiplied by a possibly different random number. Finally, for each branch we rescale its length to achieve a target diameter D for the model tree; each branch length now represents the expected number of rearrangements on that branch. From a single model tree, a set of trees is generated for simulation studies by retaining the same topology and varying the branch lengths by sampling, for each branch in the tree, from a Poisson distribution with a mean equal to that of the corresponding branch length in the model tree.

All experiments are conducted by varying three main parameters: the number of leaves, the number of genes, and the target diameter. The number of leaves in the trees simulated are 100 and 500, the number of genes are $5,000$ and $10,000$ and the target diameters range from $0.5n$ to $4n$, where n is the number of genes. For each setting of the parameters, 100 model trees are generated and from each model tree 10 datasets are created. The error rates for RF and branch length shown in the next section are averages over these $1,000$ trees.

We also test our reconstruction technique on a real dataset: genomes of 6 species from the Ensembl Mercator/Pecan alignments with 8,380 common markers. We selected these genomes for their size, to demonstrate the scalability of our approach, but also because, among vertebrate genomes, they are the best assembled: other vertebrate genomes in the alignment have anywhere from twice to ten times more contigs than the actual chromosomal number of the species.

5.2 Testing the Bootstrapping Method

To test our bootstrapping method, we generate datasets as described above, using 100 leaves, a diameter of 10,000, and genomes of size 5,000. From each inferred tree 100 bootstrap trees are generated. To test different amounts of perturbations, we use Gaussian distributions with means of $0.005n$, $0.01n$, $0.02n$, $0.05n$, $0.1n$, $0.15n$, and $0.2n$ and standard deviations of, respectively, $0.00125n$, $0.0025n$, $0.005n$, $0.02n$, $0.04n$, $0.06n$, and $0.08n$, where n is the number of genes in the dataset. We average the results obtained from 100 such datasets for each set of parameters for the Gaussian distribution.

6 Results and Discussion

6.1 Simulation Studies of the Phylogenetic Reconstruction

Fig. 1 shows RF error rates for various trees. On the left are rates for trees with 100 and 500 species, with genomes of size 5,000 and target diameters ranging from 2,500 to 20,000. The error rates are below 10% in all but the oversaturated cases. On the right are rates for trees of 100 species, with genomes of size 5,000 and 10,000 and diameters varying from half the number of genes to four times that number. As expected, error rates are significantly reduced by an increase in the size of the genome—because the larger number of genes reduces the relative error in the estimated distances.

The corresponding average branch-length errors are shown in Fig. 2. Interestingly, the average error in branch length grows more slowly than the RF error rate with increasing evolutionary diameters.

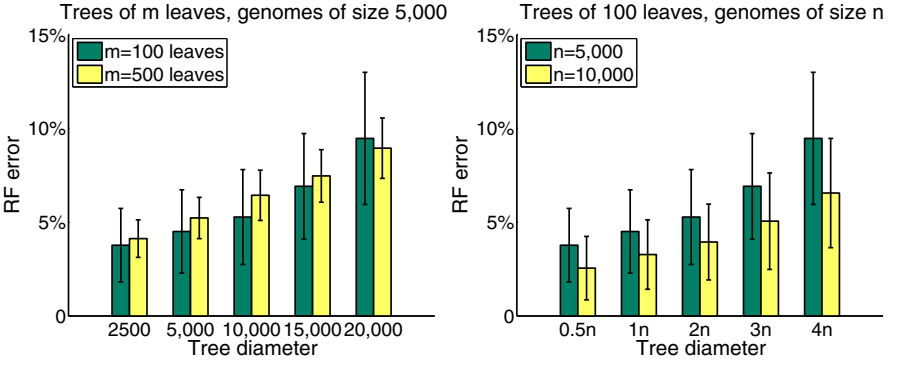


Fig. 1. RF error rates on trees of 100 and 500 leaves with genomes of size 5,000 (left) and on trees of 100 leaves with genomes of size 5,000 and 10,000 (right)

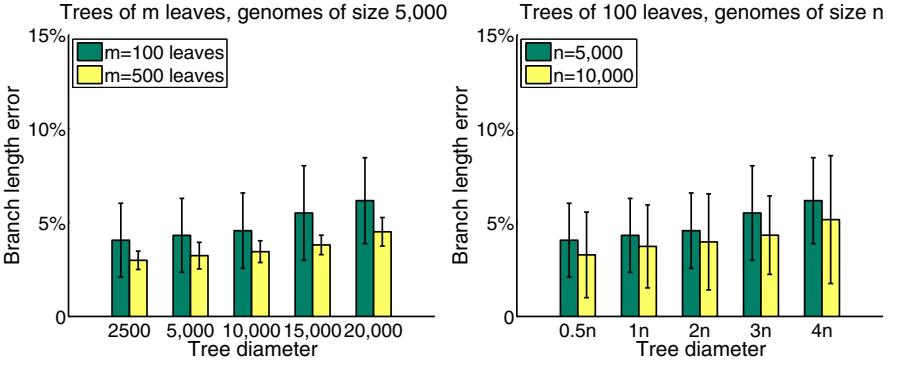


Fig. 2. Average branch length error on trees of 100 and 500 leaves with genomes of size 5,000 (left) and on trees of 100 leaves with genomes of size 5,000 and 10,000 (right)

Overall, these simulations (and many others not shown) confirm that the high precision of our distance estimator makes it possible to reconstruct accurate phylogenies with what is perhaps the simplest of all reconstruction methods, and certainly one of the fastest.

6.2 A Dataset of High-Resolution Vertebrate Genomes

Fig. 3 shows the reconstructed phylogeny of 5 mammals and chicken, along with bootstrap scores for the 3 internal edges of the tree. Building this phylogeny and computing the bootstrap scores (using 100 replicates built with a 10% perturbation rate) took under a second of computing time on a desktop computer; contrast this very fast computation with the fact that no other tool today can handle this size of genome (over 8,000 syntenic blocks) at all, not even in weeks or months of computation. Moreover, the tree is much as expected: the two edges on which the community agrees have perfect bootstrap support, while the more controversial edge creating a primate-carnivore clade has low support—low enough, by normal standards, to be considered untrustworthy. Many studies [15, 18, 2] and most current trees place primates in a clade with rodents rather than with carnivores, although a number of studies support the topology in our figure [12, 31, 6]. Rearrangement data cannot be used at this early stage to settle the rodents vs. carnivores question, but it is encouraging to see that the reconstruction agrees with studies based on sequence data, both on well supported and on poorly supported edges.

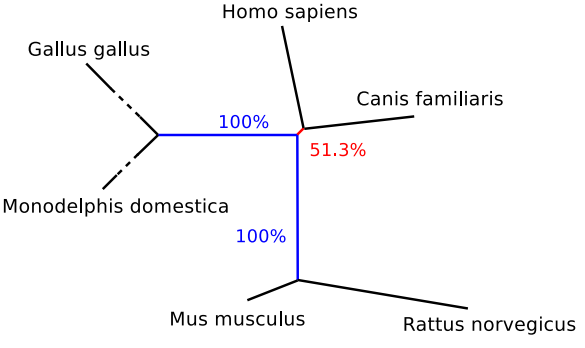


Fig. 3. Reconstructed phylogeny of man, rat, mouse, opossum, dog, and chicken (dotted edges indicate long branches not shown at scale)

The excellent scaling properties of our method and its support for bootstrapping mean that it is now possible to study the use of rearrangement data in phylogenetic reconstruction, so as to improve our understanding of the evolutionary processes at work, parameterize the model, and eventually make whole-genome rearrangement data into a source of information for systematics on a par with today's sequence data.

6.3 Simulation Studies of the Bootstrapping Method

Fig. 4 shows the distribution of bootstrap scores for false positives (edges present in the inferred tree but not in the true tree) on the left and for true positives (edges that appear in both the inferred and the true trees) on the right, binned into three categories: well supported edges (with scores above 90%), poorly supported edges (with scores below 75%), and candidate edges in a "gray" zone of support (from 75% to 90%). The percentages on the vertical axis indicate the distribution of false (on the left) or true (on the right) positives into these three bins. As expected, larger perturbations reduce the support for false positives, but they also reduce the support for true positives. However, the increase in the number of false positives that are placed in the "poorly supported" bin (below 75%) on the left is much more pronounced than the decrease in the number of true positives placed in the "well supported" bin (over 90%) on the right, indicating that perturbation rates as high as 15–20% remain quite usable.

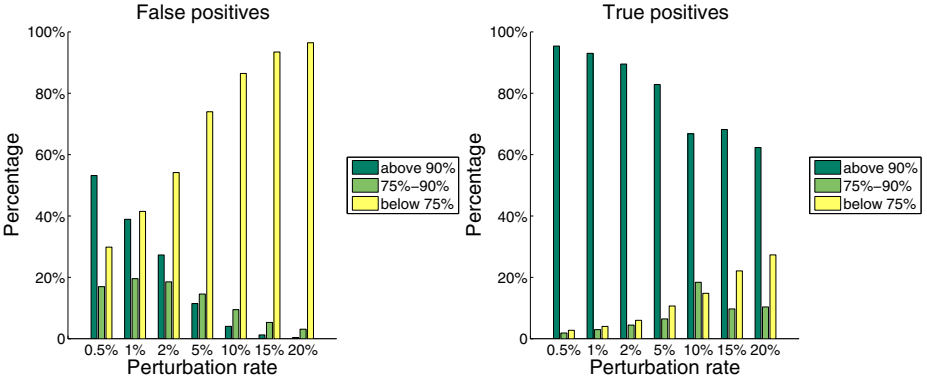


Fig. 4. False positives (left) and True positives (right)

The Receiver-Operator-Characteristic (ROC) curve in Fig. 5 (left) clarifies the issue, showing that changing the level of perturbation simply traces a curve in the ROC space. In this plot, a point is a particular bootstrapping test, defined by its sensitivity and specificity; in the system of coordinates of our figure, therefore, a perfect test would yield a point at the upper left-hand corner of the diagram. Let E be the set of edges in the true tree. The set T_t defined for a threshold t consists of those edges in the inferred tree which are contained in more than $t\%$ of the bootstrap trees. Sensitivity is the proportion of true edges that are also in T_t, $|T_t \cap E|/|E|$, while specificity is the proportion of edges in T_t that are true edges, $|T_t \cap E|/|T_t|$. In our figure, each perturbation rate is represented by 6 points, obtained by selecting values of 95, 90, 85, 80, and 75 for the threshold t. The curve shows that larger perturbations are better at specificity but worse at sensitivity. To keep false positives really low and just lose a few true positives one can achieve a trade-off at around 10% perturbation rate.

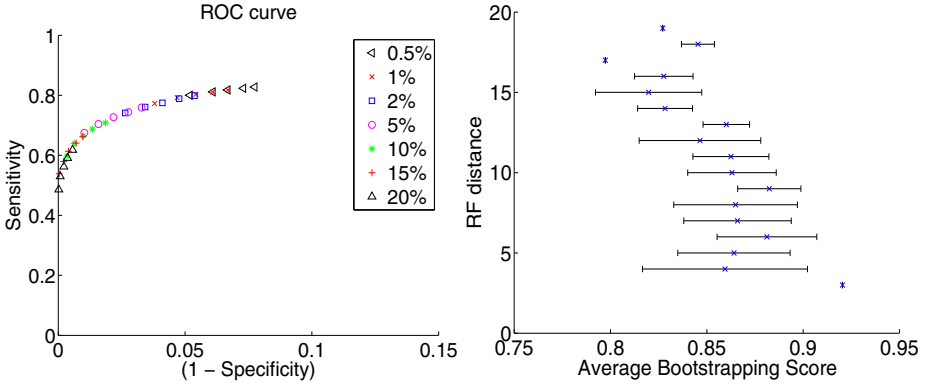


Fig. 5. ROC curve (left) and negative correlation of bootstrap scores and RF distance (right)

Finally, the plot of bootstrapping scores vs. RF distances in Fig. 5 (right) shows that trees with high average bootstrap scores are indeed very close to the true tree. That is, regardless of the choice of perturbation rate and its implications on the sensitivity, the average bootstrap score shows a negative correlation with the RF distance. (The data shown in Fig. 5 is for a perturbation rate of 10%.)

7 Conclusions

We have described a very fast, distance-based, phylogeny reconstruction method for high-resolution rearrangement data and a matching bootstrapping procedure. Both take advantage of some of the unique characteristics of whole-genome rearrangement data, given in terms of syntenic blocks: the absence of duplicates, the equal content among all genomes, and, most importantly, the lack of both homoplasy and saturation in such data, especially when used with high-resolution data. Our simulations demonstrate the accuracy of the reconstruction method and the usefulness of the bootstrapping procedure, and a proof-of-concept application to a small collection of high-resolution vertebrate genomes yields results in line with current findings.

Our methods scale to data of very high resolution (tens of thousands of syntenic blocks) and, because of the very fast running times of distance methods, to large collections of genomes. Therefore, they can be used to study rearrangement data and deepen our understanding of the evolution of the genome, as well as to turn rearrangement data into a genuine source of phylogenetic information.

Acknowledgments

We thank Wei Xu for sharing the vertebrate data and Xiuwei Zhang for many helpful comments.

References

1. Alekseyev, M.A., Pevzner, P.A.: Breakpoint graphs and ancestral genome reconstructions. Genome Research 19(5), 943–957 (2009)
2. Amrine-Madsen, H., Koepfli, K.-P., Wayne, R.K., Springer, M.S.: A new phylogenetic marker, apolipoprotein b, provides compelling evidence for eutherian relationships. Molecular Phylogenetics and Evolution 28(2), 225–240 (2003)
3. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
4. Blanchette, M., Bourque, G., Sankoff, D.: Breakpoint phylogenies. In: Miyano, S., Takagi, T. (eds.) Genome Informatics, pp. 25–34. Univ. Academy Press, Tokyo (1997)
5. Bourque, G., Pevzner, P.: Genome-scale evolution: reconstructing gene orders in the ancestral species. Genome Res. 12, 26–36 (2002)
6. Cannarozzi, G., Schneider, A., Gonnet, G.: A phylogenomic study of human, dog, and mouse. PLoS Comput. Biol. 3, e2 (2007)
7. Day, W.H.E., Sankoff, D.: The computational complexity of inferring phylogenies from chromosome inversion data. J. Theor. Biol. 127, 213–218 (1987)
8. Felsenstein, J.: Confidence limits on phylogenies: an approach using the bootstrap. Evol. 39, 783–791 (1985)
9. Felsenstein, J., Kishino, H.: Is there something wrong with the bootstrap on phylogenies? A reply to Hillis and Bull. Syst. Biol. 42(2), 193–200 (1993)
10. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. MIT Press, Cambridge (2009)
11. Hillis, D.M., Huelsenbeck, J.P.: Assessing molecular phylogenies. Science 267, 255–256 (1995)
12. Huttley, G.A., Wakefield, M.J., Easteal, S.: Rates of genome evolution and branching order from whole-genome analysis. Mol. Biol. Evol. 24(8), 1722–1730 (2007)
13. Lin, Y., Moret, B.M.E.: Estimating true evolutionary distances under the DCJ model. In: Proc. 16th Int'l Conf. on Intelligent Systems for Mol. Biol. (ISMB 2008). Bioinformatics, vol. 24(13), pp. i114–i122 (2008)
14. Lin, Y., Rajan, V., Swenson, K.M., Moret, B.M.E.: Estimating true evolutionary distances under rearrangements, duplications, and losses. In: Proc. 8th Asia Pacific Bioinf. Conf (APBC 2010). BMC Bioinformatics, vol. 11(suppl. 1), p. S54 (2010)
15. Madsen, O., Scally, M., Douady, C.J., Kao, D.J., DeBry, R.W., Adkins, R., Amrine, H.M., Stanhope, M.J., de Jong, W.W., Springer, M.S.: Parallel adaptive radiations in two major clades of placental mammals. Nature 409, 610–614 (2001)
16. Moret, B.M.E., Tang, J., Wang, L.-S., Warnow, T.: Steps toward accurate reconstructions of phylogenies from gene-order data. J. Comput. Syst. Sci. 65(3), 508–525 (2002)
17. Moret, B.M.E., Wyman, S.K., Bader, D.A., Warnow, T., Yan, M.: A new implementation and detailed study of breakpoint analysis. In: Proc. 6th Pacific Symp. on Biocomputing (PSB 2001), pp. 583–594. World Scientific Pub., Singapore (2001)
18. Murphy, W.J., Eizirik, E., Johnson, W.E., Zhang, Y.P., Ryder, O.A., O'Brien, S.J.: Molecular phylogenetics and the origins of placental mammals. Nature 409, 614–618 (2001)
19. R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2009), ISBN 3-900051-07-0
20. Robinson, D.R., Foulds, L.R.: Comparison of phylogenetic trees. Mathematical Biosciences 53, 131–147 (1981)
21. Rokas, A., Holland, P.W.H.: Rare genomic changes as a tool for phylogenetics. Trends in Ecol. and Evol. 15, 454–459 (2000)

22. Saitou, N., Nei, M.: The neighbor-joining method: A new method for reconstructing phylogenetic trees. Mol. Biol. Evol. 4, 406–425 (1987)
23. Shi, J., Zhang, Y., Luo, H., Tang, J.: Using jackknife to assess the quality of gene order phylogenies. BMC Bioinformatics 11(168) (2010)
24. Soltis, P.S., Soltis, D.E.: Applying the bootstrap in phylogeny reconstruction. Statist. Sci. 18(2), 256–267 (2003)
25. Sturtevant, A.H.: A crossover reducer in Drosophila melanogaster due to inversion of a section of the third chromosome. Biol. Zent. Bl. 46, 697–702 (1926)
26. Sturtevant, A.H., Dobzhansky, T.: Inversions in the third chromosome of wild races of drosophila pseudoobscura and their use in the study of the history of the species. Proc. Nat'l Acad. Sci., USA 22, 448–450 (1936)
27. Swofford, D.L., Olsen, G.J., Waddell, P.J., Hillis, D.M.: Phylogenetic inference. In: Hillis, D.M., Mable, B.K., Moritz, C. (eds.) Molecular Systematics, pp. 407–514. Sinauer Assoc., Sunderland (1996)
28. Tang, J., Moret, B.M.E.: Scaling up accurate phylogenetic reconstruction from gene-order data. In: Proc. 11th Int'l Conf. on Intelligent Systems for Mol. Biol (ISMB 2003). Bioinformatics, vol. 19, pp. i305–i312. Oxford U. Press, Oxford (2003)
29. Wang, L.-S.: Exact-IEBP: a new technique for estimating evolutionary distances between whole genomes. In: Proc. 33rd Ann. ACM Symp. Theory of Comput (STOC 2001), pp. 637–646. ACM Press, New York (2001)
30. Wang, L.-S., Warnow, T.: Estimating true evolutionary distances between genomes. In: Gascuel, O., Moret, B.M.E. (eds.) WABI 2001. LNCS, vol. 2149, pp. 176–190. Springer, Heidelberg (2001)
31. Wildman, D.E., Uddin, M., Opazo, J.C., Liu, G., Lefort, V., Guindon, S., Gascuel, O., Grossman, L.I., Romero, R., Goodman, M.: Genomics, biogeography, and the diversification of placental mammals. Proc. Nat'l Acad. Sci., USA 104(36), 14395–14400 (2007)
32. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21(16), 3340–3346 (2005)
33. Zwickl, D.J., Hillis, D.M.: Increased taxon sampling greatly reduces phylogenetic error. Syst. Biol. 51(4), 588–598 (2002)

A Simple Measure of the Dynamics of Segmented Genomes: An Application to Influenza

Stéphane Aris-Brosou[1,2]

[1] Department of Biology and Center for Advanced Research in Environmental Genomics, University of Ottawa, Ottawa, ON K1N 6N5, Canada
[2] Department of Mathematics and Statistics, University of Ottawa
sarisbro@uottawa.ca

Abstract. The severity of influenza epidemics, which can potentially become a pandemic, has been very difficult to predict. However, past efforts were focusing on gene-by-gene approaches, while it is acknowledged that the whole genome dynamics contribute to the severity of an epidemic. Here, putting this rationale into action, I describe a simple measure of the amount of reassortment that affects influenza at a genomic scale during a particular year. The analysis of 530 complete genomes of the H1N1 subtype, sampled over eleven years, shows that the proposed measure explains 58% of the variance in the prevalence of H1 influenza in the US population. The proposed measure, denoted nRF, could therefore improve influenza surveillance programs at a minimal cost.

1 Introduction

In March 2009, a new influenza A virus emerged in Mexico and in the United States, and spread quickly to the rest of the world in the following weeks. On May 11, the World Health Organization declared the situation as the first pandemic of the 21st century (e.g., [21]). A number of studies have now confirmed that this particular virus emerged following a series of particular exchanges of genetic material between viruses circulating in different hosts (e.g., [21]). One lesson learned during this outbreak is that nobody expected this very particular virus to emerge and to cause a human pandemic. This lack of perspective can essentially be attributed to the very traditional way the dynamics of these genomes are monitored: by studying the history (phylogeny) of each segment (i) independently and (ii) over a long period of time simultaneously analyzing several years (e.g., [12,15,21]). The objective of this work is to provide us with a simple tool that can be used to assist surveillance programs by monitoring the dynamics of influenza viruses at the genomic level, (i) integrating the information about the history of all segments simultaneously and (ii) on a yearly basis in order to be able to track the dynamics of these genomes in time.

Influenza viruses are the etiologic agents of the 'flu', a seasonal illness that in a 'regular' season kills 250,000-500,000 people, globally [18]. The virus itself is

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 149–160, 2010.

made of a protein capsid that encloses its genome, along with a few structural proteins. The influenza genome is composed of eight segments of negative-sense single-stranded RNA molecules, each of which encodes 1-2 proteins for a total of 12 proteins [24]. By approximate order of decreasing size, these genes code for polymerase subunits (PB2, PB1 and PA), the hemagglutinin (HA) and neuraminidase (NA) antigens, a nucleoprotein (NP), a ribonucleoprotein exporter (NS2, also called NEP), an interferon antagonist (NS1), an ion channel protein (M2) and a matrix protein (M1). The last two of the twelve proteins, PB2-F1 [5] and PB1-N40 [24], are less well characterized. Five major different types of influenza viruses are recognized, A B and C being the three most common. The bases of the division into types is made according to genetic information (phylogenies), mutation rates (A > B > C) or host range (A: a large number of vertebrates, B: humans and seals, C: humans and swines).

Influenza A viruses are the principal source of epidemics in the human population, and unlike the other two common types, are further subdivided into subtypes. This subtype classification is based on the type of HA and NA proteins, and therefore genes, that each virus is made of. Almost all combinations can be formed between the 16 known subtypes of HA (H1-H16) and the nine that are known for NA (N1-N9) [11, p.157]. All of these subtypes are present in wild waterfowl, but the most prevalent subtypes in the human population are H1N1 and H3N2 [18].

Because of their structure as single-stranded RNA molecules and as segmented genomes, influenza A viruses evolve quickly under two general mechanisms: antigenic drift and antigenic shift [14]. Antigenic drift is caused by the accumulation of mutations, which occur at a high rate ($\sim 10^{-3}$ substitutions per site per year [18]) due to the lack of a proof-reading mechanism during replication of the viral genome, while antigenic shift is due to the exchange of segments when at least two different viruses, potentially of different subtypes, co-infect the same cell. Such an exchange is called reassortment. With the potential to generate new combinations of antigenes and potentially new subtypes, reassortment is the main source of antigenic novelty [15,14], and has been directly implicated in the emergence of the 2009 pandemic [21]. However, the methods used to quantify reassortments are virtually inexistent. The current practice consists, first, in estimating a phylogenetic tree for each individual segment for a set of genomes sampled through many years, and then, in reconciling these trees, optimally using a cophylogenetic method [4] in order to obtain a snapshot of the history of reassortment. Although this approach allows us to reconstruct what happened *a posteriori* [21], it does not allow us to quantify the dynamics of influenza genomes in real time. Hence, the predictive power of the current approach is vanishingly small.

Here I describe a method that permits the quantification of the dynamics of viruses with segmented genomes, and apply it to a follow-up of H1N1 influenza A viruses through the eleven years between 2000 and 2010. The method, called nRF, is based on a measure of the dissimilarity of gene trees estimated for the different segments of the genome. To use nRF, I further introduce the notion

of a *genome tree*, which is a tree whose leaves represent the different genes constituting the influenza genome, and whose topology represents the amount of reassortment occurring between the different segments. If reassortment actually drives the evolution of influenza genomes [14], we should expect to observe a correlation between a quantitative measure of reassortment, namely, nRF, and prevalence (proportion of affected individuals in a given population) of the viruses under study. I show here that the nRF measure is able to explain 58% of the variance in prevalence of H1 influenza in the US population.

2 Methods

2.1 Rationale

Under the assumptions that (i) reassortment drives the evolution of influenza genomes [14] and (ii) there is no recombination [3, 2], the motivation is to find a measure of reassortment, at the scale of the genomes, which are sampled on a yearly basis. For now, I further assume that all the phylogenetic inferences below return the "true" tree, an assumption to which I come back later in section 2.3.

In the absence of reassortment, all ten segments should have exactly the same phylogeny. Therefore, if we compute the pairwise distances between estimated gene topologies, for instance with the Robinson and Foulds (RF) distance [19], we should obtain a matrix of pairwise distances full of zeros. Briefly, the RF distance, also called the symmetric distance, is the number of branch partitions that differ between two topologies.

On the other hand, if a segment has undergone a reassortment event, then its estimated phylogeny should differ from the others'. The matrix of pairwise distances computed as above should now contain a row and a column of nonzero entries. If we use this matrix of pairwise distances between the gene trees to build a new tree, that I call here a *genome tree*, its total tree length will be a measure of the amount of reassortment that occurred between the different segments (*genes*, in actuality). Reciprocally, the proposed measure can also be understood as a measure of linkage, in the sense that segments transmitted together will cluster together in the estimated genome tree.

Because different years can have different numbers of sampled genomes, RF distances should be scaled by their maximum value, $2(n-3)$ for a tree with n leaves, within each year, so that they become comparable across several years. Hereafter, I denote this measure nRF for normalized RF distance.

The hypothesis of interest is then that a correlation should be observed between nRF and a measure of prevalence of influenza A H1N1 in the human population. To test this hypothesis, I used data on the US population affected by H1 viruses from 2000-2010, as available at the US Centers for Disease Control website (`www.cdc.gov/flu/weekly`). Note that these data are theoretically for all H1 viruses, which include several subtypes; this is related to the assay used to type samples; in practice however, the circulation of subtypes other than H1N1 is negligible in the human population. Because (i) the prevalence data is available by season (in the Northern hemisphere: weeks 40-20), while the genome

data collected for this study run from week 1-52 (see below) and (ii) I specifically want to test for the predictive power of the method, I shifted the prevalence data to compare nRF during year y with prevalence during y, rather than the season running between week 40 of year y to week 20 of year $y + 1$.

2.2 Algorithm

The procedure is divided into six steps:

1. sample influenza genomes for one particular year;
2. extract protein-coding sequences (CDSs) within each segment, translate to amino acid sequences [9], align [6] and back-translate to DNA alignments [22];
3. for each CDS, reconstruct phylogenetic trees under maximum likelihood with a mixture of NNI and SPR searches [10], assuming the GTR $+\Gamma$ model of substitution [26, p.33, 44] (optimally: select the most appropriate model of evolution for each data set [16]);
4. compute the matrix of RF distances [8] for each pair of trees and scale by $2(n-3)$ (for n sequences in alignment) to obtain nRF;
5. reconstruct the Neighbor-Joining tree [8] from the matrix of pairwise RF distances to visualize "correlation" among CDSs [optional];
6. compute the tree length of the NJ tree built on normalized RF distance.

The procedure is repeated for as many years as desired or, in practice, years for which there are "enough" available data ($\geq$ 4 genomes, since with $<$ 4 genomes there is only one single unrooted topology and the phylogenetic problem becomes nonexistent). Perl scripts were written to make most of these steps automatic. Unless otherwise specified, the default settings were used for all the programs cited in the above algorithm. Computations were distributed with the `ForkManager` library (available at `search.cpan.org/dist/Parallel-ForkManager/`).

Branch lengths can affect the comparison of two trees in the following way. Consider the two rooted trees ((1, 2) ,3) and (1, (2, 3)). They have different topologies, but if we consider their respective branch lengths, then the same two trees ((1:0.1, 2:0.1):0.001, 3:0.1) and (1:0.1, (2:0.1, 3:0.1):0.001) are actually very similar. Although the trees considered here are all unrooted, a similar effect of branch lengths could artificially increase the effect of reassortment from the perspective of the RF distance. Therefore, in addition to the RF distance, I also computed the Branch Score Distance or BSD [13,8]. As with nRF, I denote the scaled measure $nBSD$.

2.3 Measure of Support

In order to estimate confidence intervals for both nRF and $nBSD$, I performed a bootstrap analysis [7] during step 3 in the algorithm given above. Here I used 100 replicates to keep computations to a minimum while demonstrating the concept. Optimally, one to ten thousand replicates would be preferable.

Steps 4–end of the algorithm are then repeated on each bootstrap replicate. For simplicity's sake, I considered that a bootstrapped genome would be formed by the bootstrap replicates taken replicate by replicate: bootstrapped genome 0, denoted b_0^G, was formed by the set of bootstrapped genes $\{b_0^{\mathrm{PB2}}, b_0^{\mathrm{PB1}}, \ldots\ b_0^{\mathrm{NS1}}\}$, b_1^G was constructed as $\{b_1^{\mathrm{PB2}}, b_1^{\mathrm{PB1}}, \ldots\ b_1^{\mathrm{NS1}}\}$, *etc.*. This matches the null assumption of independence between segments (*genes*, here), although it might be preferable to take physical linkage into account for the genes on segments 7 on the one hand (genes M2 and M1) and 8 on the other hand (genes NS2 and NS1).

The bootstrapped trees were summarized by computing the majority-rule consensus tree [7,8], constructed from the bipartitions that appeared in at least 50% of the bootstrapped replicates. Just like with the computation of bootstrapped nRF distances, a bootstrap genome was formed by the bootstrap replicates taken replicate by replicate.

2.4 Sampled Genomes

All publicly-available complete influenza A genomes of subtype H1N1 were extracted from the National Center for Biotechnology Information (available at `www.ncbi.nlm.nih.gov`, [1]) for the eleven years spanning 2000-2010. As of May 26, 2010, a total of 2,435 genomes were available (2000: 80 genomes; 2001: 118; 2002: 8; 2003: 26; 2004: 8; 2005: 25; 2006: 19; 2007: 326 [100 of which were randomly chosen for computational expediency]; 2008: 70; 2009: 1706 [100 of which were randomly chosen]; 2010: 49), sampling across swine and human hosts distributed around the world. Alignments were visually inspected and edited where necessary [23], with a particular attention to genes on segments 2, 7 and 8, which are occasionally misannotated; dubious entries were discarded, which demanded to check that each year had sequences from exactly the same individuals. As a result, the final data sets had the following sizes: 2000: 69; 2001: 96; 2002: 7; 2003: 18; 2004: 7; 2005: 19; 2006: 16; 2007: 100; 2008: 54; 2009: 100; 2010: 44, for a total number of genomes equal to 530. Over the sampled years, the distribution of genomes coming from the US was very uneven (actual numbers: 2000: 2; 2001: 24; 2002: 0; 2003: 13; 2004: 0; 2005: 1; 2006: 6; 2007: 89; 2008: 11; 2009: 44; 2010: 36), with years 2000, 2002, 2004 and 2005 having too few US genomes to perform any phylogenetic study (hence the worldwide genome sampling adopted here). Because the PB2-F1 and PB1-N40 genes are small and not always present and / or correctly annotated, I focused on the ten 'canonical' genes: PB2, PB1, PA, HA, NP, NA, M2, M1, NS2 and NS1. Note that the M genes are both on the same segment (# 7); likewise, the NS genes are both on segment # 8. All alignments used in this study are available at `www.bioinformatics.uottawa.ca/stephane`.

3 Results

3.1 Genome Dynamics

Figure 1 shows the trees based on nRF, the proposed measure for monitoring genome dynamics. A number of results are clear from that figure. First,

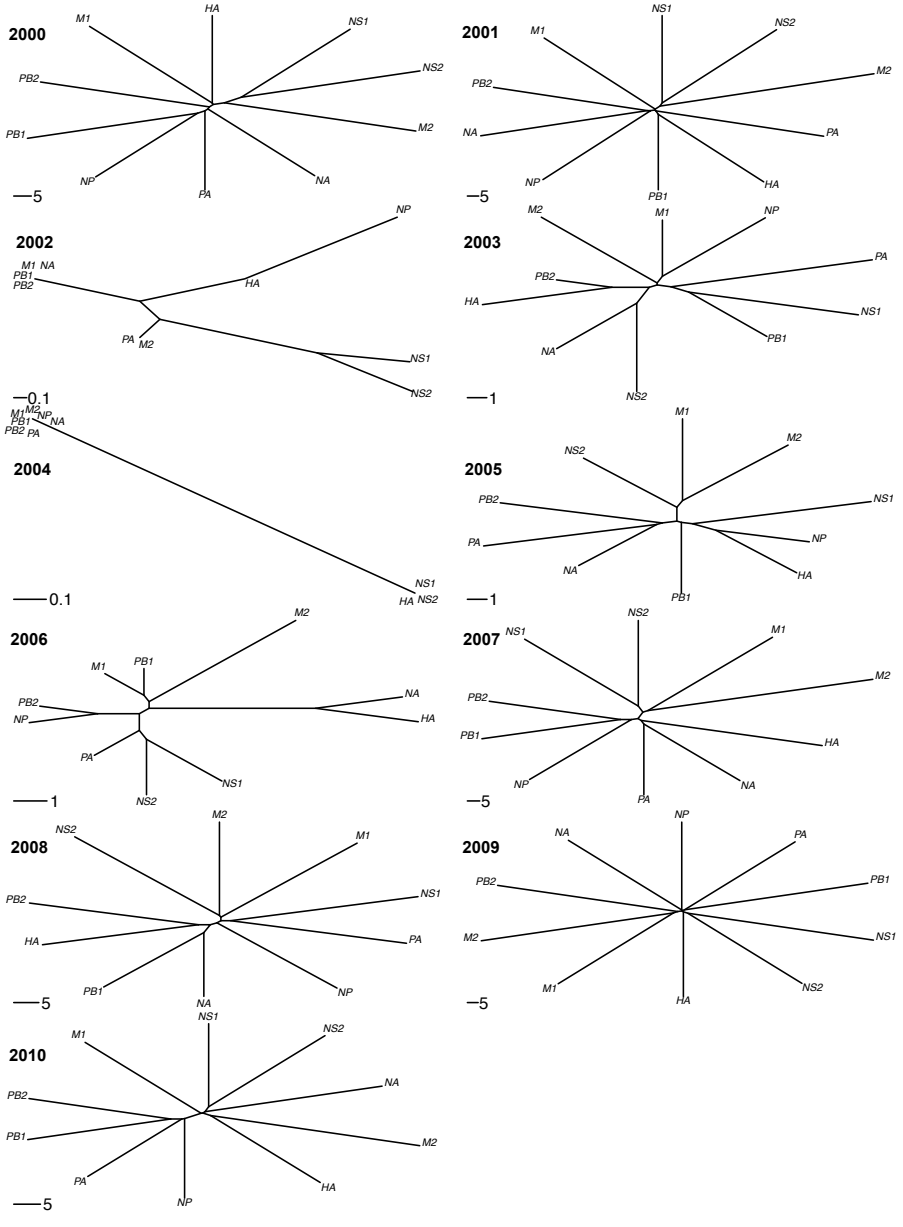


Fig. 1. Genome trees estimated over the eleven years sampled. RF distances were estimated between the gene trees for the ten 'canonical' protein-coding genes of influenza A genomes, and genome trees were reconstructed from the from the pairwise RF distances by NJ. Bootstrap support values not shown (see Figure 2).

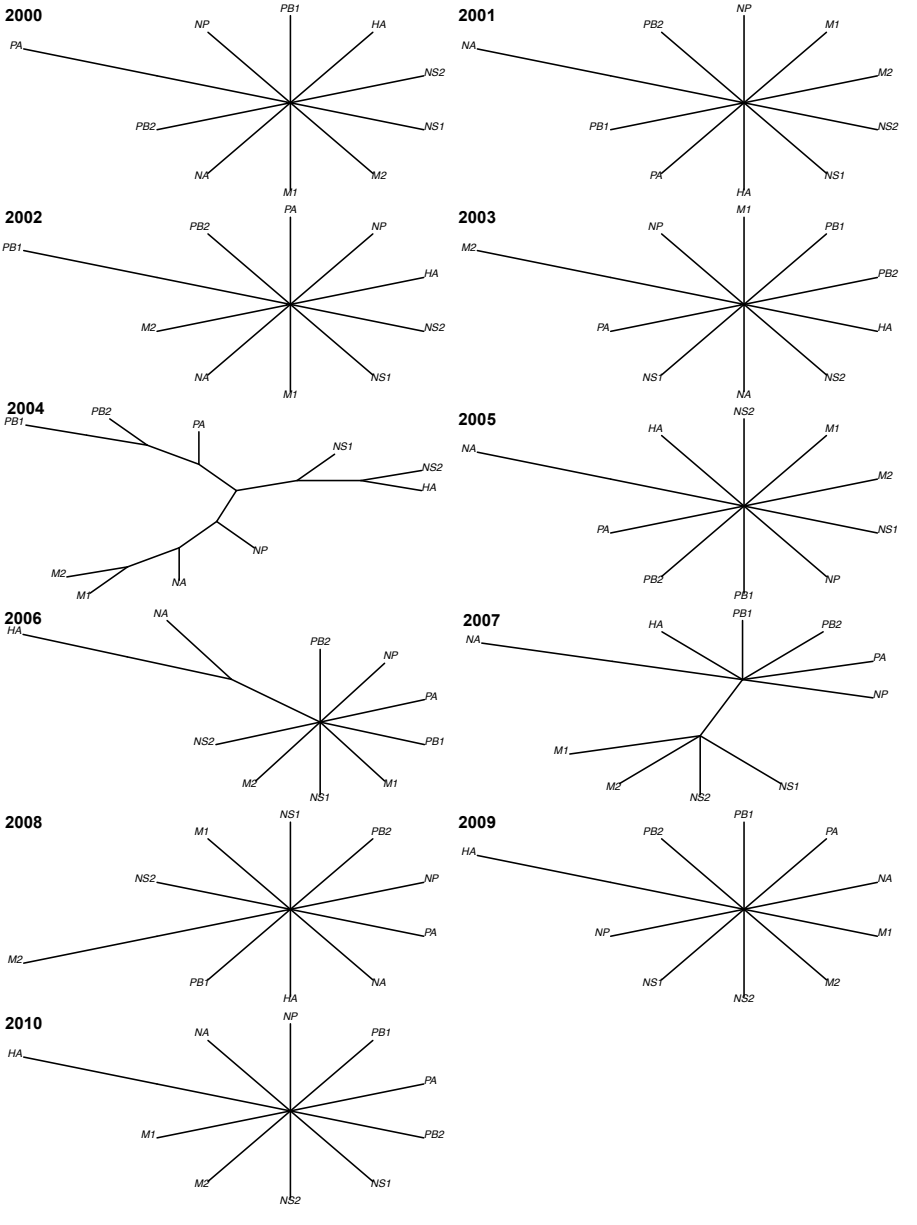


Fig. 2. Majority-rule consensus of the 100 bootstrapped genome trees estimated over the eleven years sampled. Internal branch lengths are nonzero only if the node bipartition they sustain is > 0.75; otherwise, they are set to an arbitrary nonzero number.

reassortment is a rampant process affecting these viruses every year. In the eleven years analyzed here, all the genome trees have a nonzero tree length.

Second, there is extensive variation in the amount of reassortment observed from year to year, with tree lengths in unit of normalized RF distance ranging from 0.05 in 2004 to 0.42 in 2009 (which, according to the confidence intervals calculated below, is significant). Despite the fact that 2004 is the year where only seven genomes were analyzed and 2009 comprises 100 genomes (see Methods), a robust linear regression [27] shows that there is no evidence for an association between tree lengths and the size of the data sets analyzed at the 1% level ($P = 0.037$). This result shows that the nRF measure is robust to sample size.

Third, linkage or reassortment patterns are expected, such as those for the M and NS genes, which are respectively encoded on segments 7 and 8. This is visible on trees where the M2/M1 and NS2/NS1 genes cluster together. But in most years (eight years out of eleven, or 73% of the time for the M genes), the pattern is due to very short internal branch lengths, and is probably artifactual. Indeed, when the consensus trees are computed over the 100 bootstrap replicates (Figure 2), almost all evidence of linkage disappears. Some exceptions exist, such as in 2006 and 2007, but the consensus trees show that (i) all segments are extensively exchanged among individual viruses, so that no clear linkage seems to exist, and (ii) the year of the H1N1 pandemic, 2009, was no exception: this particular pandemic was not preceded and does not exhibit any particular linkage between segments or the genes encoded on these segments.

3.2 *nRF* and *nBSD* as Predictors of Prevalence

Figure 3 shows the estimated nRF values plotted against the yearly log prevalence of H1 infections in the US population. The linear model fitted to these data is highly significant ($F_{1,9} = 14.59$; $P = 0.0041$; adjusted $R^2 = 0.576$). This means that more than half (58%) of the prevalence of influenza H1 in the US is determined by the genome dynamics at the global scale as measured by nRF. A robust linear regression gives similar results ($P = 0.0006$, $R^2 = 0.181$; P(M-bias) $= 0.7402$, P(LS-bias) $= 0.8850$), and this regression remained highly significant even after removing the 2004 data ($P = 7 \times 10^{-6}$, $R^2 = 0.271$; P(M-bias) < 0.0001, P(LS-bias) $= 0.7294$). Therefore, the regression is not solely driven by this point (see Figure 3), and the predictive power of nRF is not an artifact of the sampled data.

We also fitted the same kind of model with nRF of H1 viruses against the prevalence of H3 viruses. In this case, nRF carries no predictive power for this influenza subtype ($F_{1,9} = 3.22$; $P = 0.1063$; adjusted $R^2 = 0.182$), as expected since H1 and H3 are different subtypes.

However, contrary to the expectation that including knowledge of branch lengths might improve prediction, there was still some significant predictive power with $nBSD$ ($F_{1,9} = 10.12$; $P = 0.0111$), but R^2 decreased to 0.477 (compared to 0.576 with nRF). This lack of significance at the 1% level is probably due to the rate heterogeneity across segments, which can be substantial [18]; indeed, in presence of such rate heterogeneity, estimated branch lengths are going

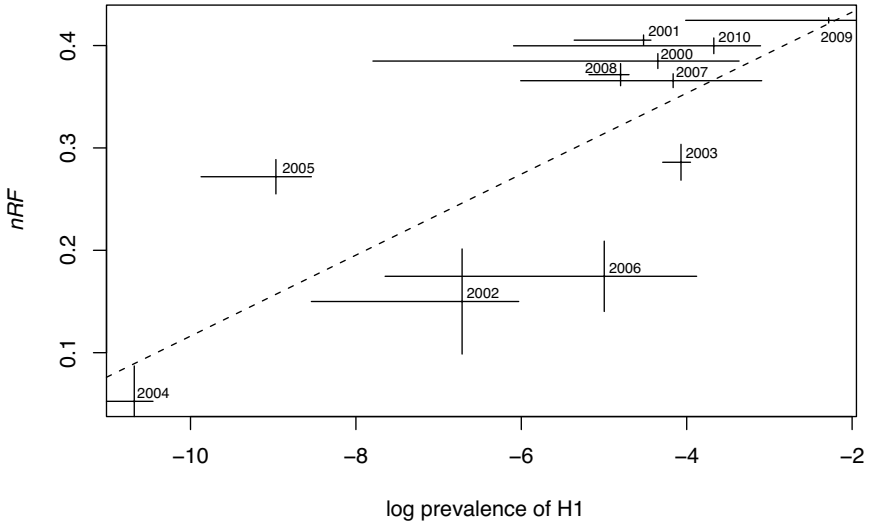


Fig. 3. *nRF* **as a function of the log prevalence of H1 subtypes.** The broken line represents the fitted regression ($P = 0.0041$). Vertical bars: 2 standard deviations; horizontal bars: upper and lower 5% quantiles of the distribution of prevalence over a calendar year.

to be dramatically different from one segment to the other, which is expected to increase BSD if the tree topologies compared share at least one branch bipartition.

4 Discussion

This work represents a proof of concept demonstrating the possibility of monitoring the severity of epidemics based on a simple measure. The measure described, *nRF*, intuitively relies on our understanding of the basic biology of RNA viruses, in that reassortment of segmented viral genomes generates new properties with respect to antigenicity, pathogenicity, virulence *etc.*, all of which can compromise the integrity of the immune system of their hosts, and potentially lead to epidemics or pandemics. Our results demonstrate that *nRF* is a powerful proxy for prevalence, at least in the case of influenza A viruses of subtype H1 over the eleven years and 530 genomes studied here.

One of the reasons why the method has good predictive power is because intra-segmental recombination is negligible in influenza A viruses [3, 2]. If this were not the case, then recombination would affect each individual gene tree and would confound the reassortment signal. As a result, the method described here only applies to non-recombining segmented viral genomes such as influenza A viruses. Independent tests of the *nRF* measure should be performed on other such viruses, which probably include most single-stranded negative sense segmented RNA viruses [17] (for a full list of segmented RNA viruses, see [11, p.4]),

for which we have prevalence data; potential candidates include the Rift Valley Fever virus [20], as well as Ebolavirus and Marburgvirus or Lyssavirus (rabies). An extension to recombining genomes is however possible through the use of ancestral recombination graphs [25].

In the context of the influenza data analyzed here, a number of points might demand to be refined in order to fully demonstrate the power of the approach. First, sampling was done on a yearly basis, not on a seasonal basis, and genomes were sampled on a global scale while compared to prevalence data from the US only. Yet, I was able to demonstrate the existence of a significant ($P = 0.0041$) and reasonably good predictive power ($R^2 = 0.576$). This suggests two mutually exclusive interpretations: (i) the approach returns a random result, which is unlikely given both the size of the data analyzed (eleven years, 530 complete genomes) and the robustness of the results to the use of robust linear regression; (ii) prevalence in the US is representative of the global dynamics of influenza A subtype H1, and that the nRF method is quite powerful in predicting the severity of an epidemic as measured by prevalence. Second, intra-host dynamics were not studied, as a few genomes from swine hosts were present in the data analyzed here. It is likely that the inclusion of these genomes from swine hosts did not affect the results because most of the genomes (93.21%) were coming from human hosts. Third, the rates of evolution, in units of substitutions per site per year, were not analyzed in correlation with the nRF measure of genome dynamics. Considering rates of evolution in conjunction with nRF would represent a further development of the method. Note however that all the three points above are linked to the sampling scheme used in this work, and therefore do not affect the approach described to monitor genome dynamics through time by means of nRF. A last potential caveat is the expected correlation between prevalence and sequencing effort: it can indeed be expected that years of high prevalence lead to an increased surveillance effort and hence to the deposition of a larger number of genomes in publicly-available databases, such as in 2009 with the H1N1 pandemic. Although there was no significant association between the length of genomes trees and the number of fully sequenced genome in public repositories (section 3.1), more independent data sets should be examined to further demonstrate the power of the nRF measure.

Could we have predicted the 2009 pandemic? Maybe, in the sense that nRF measures for 2009 (evaluated from data sampled between the end of the 2008-2009 season and the beginning of 2009-2010 season) were relatively high. Yet, the question of the existence of a threshold for nRF above which the emergence of a pandemic becomes likely is still open, as nRF measures for 2001 were also high (Figure 3), albeit significantly lower than in 2009, and no pandemic occurred back in 2001.

Finally, in the wake of the 2009 pandemic, it has been realized that the current surveillance system failed to track the emerging pathogens [21]. The approach and results presented here demonstrate that even in the absence of such information on genomes from the past, it is still possible to implement a cost-effective warning system based on the nRF of *currently* circulating genomes, provided

that the surveillance programs track full influenza genomes rather than limiting their effort to HA and NA sequencing.

Acknowledgments

I wish to thank three reviewers for very insightful comments that helped improve this paper. This work was funded by the Natural Sciences Research Council of Canada and by the Canada Foundation for Innovation to SAB.

References

1. Bao, Y., Bolotov, P., Dernovoy, D., Kiryutin, B., Zaslavsky, L., Tatusova, T., Ostell, J., Lipman, D.: The influenza virus resource at the National Center for Biotechnology Information. J. Virol. 82(2), 596–601 (2008)
2. Boni, M.F., de Jong, M.D., van Doorn, H.R., Holmes, E.C.: Guidelines for identifying homologous recombination events in influenza A virus. PLoS One 5(5), e10434 (2010)
3. Boni, M.F., Zhou, Y., Taubenberger, J.K., Holmes, E.C.: Homologous recombination is very rare or absent in human influenza A virus. J. Virol. 82(10), 4807–4811 (2008)
4. Charleston, M.A., Perkins, S.L.: Traversing the tangle: algorithms and applications for cophylogenetic studies. J. Biomed. Inform. 39(1), 62–71 (2006)
5. Chen, W., Calvo, P.A., Malide, D., Gibbs, J., Schubert, U., Bacik, I., Basta, S., O'Neill, R., Schickli, J., Palese, P., Henklein, P., Bennink, J.R., Yewdell, J.W.: A novel influenza A virus mitochondrial protein that induces cell death. Nat. Med. 7(12), 1306–1312 (2001)
6. Edgar, R.C.: MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. 32(5), 1792–1797 (2004)
7. Felsenstein, J.: Confidence limits on phylogenies: An approach using the bootstrap. Evolution 39(4), 783–791 (1985)
8. Felsenstein, J.: PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle (2005)
9. Gilbert, D.: Sequence file format conversion with command-line readseq. Curr Protoc Bioinformatics Appendix 1, Appendix 1E (2003)
10. Guindon, S., Delsuc, F., Dufayard, J.F., Gascuel, O.: Estimating maximum likelihood phylogenies with PhyML. Methods Mol. Biol. 537, 113–137 (2009)
11. Holmes, E.C.: The evolution and emergence of RNA viruses. Oxford series in ecology and evolution. Oxford University Press, Oxford (2009)
12. Holmes, E.C., Ghedin, E., Miller, N., Taylor, J., Bao, Y., St. George, K., Grenfell, B.T., Salzberg, S.L., Fraser, C.M., Lipman, D.J., Taubenberger, J.K.: Whole-genome analysis of human influenza a virus reveals multiple persistent lineages and reassortment among recent h3n2 viruses. PLoS Biol 3(9), e300 (2005)
13. Kuhner, M.K., Felsenstein, J.: A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol. Biol. Evol. 11(3), 459–468 (1994)
14. Nelson, M.I., Holmes, E.C.: The evolution of epidemic influenza. Nat. Rev. Genet. 8(3), 196–205 (2007)

15. Nelson, M.I., Simonsen, L., Viboud, C., Miller, M.A., Taylor, J., George, K.S., Griesemer, S.B., Ghedin, E., Ghedi, E., Sengamalay, N.A., Spiro, D.J., Volkov, I., Grenfell, B.T., Lipman, D.J., Taubenberger, J.K., Holmes, E.C.: Stochastic processes are key determinants of short-term evolution in influenza A virus. PLoS Pathog 2(12), e125 (2006)
16. Posada, D., Crandall, K.A.: MODELTEST: testing the model of DNA substitution. Bioinformatics 14(9), 817–818 (1998)
17. Pringle, C.R.: The Bunyaviridae, chap. Genetics and genome segment reassortment, pp. 189–226. Plenum Press, New York (1996)
18. Rambaut, A., Pybus, O.G., Nelson, M.I., Viboud, C., Taubenberger, J.K., Holmes, E.C.: The genomic and epidemiological dynamics of human influenza A virus. Nature 453(7195), 615–619 (2008)
19. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Mathematical Biosciences 53(1-2), 131–147 (1981)
20. Sall, A.A., Zanotto, P.M., Sene, O.K., Zeller, H.G., Digoutte, J.P., Thiongane, Y., Bouloy, M.: Genetic reassortment of Rift Valley fever virus in nature. J. Virol. 73(10), 8196–8200 (1999)
21. Smith, G.J.D., Vijaykrishna, D., Bahl, J., Lycett, S.J., Worobey, M., Pybus, O.G., Ma, S.K., Cheung, C.L., Raghwani, J., Bhatt, S., Peiris, J.S.M., Guan, Y., Rambaut, A.: Origins and evolutionary genomics of the 2009 swine-origin H1N1 influenza A epidemic. Nature 459(7250), 1122–1125 (2009)
22. Suyama, M., Torrents, D., Bork, P.: PAL2NAL: robust conversion of protein sequence alignments into the corresponding codon alignments. Nucleic Acids Res. 34(Web Server issue), W609–W612 (2006)
23. Waterhouse, A.M., Procter, J.B., Martin, D.M.A., Clamp, M., Barton, G.J.: Jalview version 2: a multiple sequence alignment editor and analysis workbench. Bioinformatics 25(9), 1189–1191 (2009)
24. Wise, H.M., Foeglein, A., Sun, J., Dalton, R.M., Patel, S., Howard, W., Anderson, E.C., Barclay, W.S., Digard, P.: A complicated message: Identification of a novel PB1-related protein translated from influenza A virus segment 2 mRNA. J. Virol. 83(16), 8021–8031 (2009)
25. Wiuf, C., Hein, J.: Recombination as a point process along sequences. Theor. Popul. Biol. 55(3), 248–259 (1999)
26. Yang, Z.: Computational molecular evolution. Oxford University Press, Oxford (2006)
27. Yohai, V.J.: High breakdown-point and high efficiency robust estimates for regression. The Annals of Statistics 15(2), 642–656 (1987)

Novel Definition and Algorithm for Chaining Fragments with Proportional Overlaps

Raluca Uricaru, Alban Mancheron, and Eric Rivals

LIRMM, CNRS and Université de Montpellier 2, Montpellier, France
{uricaru,mancheron,rivals}@lirmm.fr

Abstract. Chaining fragments is a crucial step in genome alignment. Existing chaining algorithms compute a maximum weighted chain with no overlaps allowed between adjacent fragments. In practice, using local alignments as fragments, instead of MEMs, generates frequent overlaps between fragments, due to combinatorial reasons and biological factors, *i.e.* variable tandem repeat structures that differ in number of copies between genomic sequences. In this paper, in order to raise this limitation, we formulate a novel definition of a chain, allowing overlaps proportional to the fragments lengths, and exhibit an efficient algorithm for computing such a maximum weighted chain. We tested our algorithm on a dataset composed of 694 genome couples and accounted for significant improvements in terms of coverage, while keeping the running times below reasonable limits.

1 Introduction

In biology, genome comparison is used for gene annotation, phylogenetic studies, and even vaccine design [12,2,8]. Many bioinformatics programs for whole genome comparison involve a fragment chaining step, which seeks to maximize the total length of the chained fragments (eg, [7]). Given the set of n shared genomic intervals, *i.e.* fragments, the Maximum Weighted Chain problem (MWC) is solved in $O(n \log n)$ time by dynamic programming when overlaps between adjacent fragments are forbidden [10,1]. Alternatively, Felsner *et al.* showed that this problem is a special case of the Maximum Weighted Independent Set problem in a trapezoid graph, which they solve by a sweep line algorithm over an equivalent box order representation of the graph [6]. These algorithms [1,6] can be extended to handle fixed length overlap between adjacent fragments, but this is not sufficient to deal with the large differences in fragment length obtained even with small bacterial genomes [14]. Moreover, with those definitions the weight does not account for overlaps. An $O(n \log n)$ time algorithm for the MWC with Fixed Length Overlaps problem was designed and used for mapping spliced RNAs on a genome [13], but the fixed bound on overlaps remains a limitation. To raise this limitation, we formulate the MWC with Proportional Length Overlaps problem (MWC-PLO) and exhibit the first chaining algorithms allowing for overlaps that are proportional to the fragment lengths, and whose chain weight function accounts for overlap. Following Felsner *et al.*, we use the

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 161–172, 2010.

box representation of a trapezoid graph and adapt the sweep line paradigm to this problem.

Small overlaps are often caused by equality over a few base pairs of fragment ends due to randomness, since the alphabet has only four letters. To handle such cases, one could set a constant, large enough, maximal allowed overlap threshold. However, biological structures like tandem repeats (TR) that vary in number of copy units generate overlaps that are large relatively to the fragments involved. To illustrate this case, let u, v, w be words and assume the sequences of two genomes G_a, G_b are $G_a = uvvw$ and $G_b = uvvvw$, *i.e.* contain a variable TR of motif v. Then, uvv generates a local alignment between G_a and G_b, as well as vvw, but both fragments overlap over v in both G_a and G_b. Since v can be large, such cases cannot be circumvented with fixed length overlaps: only proportional overlaps can handle these.

The paper is organised as follows. Section 2 presents the chaining problem without overlaps, while Section 3 defines chaining with proportional overlaps and sets the dynamic programming framework and algorithm that solves it. In Section 4 we exhibit a sweep line algorithm for this question, prove its correctness and discuss the complexities. We study its performance in Section 5 and conclude in Section 6.

2 Preliminaries

Boxes are axis parallel hyper-rectangles in $\mathbb{R}^k$, where each genome is associated with one axis. For simplicity, we consider the two dimensional case where $k = 2$, *i.e.* comparing two genomes. The length on a genome of the fragment associated with a box is the projection of that box on the corresponding axis.

Let $\alpha \in \{1, 2\}$ index the axis, and for any point $x \in \mathbb{R}^2$ let $P_\alpha(x)$ denote its projection on axis α. Let I be an interval of $\mathbb{R}$ and $\mathcal{I}$ be a set of disjoint intervals of $\mathbb{R}$; we denote by $|I|$ the length of I and by $|\mathcal{I}|$ the sum of the lengths of intervals in $\mathcal{I}$. Let B be a box of $\mathbb{R}^2$. The *upper*, resp. *lower*, corner of B is denoted by $u(B)$, resp. $l(B)$. By extension, the interval corresponding to the projection of B on axis α is denoted $P_\alpha(B)$. Let $<$ denote the classical *dominance order* between points of $\mathbb{R}^2$.

Definition 1 (Overlap free box dominance order). *Let B_x, B_y be two boxes of $\mathbb{R}^2$. We say that B_y* dominates *B_x, denoted $B_x \ll B_y$, if $l(B_y)$ dominates $u(B_x)$ in $\mathbb{R}^2$. If neither B_x dominates B_y, nor B_y dominates B_x, then B_x and B_y are* incomparable.

Felsner *et al.* showed how to transform a trapezoid graph into a *box order*, *i.e.* a set of boxes equipped with the dominance order $\ll$ such that pairs of incomparable boxes are in one-to-one correspondance with trapezoid pairs linked by edges of the graph. Hence, the Maximum Weighted Independent Set problem in a trapezoid graph is equivalent to the MWC problem in the corresponding box order [6]. Given an order, recall that a *chain* is a set of mutually comparable elements, and a *maximal element* in a set is one with no other element dominating it. Each chain has exactly one maximal element.

3 A Novel Tolerance Definition for the Maximum Weighted Chain Problem in a Box Order

To formulate a MWC with Proportional Length Overlaps problem (MWC-PLO) in our framework, we need to redefine the dominance order to accept overlaps that are proportional to the boxes' projection lengths, and to propose a weight function that truly measures the coverage on each genome. By *coverage*, it is generally meant the total length of the genomic intervals covered by the selected fragments [9]. This requires that the chain weight counts only once a subinterval covered by several overlapping fragments.

Let $r \in [0, 1[$ represent the maximal allowed overlap ratio between any two boxes.

Definition 2 (r tolerant dominance order). *Let B_u and B_v be two boxes. B_v* dominates *B_u* on axis α *in this* tolerant dominance order*, denoted by $B_u \ll_{r,\alpha} B_v$, if and only if*

$$P_\alpha(u(B_u)) - P_\alpha(l(B_v)) \leq r \min(|P_\alpha(B_u)|, |P_\alpha(B_v)|).$$

Now, we denote by $B_u \ll_r B_v$ the fact that B_v dominates *B_u if and only if for each $\alpha \in \{1, 2\}$ $B_u \ll_{r,\alpha} B_v$.*

It can be easily shown that the dominance between boxes implies the dominance between their upper, resp. lower, corners. Moreover, this tolerant dominance order is transitive.

Property 1. Let B_t, B_u two boxes such that $B_t \ll_r B_u$. Then $l(B_t) < l(B_u)$ and $u(B_t) < u(B_u)$.

Property 2. The dominance order $\ll_r$ is transitive.

From Property 1, one deduces the following corollary, which will help to compute efficiently the weight of overlapping boxes in a chain.

Corollary 1. *Let B_t, B_u, B_v be three boxes such that $B_t \ll_r B_u \ll_r B_v$. Then: $(B_t \cap B_v) \subset (B_u \cap B_v)$.*

We define the *weight of a box* as the sum of lengths of its projections on all axis, and the *weight of a chain* of boxes as the sum of the coverages on each axis.

Definition 3 (Weight of a box, of a chain). *Let B be a box and $\alpha \in [1, 2]$. Its weight on axis α is $w_\alpha(B) := |P_\alpha(B)|$, and its weight is $w(B) := \sum_{\alpha=1}^{2} w_\alpha(B)$. Let $m \in \mathbb{N}$ and $C := (B_1 \ll_r \ldots \ll_r B_m)$ be a chain of m boxes. The weight of C on axis α, denoted $W_\alpha(C)$, is*

$$W_\alpha(C) := \left| \bigcup_{i=1}^{m} P_\alpha(B_i) \right|,$$

while its weight *is $W(C) := \sum_{\alpha=1}^{2} W_\alpha(C)$.*

Note also that the weight of a box only depends on the endpoints of its projection on each axis, and hence, can be computed in constant time. Clearly, it can be easily seen that

$$W_\alpha(C) = w_\alpha(B_m) + \sum_{j=1}^{m-1} \left| P_\alpha(B_j) \setminus \bigcup_{l=j+1}^{m} P_\alpha(B_l) \right|$$
$$= w_\alpha(B_m) + \sum_{j=1}^{m-1} |P_\alpha(B_j) \setminus P_\alpha(B_{j+1})| \quad \text{by Corollary 1.} \qquad (1)$$

The following easy property will also prove useful.

Property 3. Let B_t, B_u two boxes such that $B_t \ll_r B_u$. Then

- $B_t \cap B_u$ is an, eventually empty, axis parallel rectangle of $\mathbb{R}^2$, and
- for $\alpha \in [1,2]$, $|P_\alpha(B_t) \setminus P_\alpha(B_u)| = |P_\alpha(B_t) \setminus P_\alpha(B_t \cap B_u)| = w_\alpha(B_t) - w_\alpha(B_t \cap B_u)$.

Now, we can define the MWC-PLO problem. Let $\mathcal{B}' := \{B_2, \ldots B_{n-1}\}$ be the set of input boxes. For convenience, we add two dummy boxes, B_1, B_n, such that for all $1 < i < n$: $B_1 \ll_r B_i \ll_r B_n$. Additionally, we set $w(B_1) = w(B_n) := 0$. Now, the input consists in $\mathcal{B} := \{B_1, \ldots, B_n\}$.

Definition 4 (MWC with Proportional Length Overlaps)**.** *Let $r \in [0,1[$ and $\mathcal{B} := \{B_1, \ldots, B_n\}$ a set of boxes. The* MWC with Proportional Length Overlaps *problem is to find in $\mathcal{B}$, according to the dominance order $\ll_r$, the* chain C *that starts with B_1 and ends in B_n and whose weight $W(C)$ is maximal.*

The notation of r, $\mathcal{B}$, and $W(C)$ are valid throughout the paper. For any $1 \leq i \leq n$, let us denote by $\mathcal{C}_i$ the set of chains ending in B_i, and by $W(B_i)$ the weight of the maximal weighted chain in $\mathcal{C}_i$ (not to be confounded with $w(B_i)$). From now on, all the considered boxes belong to $\mathcal{B}$ unless otherwise specified.

3.1 A Dynamic Programming Framework

Let us show that MWC-PLO can be solved by a dynamic programming algorithm. Equation 1 suggests a recurrence equation to compute $W(B_i)$, with $W(B_1) = 0$ and for all $1 < i \leq n$:

$$W(B_i) = \max_{B_j:\ B_j \ll_r B_i} W(B_j) + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B_j)| \ . \qquad (2)$$

Obviously, this implies that for all $1 \leq j < n$ the value of $W(B_j)$ will be reused for computing $W(B_i)$ for every box B_i such that $B_j \ll_r B_i$. Thus, MWC-PLO consists of *overlapping subproblems*, which suits to the framework of dynamic programming [4, chap. 15]. However, it is correct to use Equation 2 only if our problem satisfies the condition of *optimal substructures* [4, chap. 15]. In Theorem 1, we show this is true.

Theorem 1 (Optimality of substructures). *Let $m, i_1, \ldots, i_m$ be integers belonging to $[1, n]$, and let $D := (B_{i_1}, \ldots, B_{i_m})$ be an optimal weighted chain among the chains in $\mathcal{C}_{i_m}$. Thus, $D' := (B_{i_1}, \ldots, B_{i_{m-1}})$ is an optimal weighted chain among those in $\mathcal{C}_{i_{m-1}}$.*

The MWC with Proportional Length Overlaps can thus be solved by a dynamic programming algorithm, which uses two n-element arrays: $W[.]$ and Pred[.] to store for all $1 \leq i \leq n$ resp. the values of $W(B_i)$ and the predecessor of B_i in an optimal weighted chain ending in B_i. This algorithm takes $O(n^2)$ time and $O(n)$ space; in Section 4 we prove a more efficient algorithm for MWC-PLO.

Theorem 2. *A dynamic programming algorithm (Algorithm DP) solves the* MWC with Proportional Length Overlaps *problem in $O(n^2)$ time and $O(n)$ space.*

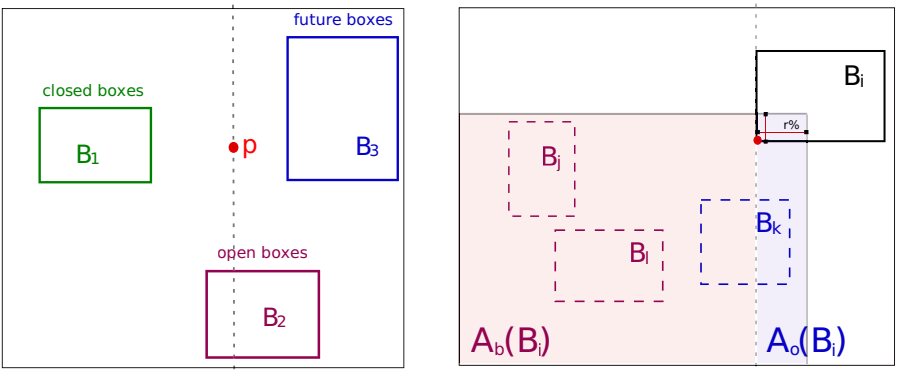
4 A Sweep Line Algorithm for MWC with Proportional Length Overlaps

Here, we exhibit a sweep line algorithm for the MWC with Proportional Length Overlaps problem (see Algorithm 1), prove it and study its complexity.

4.1 Outline of the Algorithm

Following Felsner *et al.*, we give a sweep line algorithm in which a vertical line sweeps the boxes in the plane by increasing x-coordinates of their corners, stopping at the lower left and upper right corners of each box. To avoid visiting, as in Algorithm DP, all possible predecessors when computing the best chain ending in B_x, we maintain a set, $\mathcal{A}$, of *active* boxes that can compete for being the optimal predecessor in that chain. But as predecessors can overlap B_x, this computation involves several steps, meaning that $W[B_x]$ and Pred$[B_x]$ can be updated several times before getting their final value; this differs from Algorithm DP.

Let $\mathcal{P}$ be an array containing the $2n$ points corresponding to $l()$ and $u()$ corners of the n boxes in $\mathcal{B}$. Points in $\mathcal{P}$ are ordered on their x-coordinates; among the points having identical x-coordinates, lower corners are placed before upper corners. For each point, we store to which box and to which corner it corresponds to. In Algorithm 1, the main loop sweeps the points of $\mathcal{P}$ and processes in a different manner lower (lines 8-11) and upper corners (lines 12-24). We say a box B_x is *open* when the sweep line is located between $l(B_x)$ and $u(B_x)$ inclusive, *closed* when the line has passed $u(B_x)$, and *future* when it lies before $l(B_x)$. These states are exclusive of each other, and partition at each moment $\mathcal{B}$ in three disjoint sets (see Figure 1a). All open boxes at each point are kept in a set $\mathcal{O}$ (lines 9, 13). The weight of a chain ending in, say B_i, and passing by a predecessor of B_i, B_x, can only be computed when B_x is closed (when $W[B_x]$ has reached its final value). If $P_1(u(B_x)) < P_1(l(B_i))$ then this can be done when stopping at $l(B_i)$ (lines 10-11), while if B_x overlaps B_i on x-axis, then this is done when stopping at $u(B_x)$, and at the same time for all open boxes having B_x as predecessor (lines 14-18). These two cases partition the possible predecessors



(a) (b)

Fig. 1. (a) Example of boxes in each disjoint set forming a partition of $\mathcal{B}$, when sweeping a point p. (b) Partition of the search space of possible predecessors of B_i, according to the location of their upper corners, in two areas $A_b(B_i)$ and $A_o(B_i)$. $A_b(B_i)$ and $A_o(B_i)$ partition the rectangle delimited by a solid line: $A_b(B_i)$ is at left from the dashed line, and $A_o(B_i)$ at its right.

of B_i according to the location of their upper corners in two areas $A_b(B_i)$ and $A_o(B_i)$ (cf. Figure 1b).

As above mentioned, we maintain in $\mathcal{A}$ the set of interesting predecessors for all future boxes. Boxes in $\mathcal{A}$ are *active* boxes. Hence, once closing a box (stopping at its upper corner), we test whether it should be turned active and inserted in $\mathcal{A}$ (lines 19-21). The current box, B_i, is inserted only if we cannot find a better predecessor in $\mathcal{A}$. Afterwards, if B_i has been added, currently active boxes are investigated to determine if they are less interesting than B_i, in which case they are deleted from $\mathcal{A}$ (lines 22-24). Active boxes are consulted when opening a box B_i, for computing the best chain ending in B_i with a predecessor in $A_b(B_i)$ (lines 10-11).

4.2 Correctness of the Algorithm

For $1 \leq i \leq n$, we show that $W[B_i]$ and $\text{Pred}[B_i]$ store the weight and the predecessor of B_i in a maximum weighted chain ending in B_i. First, several simple invariants emerge from Algorithm 1. I_1: At any point, the set $\mathcal{O}$ contains all open boxes. I_2: Both $W[B_i]$ and $\text{Pred}[B_i]$ store their final values once $u(B_i)$ has been processed, since they are not altered after that point. I_3: Hence, at any point all active boxes (*i.e.* boxes in $\mathcal{A}$), which are closed boxes, satisfy I_2. For conciseness, as $W[B_i]$ and $\text{Pred}[B_i]$ are computed jointly, from now on we deal only with $W[B_i]$. Since potential predecessors of B_i are partitioned in $A_b(B_i)$ (Figure 2a) and $A_o(B_i)$ (Figure 2b), we will prove two invariants: I_4: partial optimality over $A_b(B_i)$ at lower corners, and I_5: optimality at upper corners.

I_4: partial optimality over $A_b(B_i)$ at lower corners. We show that after processing $l(B_i)$, $W[B_i]$ stores the weight of a maximum weighted chain ending in B_i

Algorithm 1. MWC_Tolerance_Box_Order ($\mathcal{P}$)

Data: $r \in [0,1[$, $\mathcal{B}$ a set of n boxes, $\mathcal{P}$ an array with the $2n$ box corners
Result: W a vector of weights, with $W[B_n]$ the weight of the best chain in $\mathcal{B}$, Pred a vector containing the previous boxes in the chain

```
1  begin
2      sort_on_x_coordinate(P);
3      A ⟵ B_1;
4      W[B_1] ⟵ 0;
5      Pred[B_1] ⟵ null;
6      O ⟵ ∅;
7      foreach p ∈ P in ascending order on x-coordinate do
8          if p is a lower corner (i.e. ∃B_i : p = l(B_i)) then
9              O ⟵ O ∪ {B_i};
               Pred[B_i] ⟵ argmax_{B_j ≪_r B_i, B_j ∈ A} (W[B_j] + Σ_{α=1}^{2} |P_α(B_i) \ P_α(B_j)|);
10
               W[B_i] ⟵ W[Pred[B_i]] + Σ_{α=1}^{2} |P_α(B_i) \ P_α(Pred[B_i])|;
11
12         else                    /* p is an upper corner, i.e. ∃B_i : p = u(B_i) */
13             O ⟵ O \ {B_i};
14             foreach B_k ∈ O with B_i ≪_r B_k do
                   w_k ⟵ W[B_i] + Σ_{α=1}^{2} |P_α(B_k) \ P_α(B_i)|;
15
16                 if w_k > W[B_k] then
17                     W[B_k] ⟵ w_k;
18                     Pred[B_k] ⟵ B_i;
               B ⟵ argmax_{u(B_j) < u(B_i), B_j ∈ A} (W[B_j]);
19
20             if W[B_i] ≥ W[B] or |P_2(B_i)| > |P_2(B)| then
21                 A ⟵ A ∪ {B_i};
22                 foreach B_k ∈ A with P_2(u(B_k)) > P_2(u(B_i)) do
23                     if W[B_k] < W[B_i] and (|P_2(B_k)| < |P_2(B_i)| or
                       P_2(l(B_k)) > P_2(u(B_i))) then
24                         A ⟵ A \ {B_k};
25     traceback(Pred[B_n]);
26 end
```

with predecessor in $A_b(B_i)$. Given line 10, this is equivalent to showing that no better chain ending in B_i passes through a potential predecessor that does not belong to $\mathcal{A}$ at that point, which we prove by contradiction. While processing $l(B_i)$, $\mathcal{A}$ contains a subset of boxes in $A_b(B_i)$, but obviously none from $A_o(B_i)$. Let B be a closed box of $\mathcal{B} \setminus \mathcal{A}$ such that $B \ll_r B_i$ and $w(B_i) - w(B \cap B_i) + W[B] > W[B_i]$, in other words, B makes a better predecessor for B_i than those in $\mathcal{A}$. From $B \ll_r B_i$, we get

$$P_2(u(B)) - P_2(l(B_i)) \leq r \min(|P_2(B)|, |P_2(B_i)|) \ . \tag{3}$$

As only two possibilities exist for B not belonging to $\mathcal{A}$, we distinguish two exclusive cases.

B was not turned active when sweeping $u(B)$ (lines 19-21). B did not satisfy the condition on line 20. Let $B' := \underset{B_j \in \mathcal{A}: u(B_j) < u(B)}{\arg\max} (W[B_j])$. Our hypothesis means that $u(B') < u(B)$ and

$$W[B] < W[B'] \tag{4}$$

$$|P_2(B)| \leq |P_2(B')| \,. \tag{5}$$

For B does not overlap B_i and $u(B') < u(B)$, we have B' does not overlap B_i on the x-axis. From $u(B') < u(B)$, we get $P_2(u(B')) < P_2(u(B))$; this with equations 3 and 5 yields

$$\begin{aligned} P_2(u(B')) - P_2(l(B_i)) &< P_2(u(B)) - P_2(l(B_i)) \\ &\leq r \min(|P_2(B)|\,, |P_2(B_i)|) \\ &\leq r \min(|P_2(B')|\,, |P_2(B_i)|)\,. \end{aligned} \tag{6}$$

Equation 6 and B' not overlapping B_i on the x-axis imply $B' \ll_r B_i$. Finally, from equations 4, 5, and $u(B') < u(B)$ we obtain:

$$W[B] + \sum_{\alpha=1}^{2} (w_\alpha(B_i) - w_\alpha(B_i \cap B)) < W[B'] + \sum_{\alpha=1}^{2} (w_\alpha(B_i) - w_\alpha(B_i \cap B')),$$

and thus B' makes a better predecessor for B_i than B, a contradiction.

B was inactivated when sweeping $u(B_k)$ for some box B_k ending before $l(B_i)$ (lines 22-24). The hypothesis means that B was deleted from $\mathcal{A}$ for it satisfied $P_2(u(B_k)) < P_2(u(B))$, $W[B] < W[B_k]$, and at least one of the conditions (a) $|P_2(B)| < |P_2(B_k)|$ or (b) $P_2(u(B_k)) < P_2(l(B))$.

a) As above (see Eq. 6), from Equation 4, from $|P_2(B)| < |P_2(B_k)|$, and $P_2(u(B_k)) < P_2(u(B))$, we get

$$P_2(u(B_k)) - P_2(l(B_i)) < r \min(|P_2(B_k)|\,, |P_2(B_i)|)\,. \tag{7}$$

Moreover, as B_k does not overlap B_i on the x-axis, we obtain $B_k \ll_r B_i$. As $P_2(u(B_k)) < P_2(u(B))$, B_k and B do not overlap B_i on x-axis, and $W[B] < W[B_k]$, we finally derive

$$W[B] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B)|) < W[B_k] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B_k)|)\,. \tag{8}$$

b) By hypothesis, we know that $P_2(u(B_k)) < P_2(l(B)) < P_2(l(B_i))$, and since neither B_k nor B overlap B_i on the x-axis, we directly obtain $B_k \ll B_i$ ($B_i \cap B_k = \emptyset$). Thus, $W[B] < W[B_k]$ also implies Equation 8.

With either condition (a) or (b), B_k makes a better predecessor for B_i than B, a contradiction.

Finally, after processing $l(B_i)$, $W[B_i]$ stores the weight of a maximum weighted chain ending in B_i with predecessor in $A_b(B_i)$, which concludes the proof of I_4.

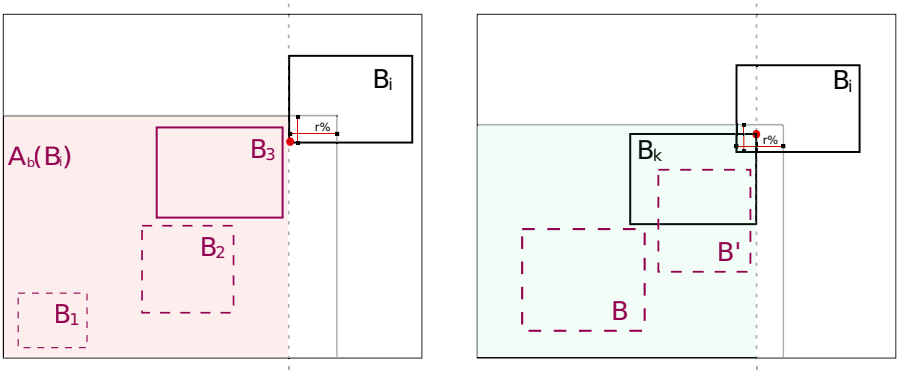


Fig. 2. (a) When the sweep line passes $l(B_i)$, Pred$[B_i]$ is a partial optimum on the set of possible predecessors of B_i lying in $A_b(B_i)$. In the example, B_3 is the best current predecessor of B_i. (b) When the sweep line passes $u(B_k)$, Pred$[B_i]$ is a partial optimum on the set of possible predecessors of B_i from $A_b(B_i) \cup \{B \in A_o(B_i)/P_1(u(B)) < P_1(u(B_k))\}$.

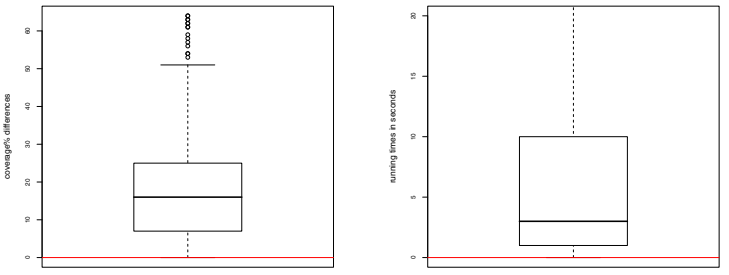
I_5*: optimality at upper corners.* We show that after processing $u(B_i)$, $W[B_i]$ stores $W(B_i)$ (a Maximum Weighted Chain with a predecessor in $A_b(B_i) \cup A_o(B_i)$). As all predecessors of B_i are closed, let us denote by B, the right most predecessor of B_i on the x-axis: $B := \arg\max_{B_j \ll_r B_i}(P_1(u(B_j)))$.

1. If $u(B) \in A_b(B_i)$ then all predecessors of B_i are contained in $A_b(B_i)$. Hence, this situation was handled when processing $l(B_i)$, and Invariant I_4 regarding the *partial optimality at lower corners*, ensures that $W[B_i]$ stores $W(B_i)$.
2. If $u(B) \in A_o(B_i)$, $W[B_i]$ has been correctly updated (lines 14-18), while B_i was open, when sweeping $u(B_k)$ for each box $B_k \in \mathcal{B}$ such that $B_k \ll_r B_i$ and $u(B_k) \in A_o(B_i)$.

Hence, all predecessors of B_i have been taken into account, and $W[B_i]$ stores $W(B_i)$. This concludes the proof of I_5, and closes the correctness proof.

4.3 Time and Space Analysis

Obviously, the sets $\mathcal{O}$ and $\mathcal{A}$ contain at most n boxes, and thus require together with arrays Pred[.] and W[.], $O(n)$ space. We use balanced binary search trees (BST) to store $\mathcal{A}$ and $\mathcal{O}$, with boxes at the leaves ordered on $P_2(u(.))$, resp. $P_1(l(.))$. Hence, the amortized time needed for all insertions, deletions, and rebalancing is $O(n \log n)$. However, looking for the active boxes that can be deleted at each execution of the outer loop (lines 22-24) may force us to examine all boxes in $\mathcal{A}$. As this is the more complex operation in the outer loop, we obtain an $O(n^2)$ worst case time complexity. Algorithm 1 maintains the subset of potential predecessors in $\mathcal{A}$ instead of searching through the whole box set as in Algorithm DP. The experimental running times observed when performing 694 whole genome comparisons show that this difference yields substantial



(a) (b)

Fig. 3. (a) Differences in coverage obtained on bacterial genome comparisons between our algorithm and the classical chaining. The difference expresses which percentage of the genomes are additionally covered when allowing for overlaps. The results presented in a box-and-whiskers plot show the improvement in coverage brought by the acceptance of overlaps in the chain: in 50% of the cases it covers 15% more of the genomes. (b) Running times in seconds of our algorithm: in 75% of the cases the algorithm needs less than 10 seconds.

improvements: Algorithm 1 takes seconds, sometimes minutes, where Algorithm DP, which is truly quadratic, takes minutes, hours, or even days, for values of n ranging from 71 to $1,000,000$ fragments.

5 Results

An issue is whether allowing for overlaps improves the chain weight (here, the genome coverage) when comparing genomes, and at which computational cost. To investigate this issue, we compared the running times and coverages obtained without (using Chainer [1]) and with proportional overlaps (using Algorithm 1) on 694 pairwise genome comparisons. Our comparison set consists in all pairwise genome comparisons of strains of the same bacteria (intra-species comparisons) as of Jan 2010: it comprises 346 different genomes from 87 species retrieved from Genome Reviews database [5]. First, we searched for local alignments between genome pairs using YASS with default parameters [11]. The output local alignments are the fragments given as input to the chaining step, for which we ran in parallel Chainer and Algorithm 1, with the weight of a fragment on each genome being the length of the aligned sequence. We authorized overlaps measuring up to 10% of the fragments' lengths ($r = 0.1$). Hence, the total chain weight, *i.e.* the sum of the chained fragments lengths minus the overlaps, computed by the chaining step gives the *genome coverage*, which we report as a percentage of the genome length. Of course as both chaining algorithms provide an optimal solution in their setup, the coverage with overlaps is larger than without overlaps.

Figure 3a plots the difference of coverages between both algorithms (*e.g.*, a value of 10 means that chaining with overlaps covers 10% of the genome more

than without overlaps). The box-and-whiskers plot shows that the improvement has a median of 15% and reaches values up to 60%. Since bacterial genomes have a median length of 2.8 Mbp, a difference of one percent means at least 28 Kbp of additionally aligned sequences. Knowing that in average 87% of these genomes are coding and bacterial genes are 1 Kbp long [15], 15% more coverage will involve 365 additional genes compared to a solution without overlaps; this is important from the biological perspective. Chainer takes < 1 s. in average and at most 17 s. We plot the running times of Algorithm 1 in Figure 3b: below 10 seconds in 75% of the comparisons, and between 3 and 54 minutes in only 30 cases (those with > 1 million fragments). Thus, allowing for overlaps improves the coverage significantly at a reasonable cost.

A biological question regards the causes of such overlaps. For example, when comparing strains $CP000046$ and $BA000018$ of *S. aureus* the classical chaining results in a coverage of 65%, where Algorithm 1 yields a 94% coverage. In fact, the chain obtained without allowing overlaps is interrupted by 17 holes of more than 10 kbp each. For 14 of these holes, at least one large fragment (average size 37 kbp) was not included in the chain, because of an overlap with an adjacent fragment on one or both genomes. All overlaps measure between 1 bp and 1.8 kbp in length (average at 218 bp). This example shows that overlaps' lengths cannot be easily bounded by a constant. Large overlaps are due to variable tandem repeat structures that differ in number of copies between the strains. Correctly aligning such structures without breaking the region in two overlapping fragments requires a more general alignment model and specific algorithms [3].

6 Conclusion

To fulfil new needs in computational biology, we extended the classical framework of Maximum Weighted Chain by authorizing overlaps between fragments in the computed chain, and formalized the Maximum Weighted Chain with Proportional Length Overlaps problem where overlaps are proportional to the fragment lengths. Difficulties arise from the fact that the weights of overlaps are deduced from the chain weight. We exhibited the first two algorithms for this problem, which both solve it in quadratic time in function of the number of fragments. Experiments on real data sets show that i/ overlaps improve significantly the coverage of genomes (median of 15%), ii/ our sweep line algorithm outperforms the truly quadratic dynamic programming solution in practice. However, the study of the average time complexity of the sweep line algorithm remains open. Comparing with fixed overlaps, as well investigating the robustness with respect to ratio of allowed overlaps are future lines of research.

Acknowledgements. RU beneficiates from a PhD fellowship from the French Ministery of Research. This work is supported by the ANR project CoCoGen (ANR-07-BLAN-0365).

References

1. Abouelhoda, M.I., Ohlebusch, E.: Chaining algorithms for multiple genome comparison. J. of Discrete Algorithms 3, 321–341 (2005)
2. Boussau, B., Daubin, V.: Genomes as documents of evolutionary history. Trends in Ecology & Evolution 25(4), 224–232 (2010)
3. Bérard, S., Rivals, E.: Comparison of minisatellites. J. Comp. Biol. 10(3-4), 357–372 (2003)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
5. Kersey, P., et al.: Integr8 and Genome Reviews: integrated views of complete genomes and proteomes. Nucleic Acids Res. 33(S1), D297–D302 (2005)
6. Felsner, S., Muller, R., Wernisch, L.: Trapezoid graphs and generalizations, geometry and algorithms. Disc. App. Math. 74, 13–32 (1995)
7. Hohl, M., Kurtz, S., Ohlebusch, E.: Efficient multiple genome alignment. Bioinformatics 18(S1), S312–S320 (2002)
8. Jackson, A.P., Gamble, J.A., Yeomans, T., Moran, G.P., Saunders, D., Harris, D., Aslett, M., Barrell, J.F., Butler, G., Citiulo, F., Coleman, D.C., de Groot, P.W.J., Goodwin, T.J., Quail, M.A., McQuillan, J., Munro, C.A., Pain, A., Poulter, R.T., Rajandream, M.-A., Renauld, H., Spiering, M.J., Tivey, A., Gow, N.A.R., Barrell, B., Sullivan, D.J., Berriman, M.: Comparative genomics of the fungal pathogens Candida dubliniensis and Candida albicans. Genome Res. 19(12), 2231–2244 (2009)
9. Lemaitre, C., Sagot, M.-F.: A small trip in the untranquil world of genomes. Theor. Comp. Sci. 395, 171–192 (2008)
10. Myers, G., Miller, W.: Chaining multiple-alignment fragments in sub-quadratic time. In: Proc. of the sixth annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 38–47 (1995)
11. Noé, L., Kucherov, G.: YASS: enhancing the sensitivity of DNA similarity search. Nucleic Acids Res. 33(S2), W540–W543 (2005)
12. Serruto, D., Rappuoli, R.: Post-genomic vaccine development. FEBS Letters 580(12), 2985–2992 (2006)
13. Shibuya, T., Kurochkin, I.: Match chaining algorithms for cDNA mapping. In: Benson, G., Page, R.D.M. (eds.) WABI 2003. LNCS (LNBI), vol. 2812, pp. 462–475. Springer, Heidelberg (2003)
14. Uricaru, R., Michotey, C., Noé, L., Chiapello, H., Rivals, E.: Improved sensitivity and reliability of anchor based genome alignment. In: Rusu et Eric Rivals, I. (ed.) Actes des Journées Ouvertes Biologie Informatique Mathématiques (JOBIM), pp. 31–36 (2009)
15. Xu, L., Chen, H., Hu, X., Zhang, R., Zhang, Z., Luo, Z.W.: Average Gene Length Is Highly Conserved in Prokaryotes and Eukaryotes and Diverges Only Between the Two Kingdoms. Mol. Biol. Evol. 23(6), 1107–1108 (2006)

Assessing the Robustness of Complete Bacterial Genome Segmentations

Hugo Devillers[1], Hélène Chiapello[1], Sophie Schbath[1], and Meriem El Karoui[2]

[1] INRA, UR1077, Mathématique, Informatique et Génome, Domaine de Vilvert, F-78352, Jouy-en-Josas, France
[2] INRA, UR888, Unité Bactéries Lactiques et Pathogènes Opportunistes, Domaine de Vilvert, F-78352, Jouy-en-Josas, France
hugo.devillers@jouy.inra.fr

Abstract. Comparison of closely related bacterial genomes has revealed the presence of highly conserved sequences forming a "backbone" that is interrupted by numerous, less conserved, DNA fragments. Segmentation of bacterial genomes into backbone and variable regions is particularly useful to investigate bacterial genome evolution. Several software tools have been designed to compare complete bacterial chromosomes and a few online databases store pre-computed genome comparisons. However, very few statistical methods are available to evaluate the reliability of these software tools and to compare the results obtained with them. To fill this gap, we have developed two local scores to measure the robustness of bacterial genome segmentations. Our method uses a simulation procedure based on random perturbations of the compared genomes. The scores presented in this paper are simple to implement and our results show that they allow to discriminate easily between robust and non-robust bacterial genome segmentations when using aligners such as MAUVE and MGA.

1 Introduction

The number of complete bacterial genomes in public databases has exponentially increased since the late 90s. The availability of this large amount of data has yielded the development of new comparative-based approaches to investigate bacterial genome evolution at different scales (*e.g.*, genomes, operons, genes). Genome comparison has proved its usefulness in many domains such as gene prediction [1], detection of regulatory regions and functional motifs [2], and assembly of new genomes [3].

Among the about 500 bacterial species for which a complete genome is available, more than 100 present at least two sequenced strains. Several approaches exist to compare closely related genomes. Among them, alignment of whole genome DNA sequences, at the nucleotide level, allows both coding and non-coding regions to be investigated. In a pioneering work, Hayashi *et al.* [4] compared the two complete genomes of the enterohemorrhagic *Escherichia coli* O157:H7 Sakai strain and the *E. coli* K-12 MG1655 laboratory strain. An extensive investigation of the alignment result revealed a sequence corresponding to

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 173–187, 2010.

around 80% of the genome length that is highly conserved between the two *E. coli* strains. This sequence, denoted the backbone, is interrupted by several DNA fragments that are not conserved in the two strains and that are called variable segments. The backbone/variable segment structure is named segmentation or mosaic structure of bacterial genomes [4,5].

Determination and analysis of the segmentation are of crucial interest to investigate the molecular mechanisms involved in the dynamics of bacterial genome evolution. Indeed, it has been shown that segments from the backbone are enriched in functional DNA motifs and consequently are particularly relevant to infer them [2]. Moreover, sequences from the backbone may correspond, in a large part, to the common ancestral chromosome of the compared strains [6]. Variable segments that may correspond to strain specificities are often associated with DNA exchange and mobile elements such as transposons or prophages [7] and can be used to investigate biological questions about bacterial physiology and pathogenicity [8]. Nature and origin of variable segments are diverse. They can be either strain specific or just too divergent to be identified as homologous. For a detailed description of variable segment diversity, see [9]. The determination of the backbone/variable segment structure has direct implications on all subsequent analyses, and thus needs to be made with accuracy.

Segmentation computation is essentially based on alignment of complete genome sequences [5]. Aligning sequences containing several million of nucleotides is a challenging task that requires specific algorithms. Moreover, besides the length of the sequences, the main difficulty relies on the fact that bacterial genomes evolve through various genetic mechanisms including point mutations, genetic rearrangements and horizontal gene transfers, which generate extremely dynamic genomes, even in closely related bacteria.

In the early 2000s, Miller pointed out the challenges and the difficulties to design specific algorithms to align complete genome sequences as well as the necessity to develop statistical methods to evaluate their reliability [10]. The software tool MUMmer [11,12], launched in 1999, was probably the first published complete genome aligner. To date, there exist about 20 different computer tools to compare and align genomes [13]. Most of the efforts so far, have been focused on the design of fast and efficient algorithms, whereas less attention has been paid to statistically assess their consistency [10,14]. Consequently, several major problems still remain in complete genome alignment methods. Thus, for example, most of the algorithms produce a unique solution that is optimal with respect to a specific algorithmic criterion whereas sub-optimal solutions that could be biologically more relevant are systematically ignored. Evidences of spurious alignments have also been reported [15]. In addition, it has been shown that small variations in the setting of algorithm parameters can have dramatic impacts on the alignment results [5]. Such drawbacks can lead to important differences between alignments produced by different algorithms [16] and even by the same aligner [5].

Because complete genome aligners suffer from a lack of reliability and may produce non-robust alignments, it is crucial to evaluate whether the segmentations

derived from them are robust. There only exist a few studies that aim at assessing the quality of genome segmentations. They are usually dedicated to evaluating the relevance of the elements that are assigned to conserved regions. Thus, for example, Prakash and Tompa [15] developed a score to discriminate regions that are well aligned from those that are suspiciously aligned. Swidan and Shamir [17] proposed two measurements to quantitatively assess the biological reliability of the backbone. However, all these approaches overlook the fact that the proposed segmentation could be non-robust.

Here we present the development of a method to measure the robustness of segmentations obtained from the comparison of two closely related genomes. It is based on a simulation procedure that randomly perturbs the compared genomes. It allows the computation of two local scores of robustness, one at the nucleotide scale, and the other one at the segment scale. Our results show that the scores are simple to implement and to interpret. They can be easily used to discriminate between robust and non-robust segmentations.

2 Methods

2.1 Segmentation Determination

Segmentation into backbone/variable segments is computed from the comparison of closely related genomes. The process can be divided into four steps [5].

1) Alignment of the complete genome sequences. Specific algorithms called anchor-based aligners are generally used [5]. They first identify all the highly conserved regions between the compared sequences. These regions are chained, *i.e.* they are sorted and some of them are selected to anchor the compared genomes together. The result is a succession of extremely conserved segments interrupted by more distant fragments called gaps.

2) Iteration of the anchor-based alignment on the gaps to extend the anchoring obtained in step 1. This step is optional.

3) "Last chance" local alignment. Local alignment methods are used on the remaining gaps to extend the complete genome alignment. This step is also optional.

4) Determination of segment boundaries. Anchored fragments and elements that are enough conserved to be aligned are joined together to form the backbone. The remaining gaps are considered as variable segments.

2.2 Simulation Procedure

The robustness of a process corresponds to its capacity to cope well with external or internal perturbations [18]. Therefore, the robustness of a computational method can be defined as its ability to maintain stable results when confronted to perturbations either of its parameters or of the input data.

Segmentation determination is essentially based on complete genome alignment (step 1). Thus, a reliable approach to perturb segmentation computation

is to disturb directly this crucial step. Our assumption is that if the alignment procedure is reliable, it will be relatively insensitive to a moderate level of random perturbations of the aligned genomes. In particular, the modification of a subset of the conserved regions that are used to anchor the genome alignments should not lead to major changes on the whole genome alignments.

Consequently, we decided to perturb these conserved regions. More precisely, to target these conserved regions, we used maximal exact matches (MEMs). MEMs correspond to exactly conserved sub-sequences that cannot be extended neither from their left nor from their right [11].

We developed the following simulation procedure:

1) Retrieval of all the MEMs between the two compared genomes. Only the MEMs with a minimal length of 20 nucleotides, from both the direct and the reverse strand, are considered for perturbation, in order to avoid spurious matches [19].

2) Random sampling of a proportion p of MEMs from the complete MEM list obtained in step 1. The value of p has to be set by the user. Both genomes are evenly perturbed with $p/2$ MEMs drawn from each genome.

3) Perturbation of the nucleotides corresponding to the MEMs selected in step 2. Three types of perturbations are used: i) Deletions, a MEM sequence is simply deleted; ii) Inversions, a MEM sequence is reverse-complemented and reinserted at the same position; iii) Double translocations, two MEM sequences are switched. Note that both genomes are perturbed in equal proportion.

4) Computation of a new segmentation with the perturbed genomes. Because this "perturbed segmentation" will be compared to the "original segmentation", it is necessary to use the same method for the two segmentations (*i.e.*, same complete genome aligner, same parameter setting).

5) Iteration of steps 2 to 4. The procedure is repeated N times to ensure the statistical reliability of the scores (defined in subsection 2.3).

6) Computation of the robustness scores (see details in subsection 2.3).

We used different perturbation strategies (see step 3) because each of them impacts differently on the perturbation step. For example, while deletions only suppress MEM sequences, inversions and translocations can reveal new MEMs. Translocations also change MEM locations in the compared genomes, affecting the chaining step. The three types of perturbations were used in equal proportions.

2.3 Score Definition

Robustness is measured from the evaluation of the differences between the original segmentation and the segmentations computed with the perturbed genomes. Two scores are derived, one focusing on the nucleotides and the other one on the segments.

Nucleotide score. Considering the nucleotide i from one of the original genomes, either from a backbone or a variable segment of the original segmentation, the nucleotide score is defined as follows:

$$S_{\text{nuc}}(i) = \frac{\#\{\text{simulations}|i \in \text{variable segment}\}}{\#\{\text{total simulations}\}}. \quad (1)$$

It is equal to the proportion of simulations in which the nucleotide i is assigned to a variable segment. Thus, S_{nuc} varies between 0 and 1. If $S_{\text{nuc}}(i)$ is near 1 then i is likely to belong to a variable segment. Conversely, if $S_{\text{nuc}}(i)$ is near 0, i is likely to belong to a backbone segment.

Segment score. Considering segment g from the original segmentation (*i.e.*, the non-perturbed segmentation), the segment score is defined by:

$$S_{\text{seg}}(g) = \frac{1}{|g|} \sum_{i \in g} S_{\text{nuc}}(i), \quad (2)$$

where $|g|$ denotes the number of nucleotides in segment g. It is equal to the average of the nucleotide scores of the nucleotides belonging to segment g. If $S_{\text{seg}}(g)$ is close to 1 then the segment g is likely to be a robust variable segment.

2.4 Implementation

The anchor-based tool MUMmer (version 3) [12] was used for retrieving the MEMs from the compared genomes for the perturbation procedure. The complete simulation procedure and the score computations were performed with Perl (Perl 5.8.8). The source code is available upon request. The graphical exploration of our results was mainly performed with the genome-browser MuGeN (version 20060919) [20] and R (version 2.8.0).

2.5 Dataset

Numerous pairs of bacterial genomes retrieved from the MOSAIC database [5] were tested. For illustration purposes, three representative pairs were selected and are presented in Table 1. The two *Streptococcus pyogenes* strains are very close, the *Escherichia coli* strains are moderately close, and the *Pseudomonas syringae* strains are distant. The genomic distance between these genomes was evaluated with the MUMi index [21]. Briefly, it is based on the evaluation of the cumulative size of maximal unique matches (MUMs) compared to the lengths of the compared genomes. MUMi varies between 0 and 1, the smaller the MUMi, the closer the compared genomes (see details in Table 1). Segmentations of these three pairs were computed using two different aligners, MGA (version 2) [22] and MAUVE Aligner (version 1.2.3) [23]. Parameter settings used for these two tools were those proposed by Chiapello *et al.* [5,24]. Table 2 summarizes the information concerning the obtained segmentations for the first strain of each pair.

Table 1. Pairs of genomes compared in this study. Length: total genome length (in nucleotides), MUMi: MUMi index of the pair, #MEMs: number of MEMs of length higher than 20 bp shared by the pair.

ID	Species	Strains	Length	MUMi	#MEMs
P1	*Streptococcus pyogenes*	MGAS9429	1836467	0.006	1773
		MGAS2096	1860355		
P2	*Escherichia coli*	MG1655 K-12	4639675	0.248	58288
		O157:H7 Sakai	5498450		
P3	*Pseudomonas syringae*	1448A	5928787	0.666	117669
		B728a	6093698		

Table 2. List of segmentations studied in this paper. These segmentations correspond to the first strain of the pairs of genomes presented in Table 1. #BS: number of backbone segments, #VS: number of variable segments, BL: cumulative length of the backbone, BC: coverage of the backbone *i.e.* ratio between the backbone length and the complete genome length in %.

ID	ID pair	Strain	Software	#BS	#VS	BL	BC
S1	P1	MGAS9429	MGA	63	62	1536000	83.64%
S2	P1	MGAS9429	MAUVE	67	66	1761765	95.93%
S3	P2	MG1655 K-12	MGA	641	640	4098265	88.33%
S4	P2	MG1655 K-12	MAUVE	817	816	4091213	88.17%
S5	P3	1448A	MGA	2870	2871	3181778	58.73%
S6	P3	1448A	MAUVE	6176	6175	4303198	72.58%

3 Results

3.1 Calibration of the Simulation Process

In our method, two user-defined parameters have to be set: 1) the proportion p of MEMs to perturb and 2) the number N of simulations. When perturbing MEM positions in the compared genomes, we aim at maximizing the impact of perturbations in the segmentation computation procedure. In other words, we need to use a p high enough to perturb all the nucleotides belonging to a MEM at least once. However, if p is too high, some nucleotides will be systematically perturbed in all the N simulations and their score will be meaningless. As shown in Fig. 1, when $p \leq 0.1$ for $N = 1000$, almost no nucleotides are perturbed in all simulations for the three pairs of genomes. For higher values of p, the number of nucleotides that are systematically perturbed rapidly increases for the pairs P2 and P3. Consequently, we decided to set $p = 0.1$ for the three pairs of genomes. This analysis can be performed for any comparison of interest to select a specific value of p for each pair of genomes or, as done in this study, to determine a p value suitable for all the pairs. The computation of the scores is based on multiple

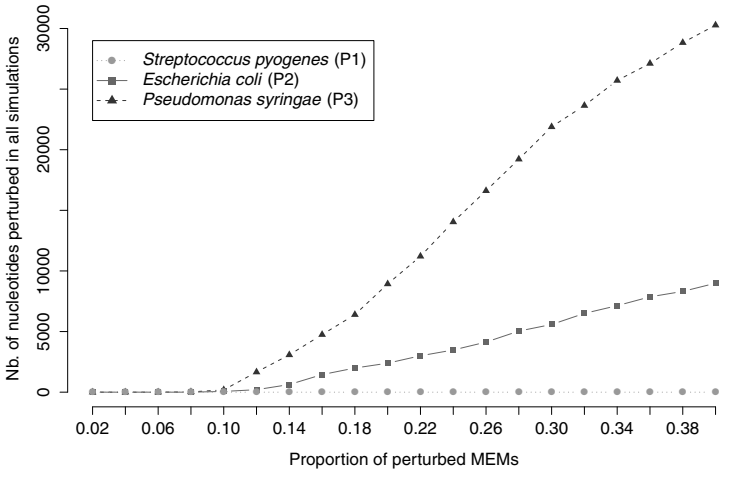


Fig. 1. Selection of the proportion p of MEMs to be perturbed for the three pairs of genomes. Each curve gives the number of nucleotides that have been systematically perturbed in each simulation ($N = 1000$) with respect to the proportion p of perturbed MEMs. P1, *S. pyogenes* (circles), P2, *E. coli* (squares), and P3, *P. syringae* (triangles).

simulations and requires a sufficient number N of simulations to be statistically reliable. In this work, we used $N = 1000$ that provided a good tradeoff between computation time and accuracy of the score.

3.2 The Nucleotide Score

Computation of the nucleotide scores was performed for the segmentations listed in Table 2. In each segmentation, we identified nucleotides whose score suggested that their assignment to a backbone or variable segment was highly sensitive to perturbations. To facilitate the analysis, we plotted the score for each nucleotide along the genome thus generating a score profile. Here we show different representative examples of score profiles and their interpretation.

Profiles of robust regions. Fig. 2A focuses on a region composed of a variable segment (in gray) surrounded by two backbone segments (in black), from the segmentation S3. It shows that nucleotides included in the variable segment have a score equal to 1 while those that belong to the backbone have a score around 0.1. The score profile at the boundaries of the segments is extremely sharp.

This kind of profile is representative of globally robust regions. Variable segments are associated with high nucleotide scores, near 1, while nucleotides from backbone fragments have very low scores. Note that the scores of the nucleotides belonging to a robust backbone rarely reach 0 but generally vary around 0.1.

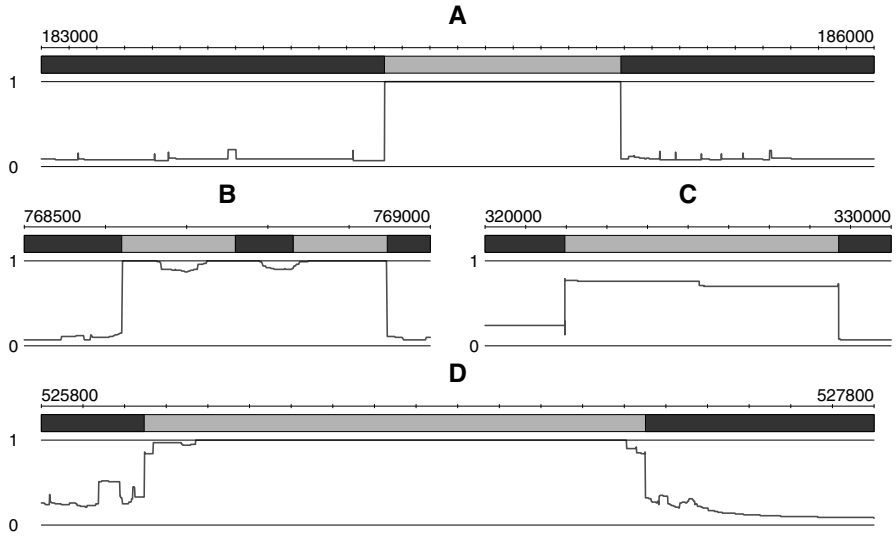


Fig. 2. Representative examples of nucleotide score profiles. The nucleotide score is plotted below the original segmentation, in which the backbone and variable segments are represented in black and gray, respectively. (A) a robust region from segmentation S3, (B) non-robust backbone segment from segmentation S6, (C) non-robust variable segment from segmentation S1, (D) Uncertain boundary determination from segmentation S3.

This is due to the fact that most nucleotides of the backbone belong to MEMs and are therefore likely to be perturbed during the simulation process. Consequently, a nucleotide that is robustly assigned to the backbone has a score expected around the value of p. This does not mean that backbone segments are generally less robust than variable segments.

The robustness score evaluates the difference between the original segmentation and the segmentations computed with the perturbed genomes. A robustly assigned nucleotide is expected to be identically assigned in the original segmentation and almost all perturbed segmentations. We suggest considering that a nucleotide is robustly assigned if its assignment is identical in the original segmentation and in at least 90% of the simulations. Consequently, nucleotides of a robust variable segment would have scores above 0.9, and nucleotides of a robust backbone segment would have scores below $p + 0.1$ (0.2).

Thus, the profile presented in Fig. 2A suggests that this part of the segmentation of S3 is strongly robust.

Profiles of non-robust regions. The segmentation portion displayed in Fig. 2B is composed of three backbone segments and two variable segments. The nucleotides of the two surrounding backbone fragments (left and right) have a low score, around 0.1, and thus, are robustly assigned. Those from the two

variable segments have score values almost equal to 1, suggesting that these two segments are robustly estimated too. The nucleotides of the backbone segment located between the two variable segments (in the middle) have scores near 1. This means that they are predicted in a variable segment in almost all the 1000 simulations. Consequently, this backbone fragment is absolutely not robust and should correspond to a variable segment.

Fig. 2C shows a part of the segmentation of S1. The scores of the nucleotides in this variable segment are rather low, with values around 0.8/0.7, which suggest that this variable segment is not robust.

These two examples illustrate the shapes of score profiles that are obtained while considering non-robust regions. Obviously, several other situations of non-robust region could be observed.

Robustness of segment boundaries. Fig. 2D focuses on a region composed of two backbone segments and one variable segment. Although the scores of the nucleotides of the variable segment are globally high, suggesting that this segment is robust, score profiles at the junctions of the segment are not as sharp as those plotted in Fig. 2A. This implies that the nucleotides located at the boundaries of this segment are less robustly assigned than the others. Such a profile highlights the fact that determination of the junction positions between the segments cannot be always accurately determined.

3.3 The Segment Score

The segment score associates one score value to each segment of the original segmentation for both backbone and variable segments. Its aim is to facilitate the analysis of the global segmentation robustness.

Fig. 3A displays the distribution of the segment score values obtained for the segmentation of S3. The distribution of the score values shows two distinct peaks, one for the variable segments (in gray) and one for the backbone segments (in black). Most of the variable segments have a score ranging from 0.9 to 1 and most of the backbone segments have a score ranging from 0 to 0.2. This suggests that most of the variable and backbone segments are robust. Thus, although there are few segments having intermediate score values, a rapid inspection of Fig. 3A allows us to conclude that the segmentation S3 is globally robust. This is clearly not the case for the segmentation S1 whose distributions of the backbone and variable segment scores are presented in Fig. 3B. These distributions are more spread out and consequently are not well separable. Almost one third of the variable segments have a segment score value below 0.8 indicating that they are not robust. About 20% of backbone segments have a score above 0.3. This implies that they are also non-robust. Thus, examination of Fig. 3B yields the conclusion that this segmentation is not globally robust.

The examples presented here clearly suggest that the segment score may be employed as an easy-to-use method to quickly evaluate the global robustness of segmentations.

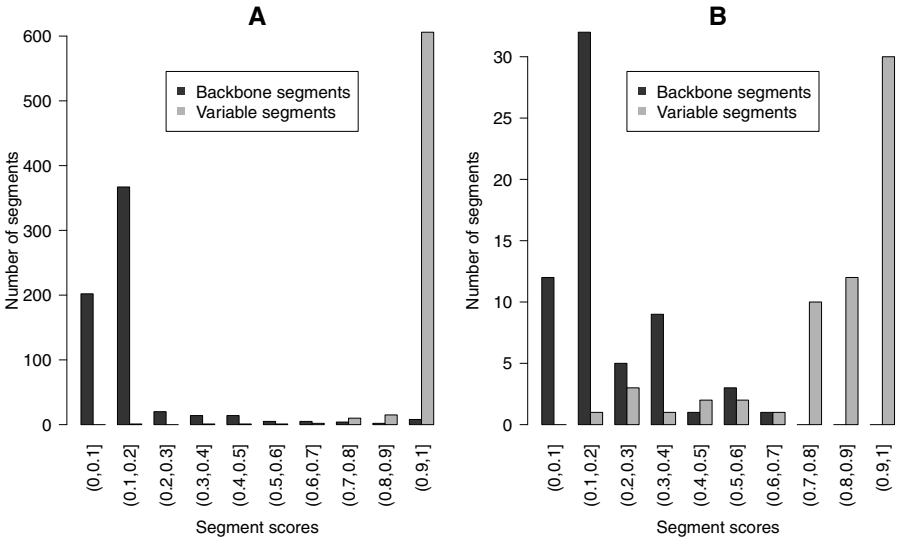


Fig. 3. Distribution of segment score values. (A) Segmentation S3: example of a globally robust segmentation. (B) Segmentation S1: example of a non-robust segmentation.

3.4 Comparing Segmentations with the Robustness Scores

Segmentation computation of a given pair of genomes can be performed using different aligner tools, providing one segmentation per alignment method. Two segmentations of the same pair of genomes may yield significantly different results as shown in Table 2. In this subsection, we assess whether our scores can be used to understand the potential disparities between segmentations and possibly to determine the best one.

Comparison of the six segmentations listed in Table 2 allowed us to illustrate this point. There were three possible comparisons: S1 *vs.* S2, S3 *vs.* S4 and S5 *vs.* S6. For each comparison, the first segmentation was computed with MGA and the second one with MAUVE. We analyzed the differences between these pairs of segmentations. To do so, we identified all the nucleotides that were assigned to the backbone in one of the segmentations while they were assigned to a variable segment in the other segmentation. Thus, we considered two distinct situations: 1) A nucleotide was assigned to a variable segment in the first segmentation (MGA) while it was assigned to the backbone in the second segmentation (MAUVE); 2) the reverse situation, a nucleotide was assigned to the backbone in the first segmentation while it was assigned to a variable segment in the second segmentation. For each comparison, we counted the number of nucleotides corresponding to situation 1 and 2. We also computed the average nucleotide score (subsection 2.3) over all nucleotides in each situation. This is summarized in Table 3.

Table 3. Comparisons of segmentation pairs: S1 *vs.* S2, S3 *vs.* S4, and S5 *vs.* S6. #Nucleotides: number of nucleotides assigned by MGA to variable segments in the first segmentation but assigned by MAUVE to the backbone in the second segmentation (situation 1) and the converse situation (situation 2). Score: average nucleotide score, the score values of the nucleotides that were assigned to variable segments appear in bold.

	Situation 1	Situation 2	Total
S1 *vs.* S2			
#Nucleotides	226064	299	226363
Score S1 (MGA)	**0.755**	0.301	
Score S2 (MAUVE)	0.068	**0.736**	
S3 *vs.* S4			
#Nucleotides	18584	25636	44220
Score S3 (MGA)	**0.939**	0.319	
Score S4 (MAUVE)	0.314	**0.851**	
S5 *vs.* S6			
#Nucleotides	975750	154490	1130240
Score S5 (MGA)	**0.999**	0.173	
Score S6 (MAUVE)	0.085	**0.988**	

Identification of segmentation errors: S1 *vs.* S2. Although S1 and S2 have almost the same number of segments, the cumulative backbone length of S2 is much higher than the one of S1 (see Table 2). More than 226000 nucleotides are assigned to the backbone in S2 while they are assigned to variable segments in S1 (situation 1). Average nucleotide scores of these nucleotides indicate that the backbone assignment is more robust in S2 (average score of 0.068) than the variable segment assignment in S1 (average score of 0.755).

Analysis of the non-robust variable segments of S1 revealed that these segments were associated with large overlapping MEMs. Indeed, when comparing very closely related genomes, long MEMs (> several kb) can overlap together. This induces problems in the chaining step of the alignment for MGA because its chaining algorithm discards overlapping MEMs [22]. Thus, comparing extremely close genomes with MGA might produce unexpected variable segments. Our method is able to detect this problem, assigning a low robustness to these variable segments and indicating that S2 is globally more robust than S1.

Comparing very similar segmentations: S3 *vs.* S4. Although S4 (MAUVE) contains more segments than S3 (MGA), these segmentations have identical backbone coverage around 88% (see Table 2) and thus, these segmentations are very similar. Segment score distributions of S3 and S4 reveal that these segmentations are globally robust (see Fig. 3A and data not shown). In this case two similar segmentations produced by two distinct aligner tools are associated with a similar robustness score indicating that our method does not tend to favor particularly one algorithm over the other.

The robustness score is also relevant to compare the slight differences between S3 and S4. About 18000 nucleotides correspond to situation 1 and 25000 to situation 2 representing about 0.5% of the genome length in both cases (see Table 3). Table 3 indicates that in both situations, variable segments are more robust, with average scores equal to 0.939 and 0.851, than backbone with average scores around 0.3. Consequently, for the nucleotides in situation 1, the assignment made by MGA (S3) is globally more robust than the one made by MAUVE (S4). Conversely, MAUVE seems to provide a more robust segmentation than MGA for the nucleotides in situation 2.

Selection of the best suited aligner: S5 *vs.* S6. There is a large difference between the backbone lengths of segmentations S5 and S6 (see Table 2). Backbone of S6 is about 35% longer than the one of S5. However, Table 3 shows that in both situations, backbone segments have average scores less than 0.2 and variable segments have average scores greater than 0.9. This means that, in both situations, the two segmentations are robust. Consequently, from this result, it is not obvious to decide which segmentation is the best one. Investigation of these comparisons revealed that the dissimilarities were associated with rearranged regions between these genomes. Since MGA does not treat rearrangements, it systematically ignores rearranged regions and classifies them into variable segments yielding to robust, yet biologically not relevant, variable segments. It is important to note that our score is dedicated to evaluate how a segmentation is sensitive to perturbations on the original sequences. In this trivial example, MGA is incorrectly used, producing robust variable segments when considering conserved rearranged regions. This example shows that before computing segmentations, it is crucial to perfectly know the compared genomes and to clearly identify the limitations of the aligner tool used.

4 Discussion

For the first time, an attempt has been made to statistically assess the robustness of bacterial genome segmentations at different scales, from nucleotides to entire segments and even to the whole segmentation. The two proposed scores provide useful information, are easy to implement and their interpretation is intuitive. Their computation can be performed with any aligner tool and does not require any external data. We also have clearly shown that these scores are able to identify both robust and non-robust fractions of segmentations. Comparison of segmentations obtained with different aligner tools and their corresponding robustness scores provide fruitful information to identify inaccuracies.

The distance between the compared genomes is not a limitation for computing our scores. Indeed, although segmentations are usually computed on closely related genomes, the distance between the compared genomes can vary (see MUMi index in Table 1). This involves important variations in the number of MEMs that are retrieved between the sequences (see #MEMs in Table 1). However, we showed that our scores can be used in a wide range of distances between

the compared genomes. As a consequence, our method can be extended to the comparison of bacterial genomes at the inter-species level, provided that the compared sequences are close enough to be aligned. Last, it is possible to adapt the method for the comparison of eukaryotic genomes, for example, by defining additional perturbation types such as duplications or transpositions, but with some limitations due to the computation time required to align such large sequences.

Although homologous regions are expected to belong to the backbone, some of them can be assigned to variable segments. Indeed, homologous regions are more or less diverged along the chromosomes. This implies that the most diverged ones can be either classified as backbone or as variable segments depending on the aligner tool used and on the parameter setting. Such regions are expected to show a low robustness.

The lack of robustness of a segmentation or a part of it can be due to different reasons. First, non-robust segments are often associated with repeated regions that are known to be often incorrectly treated by aligner tools [25]. Problems can also arise from the chaining algorithm or from calibration of the parameters of the complete genome aligner. Indeed, variations of some parameter values, such as the minimal length of MEMs to retrieve, have an important impact on the alignment results [5]. Last, computation of segmentations often requires the use of local alignment tools to investigate specific short regions. This step of the segmentation determination can also be at the origin of low robustness [15]. The two scores proposed here do not determine directly the reason explaining the lack of robustness of a segmentation but they allow to easily identify the non-robust regions and hence, they facilitate investigations to determine the origin of this behavior.

The scores presented here can be used at a larger scale to compare and to categorize simultaneously several segmentations. They can be calculated for standalone comparisons [4], but they could also be directly integrated into databases that store numerous pre-computed complete genome comparisons such as xBASE [26] and MOSAIC [5,24] to provide an evaluation of the robustness of their contents. In this context, the development of a unique global segmentation score or a statistical test directly derived from the segment scores should be useful.

Our scores raise many questions about genome structure, segmentation determination and alignment algorithms. Currently, the method is implemented only for pairwise comparisons at the intra-species level. However, further improvements are in progress to adapt the approach to multiple genome comparisons. The most challenging task is to determine the optimal number of MEMs to perturb by accounting for the combinatorial aspect that this process implies. Preliminary investigations showed that this issue can be addressed by using a subset of MEMs called "rare multiMEMs" corresponding to MEMs that are repeated a limited number of times between the compared genomes [27].

The originality of our method relies on the fact that we investigate the robustness of a segmentation rather than its biological relevance as it has been done in other studies. Our scores are therefore complementary to other measures such

as the number of backbone segments that disrupt orthologous genes proposed by Swidan and Shamir [17]. They could even be used jointly to better assess the reliability of genome segmentations.

Acknowledgments. This work was supported by the French "Agence Nationale de la Recherche" project CoCoGen (BLAN07-1_185484). We are grateful to the INRA MIGALE bioinformatics platform (http://migale.jouy.inra.fr) for their help and computational resources. We thank Dr JM Pedraza for valuable comments on this work.

References

1. Kellis, M., Patterson, N., Endrizzi, M., Birren, B., Lander, E.S.: Sequencing and comparison of yeast species to identify genes and regulatory elements. Nature 423, 241–254 (2003)
2. Goto, N., Kurokawa, K., Yasunaga, T.: Analysis of invariant sequences in 266 complete genomes. Gene 401, 172–180 (2007)
3. Halpern, D., Chiapello, H., Schbath, S., Robin, S., Hennequet-Antier, C., Gruss, A., El Karoui, M.: Identification of DNA motifs implicated in maintenance of bacterial core genomes by predictive modeling. PLoS genet. 9, 153–160 (2007)
4. Hayashi, T., Makino, K., Ohnishi, M., Kurokawa, K., Ishii, K., Yokoyama, K., Han, C.G., Ohtsubo, E., Nakayama, K., Murata, T., et al.: Complete genome sequence of enterohemorrhagic *Escherichia coli* O157:H7 and genomic comparison with a laboratory strain K-12. DNA Res. 8, 11–22 (2001)
5. Chiapello, H., Bourgait, I., Sourivong, F., Heuclin, G., Gendrault-Jacquemard, A., Petit, M.A., El Karoui, M.: Systematic determination of the mosaic structure of bacterial genomes: species backbone *versus* strain-specific loops. BMC Bioinformatics 6, 171–180 (2005)
6. Touchon, M., Hoede, C., Tenaillon, O., Barbe, V., Baeriswyl, S., Bidet, P., Bingen, E., Bonacorsi, S., Bouchier, C., Bouvet, O., et al.: Organised genome dynamics in the *Escherichia coli*species results in highly diverse adaptive paths. Plos Genet. 5, 1000344 (2009)
7. Canchaya, C., Duperchy, E., Brussow, H.: The impact of prophages on bacterial chromosomes. Mol. Microbiol. 53, 9–18 (2004)
8. Prentice, M.B.: Bacterial comparative genomics. Genome Biol. 5, 338 (2004)
9. Touzain, F., Denamur, E., Médique, C., Barbe, V., El Karoui, M., Petit, M.A.: Small variable segments constitute a major type of diversity of bacterial genomes at the species level. Genome Biol. 11, R45 (2010)
10. Miller, W.: Comparison of genomic DNA sequences: solved and unsolved problems. Bioinformatics 17, 391–397 (2001)
11. Delcher, A.L., Kasif, S., Fleischmann, R.D., Peterson, J., White, O., Salzberg, S.L.: Alignment of whole genomes. Nucleic Acids Res. 27, 2369–2376 (1999)
12. Kurtz, S., Phillipy, A., Delcher, A.L., Smoot, M., Shumway, M., Antonescu, C., Salzberg, S.L.: Versatile and open software for comparing large genomes. Genome Biol. 5, R12 (2004)
13. Treangen, T.J., Messeguer, X.: M-GCAT: interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. BMC Bioinformatics 7, 433–447 (2006)

14. Dubchak, I., Pachter, L.: The computational challenges of applying comparative-based computational methods to whole genomes. Brief. Bioinformatics 3, 18–22 (2002)
15. Prakash, A., Tompa, M.: Measuring the accuracy of genome-size multiple alignments. Genome Biol. 8, R124 (2007)
16. Margulies, E.H., Cooper, G.M., Asimenos, G., Thomas, D.J., Dewey, C.N., Siepel, A., Birney, E., Keefe, D., Schwartz, A.S., Hou, M., et al.: Analyses of deep mammalian sequence alignments and constraint predictions for 1% of the human genome. Genome Res. 17, 760–774 (2007)
17. Swidan, F., Shamir, R.: Assessing the quality of whole genome alignments in bacteria. Adv. Bioinformatics 1, 1–8 (2009)
18. Kitano, H.: Biological robustness. Nat. Rev. Genet. 5, 826–837 (2004)
19. Guyon, F., Guénoche, A.: Comparing bacterial genomes from linear orders of paterns. Discrete Appl. Math. 156, 1251–1262 (2008)
20. Hoebeke, M., Nicolas, P., Bessieres, P.: MuGeN: simultaneous exploration of multiple genomes and computer analysis results. Bioinformatics 19, 859–864 (2003)
21. Deloger, M., El Karoui, M., Petit, M.A.: A genomic distance based on MUM indicates discontinuity between most bacterial species and genera. J. Bacteriol. 191, 91–99 (2009)
22. Höhl, M., Kurtz, S., Ohlebusch, E.: Efficient multiple genome alignment. Bioinformatics 18, S312–S320 (2002)
23. Darling, A.C., Mau, B., Blattner, F.R., Perna, N.T.: Mauve: multiple alignment of conserved genomic sequence with rearrangements. Genome Res. 15, 184–194 (2004)
24. Chiapello, H., Gendrault, A., Caron, C., Blum, J., Petit, M.A., El Karoui, M.: MOSAIC: an online database dedicated to the comparative genomics of bacterial strains at the intra-species level. BMC Bioinformatics 9, 498–506 (2008)
25. Dubchak, I., Poliakov, A., Kislyuk, A., Brudno, M.: Multiple whole-genome alignments without a reference organism. Genome Res. 19, 682–689 (2009)
26. Chaudhuri, R.R., Pallen, M.J.: xBASE, a collection of online databases for bacterial comparative genomics. Nucleic Acids Res. 34, 335–337 (2006)
27. Abouelhoda, M.I., Kurtz, S., Ohlebusch, E.: CoCoNUT: an efficient system for the comparison and analysis of genomes. BMC Bioinformatics 9, 456–462 (2008)

An Algorithm to Solve the Motif Alignment Problem for Approximate Nested Tandem Repeats

Atheer A. Matroud[1,2], Michael D. Hendy[1,2], and Christopher P. Tuffley[2]

[1] Allan Wilson Centre for Molecular Ecology and Evolution, Massey University, Private Bag 11222, Palmerston North, New Zealand
[2] Institute of Fundamental Sciences, Massey University, Private Bag 11222, Palmerston North, New Zealand

Abstract. An *approximate nested tandem repeat* (NTR) in a string $\mathbf{T}$ is a complex repetitive structure consisting of many approximate copies of two substrings $\mathbf{x}$ and $\mathbf{X}$ ("motifs") interspersed with one another. NTRs have been found in real DNA sequences and are expected to have applications for evolutionary studies, both as a tool to understand concerted evolution, and as a potential marker in population studies.

In this paper we describe software tools developed for database searches for NTRs. After a first program `NTRFinder` identifies putative NTR motifs, a confirmation step requires the application of the alignment of the putative NTR against exact NTRs built from the putative template motifs $\mathbf{x}$ and $\mathbf{X}$. In this paper we describe an algorithm to solve this alignment problem in $O(|\mathbf{T}|(|\mathbf{x}| + |\mathbf{X}|))$ space and time. Our alignment algorithm is based on Fischetti et al.'s wrap-around dynamic programming.

1 Introduction

An *approximate nested tandem repeat* (NTR) in a string $\mathbf{T}$ is a complex repetitive structure consisting of many approximate copies of two substrings $\mathbf{x}$ and $\mathbf{X}$ ("motifs") interspersed with one another. The name derives from the fact that an NTR may be thought of as two tandem repeats nested within one another.

Approximate nested tandem repeats have been found in real DNA sequences, such as that of *Colocasia esculenta*, the ancient staple food crop taro (Matroud *et al.* [6]). The intergenic spacer (IGS) region in the taro ribosomal DNA contains an NTR consisting of eleven approximate copies of a 48 bp motif, interspersed within a tandem repeat consisting of 96 approximate copies of an 11 bp motif. The NTR found in taro, used as a genetic marker, offers the potential to elucidate the prehistory of the early agriculture of this ancient food crop, as mutation events appear to be accumulating on a thousand-year time scale. NTRs in general also offer an opportunity to investigate concerted evolution whereby mutations are propagated throughout the many hundreds of copies of the IGS region in the taro genome.

To develop a fuller understanding of the nature of the NTR, we have developed software to find them [6]. This comprises two phases. The first phase is the detection

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 188–197, 2010.

phase where the sequence is scanned and candidate NTRs are detected then the two consensus motifs $\mathbf{X}$ and $\mathbf{x}$ that form the NTR are constructed. The second phase is the verification phase where the putative NTR is aligned against all patterns of the form

$$\mathbf{x}^{s_0}\mathbf{X}^{t_0}\mathbf{x}^{s_1}\mathbf{X}^{t_1}\cdots\mathbf{x}^{s_k}\mathbf{X}^{t_k}.$$

Such an alignment is needed to find the extent and structure of the NTR (that is, to find the exponents s_i, t_i occurring above), and may also be used to evaluate the fit of the template motifs $\mathbf{x}$ and $\mathbf{X}$. We call this problem the *motif alignment problem* for NTRs, to distinguish it from the *variant alignment problem* that arises at later stages of the analysis.

The variant alignment problem is the problem of aligning two different NTRs with the same or similar motifs, such as might be found in different varieties of the same plant, with a view to finding the extent of their shared evolutionary history. At that stage of the analysis the copy variants of the template motifs will be treated as the characters to be aligned; clearly, a good solution to the motif alignment problem is a necessary prerequisite for this.

The purpose of this paper is to present an algorithm to solve the motif alignment problem for approximate NTRs, given a sequence $\mathbf{T}$, and the motifs $\mathbf{x}$ and $\mathbf{X}$ identified by our NTR search algorithm `NTRFinder` [6]. Our alignment algorithm runs in $O(|\mathbf{T}|(|\mathbf{x}|+|\mathbf{X}|))$ space and time, and plays a key rule in the verification of the putative NTR. It is based on the wrap-around dynamic programming technique introduced by Fischetti *et al.* [3] to solve the corresponding problem for (ordinary) tandem repeats. We show it can be readily adapted for use with more complex repetitive structures built from three or more motifs (discussed in Section 4.4).

2 Related Work

Many algorithms have been introduced to solve the problem of finding the similarity among sequences (see [4] for an overview); String similarity algorithms under edit distance (where insertion, deletion and mutation is allowed) were investigated recently where dynamic programming (DP) technique is one common approach to produce an optimal solution. This approach uses a scoring function that plays a critical role in the final alignment output.

Different alignment problems have been studied in the past. Some studied the alignment of two entire strings A and B [8], while others studied the alignment in substrings of the strings A and B [10]. Another alignment problem was studied where all the occurrences of string B in string A (See [7] for a survey). Another useful alignment problem is finding the substring of $\mathbf{T}$ which best matches a substring of x^s for $s > 1$. To solve this problem, Fischetti *et al.* introduced the wrap-around dynamic programming [3] which has $O(mn)$ space and time complexity. This algorithm was used by different tandem repeat finder programs to verify and produce the final report [1,2,5,11].

Standard tandem repeat software will generally not find more than one band of an NTR. The algorithm of Hauth and Joseph [5] searches for NTRs with motifs limited to lengths of up to 6 nucleotides. The current implementation of our algorithm [6]

NTRFinder searches for motifs of up to 100 nucleotides. Xstream [9] is a program which finds NTRs which are also tandem repeats, so that the number of repeats of the interspersed motif is fixed. NTR-Finder is the only software that can locate the NTR of taro IGS that we are investigating.

3 Definitions

3.1 Alphabets and Strings

An *alphabet* is a nonempty set Σ of symbols or *characters*, and a *string* over Σ is a finite sequence of elements of Σ. We write Σ^* for the set of all strings over the alphabet Σ, and $|\mathbf{S}|$ for the length of the string $\mathbf{S}$.

Given a string $\mathbf{S}$ and integers i, j such that $0 < i \leq j \leq |\mathbf{S}|$, we will write $\mathbf{S}[i]$ for the ith character of $\mathbf{S}$, and $\mathbf{S}[i, j]$ for the substring consisting of the ith to jth characters of $\mathbf{S}$. Given a second string $\mathbf{T}$, the *concatenation* of $\mathbf{S}$ and $\mathbf{T}$ is the string $\mathbf{ST}$, where

$$(\mathbf{ST})[i] = \begin{cases} \mathbf{S}[i] & \text{if } i \leq |\mathbf{S}|, \\ \mathbf{T}[i - |\mathbf{S}|] & \text{if } i > |\mathbf{S}|. \end{cases}$$

In applications to DNA sequences Σ is typically the set $\{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$, and we will use this alphabet in examples. However, our algorithm is not restricted to this case.

3.2 Tandem Repeats and Nested Tandem Repeats

An *exact tandem repeat* is a string of the form $\mathbf{X}^l$ for some $l \geq 2$. Thus, an exact tandem repeat is a string comprised of two or more contiguous exact copies of the same substring $\mathbf{X}$. This substring is called the *motif* of the tandem repeat. We obtain an *approximate tandem repeat* by allowing approximate rather than exact copies of the template motif $\mathbf{X}$. More precisely, an approximate tandem repeat is a string of the form $\mathbf{X}_1\mathbf{X}_2 \cdots \mathbf{X}_l$, where $d(\mathbf{X}, \mathbf{X}_i) \leq k|\mathbf{X}|$ for each i, for some fixed $k < 1$ and template motif $\mathbf{X}$. Where the value of the parameter k is important we may say that we have a *k-approximate tandem repeat* (k-TR). For simplicity of notation, we will write $\tilde{\mathbf{X}}^l$ to mean an approximate tandem repeat, consisting of l approximate copies of $\mathbf{X}$.

Given two motifs $\mathbf{X}$ and $\mathbf{x}$ such that $d(\mathbf{X}, \mathbf{x}) \gg 0$, an *exact nested tandem repeat* is a string of the form

$$\mathbf{x}^{s_0}\mathbf{X}^{t_0}\mathbf{x}^{s_1}\mathbf{X}^{t_1} \cdots \mathbf{x}^{s_n}\mathbf{X}^{t_n},$$

where $n > 1$, $s_i \geq 1$ for each i, and $t_i \geq 1$ for $i = 1, \ldots, n-1$. We again obtain an *approximate nested tandem repeat* by allowing the copies of the motifs $\mathbf{X}$ and $\mathbf{x}$ to be approximate rather than exact. Thus, an approximate nested tandem repeat is a string of the form

$$\tilde{\mathbf{x}}^{s_0}\tilde{\mathbf{X}}^{t_0}\tilde{\mathbf{x}}^{s_1}\tilde{\mathbf{X}}^{t_1} \cdots \tilde{\mathbf{x}}^{s_n}\tilde{\mathbf{X}}^{t_n},$$

where $n > 1$, $s_i \geq 1$ for each i, $t_i \geq 1$ for $i = 1, \ldots, n-1$, and such that $\tilde{\mathbf{x}}^{s_0}\tilde{\mathbf{x}}^{s_1} \cdots \tilde{\mathbf{x}}^{s_n}$ is an approximate tandem repeat with motif $\mathbf{x}$, and $\tilde{\mathbf{X}}^{t_0}\tilde{\mathbf{X}}^{t_1} \cdots \tilde{\mathbf{X}}^{t_n}$ is an approximate tandem repeat with motif $\mathbf{X}$.

Note that the definition of an approximate nested tandem repeat includes exact nested tandem repeats as a special case. "Nested tandem repeat" or "NTR" by itself will always mean an *approximate* nested tandem repeat, unless explicitly stated otherwise.

Remark. The definition of an NTR given here is slightly more general than that given in Matroud *et al.* [6]. In that paper, a nested tandem repeat is required to satisfy $t_i \leq 1$ for each i.

3.3 Alignment

Given an alphabet Σ, let $\bar{\Sigma}$ be the alphabet $\Sigma \cup \{-\}$, where "$-$" ("*gap*") is a character that does not belong to Σ. We define $\phi : \bar{\Sigma}^* \to \Sigma^*$ to be the function that deletes all gaps.

Given two strings $\mathbf{A}, \mathbf{B} \in \Sigma^*$, an *alignment* of $\mathbf{A}$ and $\mathbf{B}$ is a choice of a pair of strings $(\bar{\mathbf{A}}, \bar{\mathbf{B}}) \in \bar{\Sigma}^* \times \bar{\Sigma}^*$ satisfying the following conditions:

A1. $\phi(\bar{\mathbf{A}}) = \mathbf{A}$ and $\phi(\bar{\mathbf{B}}) = \mathbf{B}$;
A2. $|\bar{\mathbf{A}}| = |\bar{\mathbf{B}}|$; and
A3. there is no index i for which $\bar{\mathbf{A}}[i] = \bar{\mathbf{B}}[i] = -$.

Thus, $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are obtained from $\mathbf{A}$ and $\mathbf{B}$ respectively by inserting gaps in such a way that the resulting strings have the same length, and do not both have a gap in the same position.

To score an alignment we use a scoring matrix σ, which specifies the reward or penalty for aligning any two characters of $\bar{\Sigma}$ against each other. See Table 1 for an example. We will assume throughout that σ penalises gaps (that is, $\sigma(-, \alpha)$ and $\sigma(\alpha, -)$ are both negative for all $\alpha \in \bar{\Sigma}$), and we set $\sigma(-, -) = -\infty$ to reflect condition A3 above. Given an alignment $(\bar{\mathbf{A}}, \bar{\mathbf{B}})$ for which $|\bar{\mathbf{A}}| = |\bar{\mathbf{B}}| = L$, the *alignment score* of $(\bar{\mathbf{A}}, \bar{\mathbf{B}})$ is then defined to be

$$\sigma(\bar{\mathbf{A}}, \bar{\mathbf{B}}) = \sum_{i=1}^{L} \sigma(\bar{\mathbf{A}}[i], \bar{\mathbf{B}}[i]).$$

An *optimal global alignment* is an alignment of $\mathbf{A}$ and $\mathbf{B}$ which maximises the alignment score over all such alignments. See Navarro [7] for a survey of this and other alignment problems.

Table 1. A sample scoring matrix for DNA sequences. This matrix rewards matching characters from Σ with a score of $+1$, and penalises mis-matching characters from Σ with a score of -1. The penalty for aligning a gap against a character from Σ is -2. The value $\sigma(-, -) = -\infty$ reflects condition A3, which prohibits a gap being aligned against a gap.

σ	$-$	A	C	G	T
$-$	$-\infty$	-2	-2	-2	-2
A	-2	1	-1	-1	-1
C	-2	-1	1	-1	-1
G	-2	-1	-1	1	-1
T	-2	-1	-1	-1	1

4 The Motif Alignment Problem for Approximate Nested Tandem Repeats

4.1 The Problem

The *motif alignment problem for approximate nested tandem repeats* is the following:

Given
1. a string $\mathbf{T}$ and motifs $\mathbf{x}$ and $\mathbf{X}$ over the alphabet Σ, and
2. a scoring matrix σ defined over $\bar{\Sigma} \times \bar{\Sigma}$,

find an optimal alignment of $\mathbf{T}$ against substrings of strings of the form

$$\mathbf{x}^{s_0}\mathbf{X}^{t_0}\mathbf{x}^{s_1}\mathbf{X}^{t_1}\cdots\mathbf{x}^{s_k}\mathbf{X}^{t_k}.$$

Thus, given a string $\mathbf{T}$ that is presumed to contain an approximate nested tandem repeat with motifs $\mathbf{x}$ and $\mathbf{X}$, and a scoring matrix σ, the problem is to find a optimal alignment of $\mathbf{T}$ against substrings of *exact* nested tandem repeats with motifs $\mathbf{x}$ and $\mathbf{X}$.

4.2 Solution to the Problem via Nested Wrap-Around Dynamic Programming

The motif alignment problem for NTRs is closely related to the corresponding problem for tandem repeats, which was solved by Fischetti *et al.* [3] using wrap-around dynamic programming. We solve the problem by adapting their technique. The key differences are the introduction of a second matrix, to hold information relating to the second motif, and a modification to the update rule used between the first and second passes.

In what follows we let $n = |\mathbf{T}|$, $m = |\mathbf{x}|$, and $l = |\mathbf{X}|$. An optimal alignment will be calculated using two matrices $D^{(1)}$ and $D^{(2)}$. The matrix $D^{(1)}$ is $(m+1) \times (n+1)$, and will record scores related to aligning portions of $\mathbf{T}$ against $\mathbf{x}$, while the matrix $D^{(2)}$ is $(l+1) \times (n+1)$, and will record scores related to aligning portions of $\mathbf{T}$ against $\mathbf{X}$. Both matrices will be indexed starting from 0, and we will denote the (i, j) entry of $D^{(k)}$ by $D^{(k)}[i, j]$. We write $D^{(k)}_{i,j}$ for the upper-left $(i+1) \times (j+1)$ submatrix of $D^{(k)}$.

The score matrices $D^{(1)}$ and $D^{(2)}$ are filled as follows:

1. We initialise the two matrices by setting

$$D^{(k)}[0, j] := 0, \qquad D^{(k)}[i, 0] := 0$$

for all i, j and k.
2. We compute each column of the matrices (starting from $j = 1$) in two rounds. In the first round we compute $D^{(1)}[i, j]$ using the recursive function

$$D^{(1)}[i, j] := \max \left\{ \begin{array}{c} D^{(1)}[i-1, j-1] + \sigma(\mathbf{x}[i], \mathbf{T}[j]), \\ D^{(1)}[i-1, j] + \sigma(\mathbf{x}[i], -), \\ D^{(1)}[i, j-1] + \sigma(-, \mathbf{T}[j]) \end{array} \right\}.$$

We then compute $D^{(2)}[i, j]$ in a similar fashion.

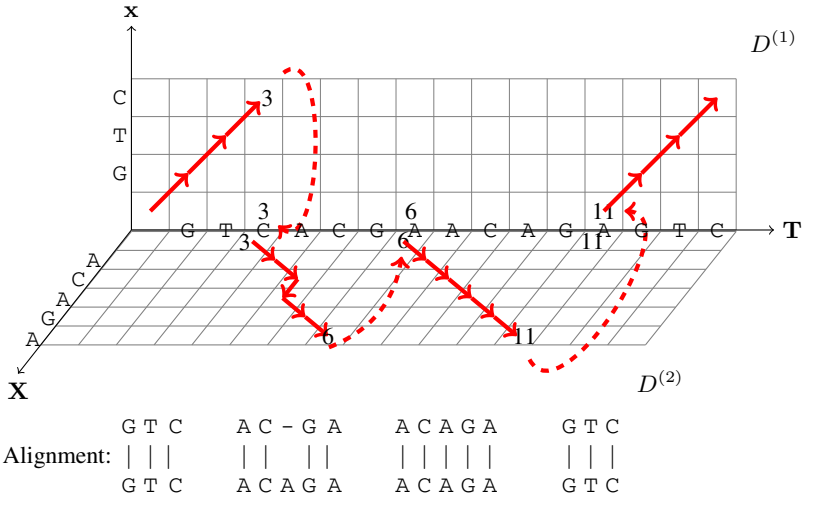


Fig. 1. Visualisation of the algorithm applied to the string $\mathbf{T} = \texttt{GTCACGAACAGAGTC}$, with template motifs $\mathbf{x} = \texttt{GTC}$, $\mathbf{X} = \texttt{ACAGA}$. The matrix $D^{(1)}$ lies in the $(\mathbf{x}, \mathbf{T})$ plane, while $D^{(2)}$ lies in the $(\mathbf{X}, \mathbf{T})$ plane. The majority of the matrix entries have been omitted for clarity. Solid arrows represent the optimal alignment path, while dashed arrows indicate that the value at its tail is fed to the location at its head. The corresponding alignment appears below the diagram.

In the second round, we update both $D^{(1)}[0, j]$ and $D^{(2)}[0, j]$ with the value $\max\{D^{(1)}[m, j], D^{(2)}[l, j]\}$, and then update $D^{(1)}[i, j]$ for $1 \leq i \leq m$ using the formula above, which simplifies to

$$D^{(1)}[i, j] := \max\{D^{(1)}[i, j], D^{(1)}[i-1, j] + \sigma(\mathbf{x}[i], -)\}$$

during the second round. The entries $D^{(2)}[i, j]$ for $1 \leq i \leq l$ are then updated in a similar fashion.

Pseudo-code for the matrix-filling algorithm appears as Algorithm 1 below.

Once the matrices are filled, an optimal alignment is found using the standard traceback procedure for dynamic programming (see for example Fischetti *et al.* [3]), beginning from the largest entry in the righthand columns of $D^{(1)}$, $D^{(2)}$. The algorithm clearly has space complexity $O(n(m+l))$, and the matrices $D^{(1)}$ and $D^{(2)}$ are filled in time $O(n(m+l))$.

4.3 Correctness of the Algorithm

We now prove by induction that the matrices $D^{(1)}$ and $D^{(2)}$ have been calculated correctly to produce the optimal alignment. In what follows let $NTR(\mathbf{x}, \mathbf{X})$ denote the set of all strings $\mathbf{N}$ that occur as substrings of exact NTRs with motifs $\mathbf{x}$ and $\mathbf{X}$.

Suppose that the two sub-matrices $D^{(1)}_{m,j-1}$ and $D^{(2)}_{l,j-1}$ have been correctly computed for some $j \geq 1$. That is, assume that $D^{(1)}[i, j-1]$ is the optimal alignment score of

Data: Strings $\mathbf{T}$, $\mathbf{X}$, $\mathbf{x}$ and scoring matrix σ
Result: Matrices $D^{(1)}$, $D^{(2)}$ containing optimal alignment scores with respect to σ of alignments of $\mathbf{T}$ against substrings of exact NTRs with motifs $\mathbf{X}$ and $\mathbf{x}$
for $j = 0$ **to** $|\mathbf{T}|$ **do**
  for $i = 0$ **to** $|\mathbf{x}|$ **do**
    $D^{(1)}[i, j] := 0$
  end
  for $i = 1$ **to** $|\mathbf{X}|$ **do**
    $D^{(2)}[i, j] := 0$
  end
end
for $j = 1$ **to** $|\mathbf{T}|$ **do**
  for $i = 1$ **to** $|\mathbf{x}|$ **do**
    $D^{(1)}[i, j] := \max\{D^{(1)}[i-1, j-1] + \sigma(\mathbf{x}[i], \mathbf{T}[j]),$
    $D^{(1)}[i-1, j] + \sigma(\mathbf{x}[i], -), D^{(1)}[i, j-1] + \sigma(-, \mathbf{T}[j])\}$
  end
  for $i = 1$ **to** $|\mathbf{X}|$ **do**
    $D^{(2)}[i, j] := \max\{D^{(2)}[i-1, j-1] + \sigma(\mathbf{X}[i], \mathbf{T}[j]),$
    $D^{(2)}[i-1, j] + \sigma(\mathbf{X}[i], -), D^{(2)}[i, j-1] + \sigma(-, \mathbf{T}[j])\}$
  end
  $D^{(1)}[0, j] := \max\{D^{(1)}[|\mathbf{x}|, j], D^{(2)}[|\mathbf{X}|, j]\}$
  $D^{(2)}[0, j] := \max\{D^{(1)}[|\mathbf{x}|, j], D^{(2)}[|\mathbf{X}|, j]\}$
  for $i = 1$ **to** $|\mathbf{x}|$ **do**
    $D^{(1)}[i, j] := \max\{D^{(1)}[i, j], D^{(1)}[i-1, j] + \sigma(\mathbf{x}[i], -)\}$
  end
  for $i = 1$ **to** $|\mathbf{X}|$ **do**
    $D^{(2)}[i, j] := \max\{D^{(2)}[i, j], D^{(2)}[i-1, j] + \sigma(\mathbf{X}[i], -)\}$
  end
end

Algorithm 1. Pseudo-code for our nested wrap-around dynamic programming algorithm for the motif alignment problem for NTRs

any alignment of $\mathbf{T}[1, j-1]$ against a string $\mathbf{N} \in NTR(\mathbf{x}, \mathbf{X})$ that ends with a suffix of $\mathbf{x}[1, i]$, and similarly that $D^{(2)}[i, j-1]$ is the optimal alignment score of any alignment of $\mathbf{T}[1, j-1]$ against a string $\mathbf{N} \in NTR(\mathbf{x}, \mathbf{X})$ that ends with a suffix of $\mathbf{X}[1, i]$. When $i = 0$ our assumption is that

$$D^{(1)}[0, j-1] = D^{(2)}[0, j-1] = \max\{D^{(1)}[m, j-1], D^{(2)}[l, j-1]\},$$

so that this common value is the optimal score of an alignment of $\mathbf{T}[1, j-1]$ against a string $\mathbf{N} \in NTR(\mathbf{x}, \mathbf{X})$ ending in either $\mathbf{x}[m]$ or $\mathbf{X}[l]$.

Consider an alignment $(\bar{\mathbf{N}}, \bar{\mathbf{S}})$ of $\mathbf{S} = \mathbf{T}[1, j]$ against a string $\mathbf{N} \in NTR(\mathbf{x}, \mathbf{X})$ ending in $\mathbf{x}[1, i]$ or $\mathbf{X}[1, i]$. We consider three cases, according to the final characters of $\bar{\mathbf{S}}$ and $\bar{\mathbf{N}}$:

1. If $\bar{\mathbf{S}}$ ends in $\mathbf{T}[j]$ and $\bar{\mathbf{N}}$ in $\mathbf{x}[i]$, then deleting these characters gives an alignment of $\mathbf{T}[1, j-1]$ against a string $\mathbf{N}' \in NTR(\mathbf{x}, \mathbf{X})$ ending in $\mathbf{x}[i-1]$ if $i > 1$, or in

either $\mathbf{x}[m]$ or $\mathbf{X}[l]$ if $i = 1$. It follows that

$$\sigma\left(\bar{\mathbf{N}}, \bar{\mathbf{S}}\right) \leq D^{(1)}[i-1, j-1] + \sigma(\mathbf{x}[i], \mathbf{T}[j]),$$

with equality when $\bar{\mathbf{N}}$ and $\bar{\mathbf{S}}$ are obtained by appending $\mathbf{x}[i]$ and $\mathbf{T}[j]$ to an optimal alignment at $D^{(1)}[i-1, j-1]$. A similar argument applies if $\bar{\mathbf{N}}$ ends in $\mathbf{X}[i]$.

2. If $\bar{\mathbf{S}}$ ends in $\mathbf{T}[j]$ and $\bar{\mathbf{N}}$ in a gap, then deleting these characters gives an alignment of $\mathbf{T}[1, j-1]$ against $\mathbf{N}$. If $\mathbf{N}$ ends in $\mathbf{x}[i]$ then

$$\sigma\left(\bar{\mathbf{N}}, \bar{\mathbf{S}}\right) \leq D^{(1)}[i, j-1] + \sigma(-, \mathbf{T}[j]),$$

with equality when $\bar{\mathbf{N}}$ and $\bar{\mathbf{S}}$ are obtained by appending "$-$" and $\mathbf{T}[j]$ to an optimal alignment at $D^{(1)}[i, j-1]$. A similar argument applies if $\mathbf{N}$ ends in $\mathbf{X}[i]$.

3. If $\bar{\mathbf{S}}$ ends in a gap then we may express $\bar{\mathbf{S}}$ in the form

$$\bar{\mathbf{S}} = \bar{\mathbf{S}}'(-)^s,$$

where $s \geq 1$ is as large as possible. Let $\bar{\mathbf{N}} = \bar{\mathbf{N}}'\mathbf{M}$ with $|\mathbf{M}| = s$. Then $\left(\bar{\mathbf{N}}', \bar{\mathbf{S}}'\right)$ is an alignment of one the types considered in cases 1 and 2 above, so

$$\begin{aligned}\sigma\left(\bar{\mathbf{N}}, \bar{\mathbf{S}}\right) &= \sigma\left(\bar{\mathbf{N}}', \bar{\mathbf{S}}'\right) + \sigma(\mathbf{M}, (-)^s) \\ &\leq D^{(k')}[i', j] + \sigma(\mathbf{M}, (-)^s)\end{aligned}$$

for integers $i' \geq 1$ and $k' \in \{1, 2\}$ determined by the tail of $\mathbf{N}'$.

For conciseness let $\mathbf{Y}_1 = \mathbf{x}$ and $\mathbf{Y}_2 = \mathbf{X}$. Then the string $\mathbf{M}$ is an element of $NTR(\mathbf{x}, \mathbf{X})$ of length s ending with $\mathbf{Y}_k[i]$ and beginning with

$$\mathbf{M}[1] = \begin{cases} \mathbf{Y}_{k'}[i'+1] & \text{if } i' < |\mathbf{Y}_{k'}|, \\ \mathbf{x}[1] \text{ or } \mathbf{X}[1] & \text{if } i' = |\mathbf{Y}_{k'}|. \end{cases}$$

So what we must show is that for such strings we have

$$D^{(k)}[i, j] \geq D^{(k')}[i', j] + \sum_{a=1}^{s} \sigma(\mathbf{M}[a], -).$$

By the update rules we have

$$D^{(k'')}[a, j] \geq D^{(k'')}[a-1, j] + \sigma(\mathbf{Y}_{k''}[a], -)$$

for $k'' = 1, 2$ and $a \geq 1$, so the necessary inequality will be true by induction provided we can show that we still have

$$D^{(k'')}[0, j] = \max\{D^{(1)}[m, j], D^{(2)}[l, j]\} \tag{1}$$

after the second update round. This equality follows from the fact that the larger of $D^{(1)}[m, j], D^{(2)}[l, j]$ is unchanged during the second round. Indeed, if the value $D^{(1)}[m, j]$ is changed during the second round then it must have been *increased* to

$$D^{(1)}[m, j] = D^{(1)}[0, j] + \sum_{b=1}^{m} \sigma(\mathbf{x}[b], -),$$

and this is strictly less than $D^{(1)}[0,j]$, because $\sigma(\alpha,-) < 0$ for all α. A similar argument applies to $D^{(2)}[l,j]$, so the larger of these is unchanged and remains the maximum.

By the above we have $\sigma(\bar{\mathbf{N}},\bar{\mathbf{S}}) \leq D^{(k)}[i,j]$. It remains to show that there is in fact an alignment with score $D^{(k)}[i,j]$ when $D^{(k)}[i,j] = D^{(k)}[i-1,j] + \sigma(\mathbf{Y}_k[i],-)$. Consider the trace back procedure beginning from $D^{(k)}[i-1,j]$. This must eventually reach an (i',j)-entry of either $D^{(1)}$ or $D^{(2)}$ that derives from column $j-1$ (since for example the largest entry in each column of each matrix must be derived this way), and we obtain the desired alignment by appending suitable strings to an optimal alignment at this point.

Cases 1–3 above show that $D^{(1)}[i,j]$ and $D^{(2)}[i,j]$ have been correctly computed for $i \geq 1$, and equation (11) shows that $D^{(1)}[0,j]$ and $D^{(2)}[0,j]$ have been too. It follows by induction that both matrices $D^{(1)}$ and $D^{(2)}$ have been correctly computed.

4.4 Extension to Nested Tandem Repeats with Three or More Motifs

Our algorithm is easily adapted to the motif alignment problem for more complex NTRs built from three or more motifs $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_r$. Let $|\mathbf{X}_k| = m_k$ for each k, and again let $|\mathbf{T}| = n$, where $\mathbf{T}$ is the text containing the NTR. For each $k = 1,\ldots,r$ we introduce an $(m_k+1) \times (n+1)$ matrix $D^{(k)}$, and we initialise these as in Section 4.2. After the jth column of each matrix has been filled as in the first round above we update $D^{(k)}[0,j]$ with $\max\{D^{(i)}[m_i,j] | i = 1,\ldots,r\}$ for each k, and then run a second round as above to update the jth column of each matrix. Once the matrices have been filled, an optimal alignment may then be found using the standard trace-back procedure. The time and space complexity for the k-motif alignment algorithm is $O(T(|m_1| + |m_2|..[m_r])$ as it takes $2 \times m_k$ to fill each matrix $D^{(k)}$. In the case where the motifs have the same length then the complexity would be $O(|T| * (2 * k * |m|))$.

5 Conclusion

In this paper, we presented an algorithm to solve the problem of the alignment of nested tandem repeats. This algorithm has $O(|\mathbf{T}|(|\mathbf{x}| + |\mathbf{X}|))$ time complexity. The nested WDP alignment is incorporated in the program NTRFinder [6] we are developing, as part of the verification phase. It has been tested on both simulated and real data and has proved to be very effective.

References

1. Benson, G.: Tandem repeats finder: a program to analyze DNA sequences. Nucl. Acids Res. 27(2), 573–580 (1999)
2. Domaniç, N.O., Preparata, F.P.: A novel approach to the detection of genomic approximate tandem repeats in the levenshtein metric. Journal of Computational Biology 14(7), 873–891 (2007)

3. Fischetti, V.A., Landau, G.M., Sellers, P.H., Schmidt, J.P.: Identifying periodic occurrences of a template with applications to protein structure. Information Processing Letters 45, 11–18 (1993)
4. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, Cambridge (1997)
5. Hauth, A.M., Joseph, D.: Beyond tandem repeats: complex pattern structures and distant regions of similarity. In: ISMB, pp. 31–37 (2002)
6. Matroud, A.A., Hendy, M.D., Tuffley, C.P.: NTRFinder: An Algorithm to Find Nested Tandem Repeats (2010), http://awcmee.massey.ac.nz/pdf_files/AtheerNTR_Apr.pdf
7. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys 33, 2001 (1999)
8. 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)
9. Newman, A., Cooper, J.: Xstream: A practical algorithm for identification and architecture modeling of tandem repeats in protein sequences. BMC Bioinformatics 8(1), 382 (2007)
10. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
11. Wexler, Y., Yakhini, Z., Kashi, Y., Geiger, D.: Finding approximate tandem repeats in genomic sequences. Journal of Computational Biology 12(7), 928–942 (2005); PMID: 16201913

Limited Lifespan of Fragile Regions in Mammalian Evolution

Max A. Alekseyev[1] and Pavel A. Pevzner[2]

[1] University of South Carolina, Columbia, SC, U.S.A.
[2] University of California, San Diego, CA, U.S.A.

Abstract. An important question in genome evolution is whether there exist fragile regions (*rearrangement hotspots*) where chromosomal rearrangements are happening over and over again. Although nearly all recent studies supported the existence of fragile regions in mammalian genomes, the most comprehensive phylogenomic study of mammals (Ma et al. (2006) *Genome Research* **16**, 1557-1565) raised some doubts about their existence. We demonstrate that fragile regions are subject to a "birth and death" process, implying that fragility has limited evolutionary lifespan. This finding implies that fragile regions migrate to different locations in different mammals, explaining why there exist only a few chromosomal breakpoints shared between different lineages. The birth and death of fragile regions phenomenon reinforces the hypothesis that rearrangements are promoted by matching segmental duplications and suggests putative locations of the currently active fragile regions in the human genome.

1 Introduction

In 1970 Susumu Ohno [35] came up with the Random Breakage Model (RBM) of chromosome evolution, implying that there are no rearrangement hotspots in mammalian genomes. In 1984 Nadeau and Taylor [34] laid the statistical foundations of RBM and demonstrated that it was consistent with the human and mouse chromosomal architectures. In the next two decades, numerous studies with progressively increasing resolution made RBM the *de facto* theory of chromosome evolution.

RBM was refuted by Pevzner and Tesler, 2003 [38] who suggested the Fragile Breakage Model (FBM) postulating that mammalian genomes are mosaics of fragile and solid regions. In contrast to RBM, FBM postulates that rearrangements are mainly happening in fragile regions forming only a small portion of the mammalian genomes. While the rebuttal of RBM caused a controversy [7, 43, 44], Peng et al., 2006 [36] and Alekseyev and Pevzner, 2007 [2] revealed some flaws in the arguments against FBM. Furthermore, the rebuttal of RBM was followed by many studies supporting FBM [6, 8, 9, 12, 14, 17, 19, 20, 21, 24, 27, 28, 29, 30, 31, 39, 40, 41, 46, 47, 49, 51].

Comparative analysis of the human chromosomes reveals many short adjacent regions corresponding to parts of several mouse chromosomes [16]. While such

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 198–215, 2010.

a surprising arrangement of synteny blocks points to potential rearrangement hotspots, it remains unclear whether these regions reflect genome rearrangements or duplications/assembly errors/alignment artifacts. Early studies of genomic architectures were unable to distinguish short synteny blocks from artifacts and thus were limited to constructing large synteny blocks. Ma et al., 2006 [25] addressed the challenge of constructing high-resolution synteny blocks via the analysis of multiple genomes. Remarkably, their analysis suggests that there is limited breakpoint reuse, an argument against FBM, that led to a split among researchers studying chromosome evolution and raised a challenge of reconciling these contradictory results. Ma et al., 2006 [25] wrote: *"a careful analysis* [of the RBM vs FBM controversy] *is beyond the scope of this study"* leaving the question of interpreting their findings open.

Various models of chromosome evolution imply various statistics and thus can be verified by various tests. For example, RBM implies exponential distribution of the synteny block sizes, consistent with the human-mouse synteny blocks observed in [34]. Pevzner and Tesler, 2003 [38] introduced the "pairwise breakpoint reuse" test and demonstrated that while RBM implies low breakpoint reuse, the human-mouse synteny blocks expose rampant breakpoint reuse. Thus RBM is consistent with the "exponential length distribution" test [34] but inconsistent with the "pairwise breakpoint reuse" test [37]. Both these tests are applied to *pairs* of genomes, not taking an advantage of multiple genomes that were recently sequenced. Below we introduce the "multispecies breakpoint reuse" test and demonstrate that both RBM and FBM do not pass this test. We further propose the *Turnover Fragile Breakage Model (TFBM)* that extends FBM and complies with the multispecies breakpoint reuse test.

Technically, findings in [25] (limited breakpoint reuse between different lineages) are not in conflict with findings in [38] (rampant breakpoint reuse in chromosome evolution). Indeed, Ma et al., 2006 [25] only considered *inter-reuse* between different branches of the phylogenetic tree and did not analyze *intra-reuse* within individual branches of the tree. TFBM reconciles the recent studies supporting FBM with the Ma et al., 2006 [25] analysis. We demonstrate that data in [25] reveal rampant but elusive breakpoint reuse that cannot be detected via counting repeated breakages between various pairs of branches of the evolutionary tree. TFBM is an extension of FBM that reconciles seemingly contradictory results in [6, 8, 9, 12, 14, 17, 19, 20, 21, 24, 27, 28, 29, 30, 31, 39, 40, 41, 46, 47, 49, 51] and [25] and explains that they do not contradict to each other. TFBM postulates that fragile regions have a limited lifespan and implies that they can migrate between different genomic locations. The intriguing implication of TFBM is that few regions in a genome are fragile at any given time raising a question of finding the currently active fragile regions in the human genome.

While many authors have discussed the causes of fragility, the question what makes certain regions fragile remains open. Previous studies attributed fragile regions to segmental duplications [5, 18, 42, 50], high repeat density [32], high recombination rate [33], pairs of tRNA genes [10, 23], inhomogeneity of gene distribution [36], and long regulatory regions [17, 30, 36]. Since we observed

the birth and death of fragile regions, we are particularly interested in features that are also subject to birth and death process. Recently, Zhao and Bourque, 2009 [50] provided a new insight into association of rearrangements with segmental duplications by demonstrating that many rearrangements are flanked by *Matching Segmental Duplications* (MSDs), i.e., a pair of long similar regions located within a pair of breakpoint regions corresponding to a rearrangement event. MSDs arguably represent an ideal match for TFBM among the features that were previously implicated in breakpoint reuses. TFBM is consistent with the hypothesis that MSDs promote fragility since the similarity between MSDs deteriorates with time, implying that MSDs are also subjects to a "birth and death" process.

2 Results

2.1 Rearrangements and Breakpoint Graphs

For the sake of simplicity, we start our analysis with *circular genomes* consisting of circular chromosomes. While we use circular chromosomes to simplify the computational concepts discussed in the paper, all analysis is done with real (linear) mammalian chromosomes (see Alekseyev, 2008 [1] for subtle differences between circular and linear chromosome analysis). We represent a circular chromosome with synteny blocks $x_1, \ldots, x_n$ as a cycle (Fig. 1) composed of n directed labelled edges (corresponding to the blocks) and n undirected unlabeled edges (connecting adjacent blocks). The directions of the edges correspond to *signs* (strands) of the blocks. We label the *tail* and *head* of a directed edge x_i as x_i^t and x_i^h respectively. We represent a genome as a *genome graph* consisting of disjoint cycles (one for each chromosomes). The edges in each cycle alternate between two colors: one color reserved for undirected edges and the other color (traditionally called "obverse") reserved for directed edges.

Let P be a genome represented as a collection of *alternating* black-obverse cycles (a cycle is alternating if the colors of its edges alternate). For any two black edges (u, v) and (x, y) in the genome (graph) P we define a *2-break* rearrangement (see [3]) as replacement of these edges with either a pair of edges (u, x), (v, y), or a pair of edges (u, y), (v, x) (Fig. 2). 2-breaks extend the standard operations of reversals (Fig. 2a), fissions (Fig. 2b), or fusions/translocations (Fig. 2c) to the case of circular chromosomes. We say that a 2-break on edges (u, x), (v, y) *uses* vertices u, x, v and y.

Let P and Q be "black" and "red" genomes on the same set of synteny blocks $\mathcal{X}$. The *breakpoint graph* $G(P, Q)$ is defined on the set of vertices $V = \{x^t, x^h \mid x \in \mathcal{X}\}$ with black and red edges inherited from genomes P and Q (Fig. 1). The black and red edges form a collection of alternating *black-red cycles* in $G(P, Q)$ and play an important role in analyzing rearrangements (see [11] for background information on genome rearrangements). The *trivial cycles* in $G(P, Q)$, formed by pairs of parallel black and red edges, represent common adjacencies between synteny blocks in genomes P and Q. Vertices of the non-trivial cycles in $G(P, Q)$ represent *breakpoints* that partition genomes P and Q into (P, Q)-synteny blocks.

The 2-*break distance* $d(P, Q)$ between circular genomes P and Q is defined as the minimum number of 2-breaks required to transform one genome into the other. In contrast to the genomic distance [13] (for linear genomes), the 2-break distance for circular genomes is easy to compute [48]:

Theorem 1. *The 2-break distance between circular genomes P and Q is $b(P, Q) - c(P, Q)$ where $b(P, Q)$ and $c(P, Q)$ are respectively the number of (P, Q)-synteny blocks and non-trivial black-red cycles in $G(P, Q)$.*

2.2 Inter- and Intra- Breakpoint Reuse

Figure 3 shows a phylogenetic tree with specified rearrangements on its branches (we write $\rho \in e$ to refer to a 2-break ρ on an edge e). We represent each genome as a genome graph (i.e., a collection of cycles) on the same set V of $2n$ vertices (corresponding to the endpoints of the synteny blocks). Given a set of genomes and a phylogenetic tree describing rearrangements between these genomes, we define the notions of inter- and intra-breakpoint reuses. A vertex $v \in V$ is *inter-reused* on two distinct branches e_1 and e_2 of a phylogenetic tree if there exist 2-breaks $\rho_1 \in e_1$ and $\rho_2 \in e_2$ that both use v. Similarly, a vertex $v \in V$ is *intra-reused* on a branch e if there exist two distinct 2-breaks $\rho_1, \rho_2 \in e$ that both use v. For example, a vertex c^h is inter-reused on the branches (Q_3, P_1) and (Q_2, P_3), while a vertex f^h is intra-reused on the branch (Q_3, Q_2) of the tree in Fig. 3. We define $br(e_1, e_2)$ as the number of vertices inter-reused on the branches e_1 and e_2, and $br(e)$ as the number of vertices intra-reused on the branch e. An alternative approach to measuring breakpoint intra-reuse is to define *weighted intra-reuse* of a vertex v on a branch e as $\max\{0, use(e, v) - 1\}$ where $use(e, v)$ is the number of 2-breaks on e using v. The weighted intra-reuse $BR(e)$ on the branch e is the sum of weighted intra-reuse of all vertices.[1]

Given simulated data, one can compute $br(e)$ for all branches and $br(e_1, e_2)$ for all pairs of branches in the phylogenetic tree. However, for real data, rearrangements along the branches are unknown, calling for alternative ways for estimating the inter- and intra-reuse.

Cycles in the breakpoint graphs provide yet another way to estimate the inter- and intra-reuse. For a branch $e = (P, Q)$ of the phylogenetic tree, one can estimate $br(e)$ by comparing the reversal distance $d(P, Q)$ and the number of breakpoints $b(P, Q)$ between the genomes P and Q. It results in the lower bound $bound(e) = 4 \cdot d(P, Q) - 2 \cdot b(P, Q)$ for $BR(e)$ [37] that also gives a good approximation for $br(e)$. On the other hand, one can estimate $br(e_1, e_2)$ as the number $bound(e_1, e_2)$ of vertices shared between non-trivial cycles in the breakpoint graphs corresponding to the branches e_1 and e_2 (similar approach was used in [22] and later explored in [25, 31]). Assuming that the genomes at the internal nodes of the phylogenetic tree can be reliably reconstructed [4, 25, 26, 45], one can compute $bound(e)$ and $bound(e_1, e_2)$ for all (pairs of) branches. Below we show that these bounds accurately approximate the intra- and inter-reuse.

[1] We remark that if no vertex is used more than twice on a branch e then $BR(e) = br(e)$.

2.3 Analyzing Breakpoint Reuse (Simulated Genomes)

We start from analyzing simulated data based on FBM with n fragile regions present in k genomes that evolved according to a certain phylogenetic tree (for the varying parameter n). We represent one of the leaf genomes as the genome with 20 random circular chromosomes and simulate hundred 2-breaks on each branch of the tree.

Figure 4 represents a phylogenetic tree on five leaf genomes, denoted M, R, D, Q, H, and three ancestral genomes, denoted MR, MRD, QH. Table 1(left panel) presents the results of a single FBM simulation and illustrates that $bound(e_1, e_2)$ provides an excellent approximation for inter-reuses $br(e_1, e_2)$ for all 21 pairs of branches.[2] While $bound(e)$ (on the diagonal of Table 1, left panel) is somewhat less accurate, it also provides a reasonable approximation for $br(e)$.

Below we describe analytical approximations for the values in Table 1(left panel). Since every 2-break uses 4 out of $2n$ vertices in the genome graph, a random 2-break uses a vertex v with the probability $\frac{2}{n}$. Thus, a sequence of t random 2-breaks does not use a vertex v with the probability $(1-\frac{2}{n})^t \approx e^{-\frac{2t}{n}}$ (for $t \ll n$). For branches e_1 and e_2 with respectively t_1 and t_2 random 2-breaks, the probability that a particular vertex is inter-reused on e_1 and e_2 is approximated as $(1-e^{-\frac{2t_1}{n}})\cdot(1-e^{-\frac{2t_2}{n}})$. Therefore, the expected number of inter-reused vertices is approximated as $2n\cdot(1-e^{-\frac{2t_1}{n}})\cdot(1-e^{-\frac{2t_2}{n}})$. Below we will compare the observed inter-reuse with the expected inter-reuse in FBM to see whether they are similar thus checking whether FBM represents a reasonable null hypothesis. We will use the term *scaled inter-reuse* to refer to the observed inter-reuse divided by the expected inter-reuse. If FBM is an adequate null hypothesis we expect the scaled inter-reuse to be close to 1.

Similarly, a sequence of t random 2-breaks uses a vertex v exactly once with the probability $t \cdot \frac{2}{n} \cdot \left(1-\frac{2}{n}\right)^{t-1} \approx \frac{2t}{n} e^{\frac{2(t-1)}{n}}$. Therefore, the probability of a particular vertex being intra-reused on a branch with t random 2-breaks is approximately $1 - e^{-\frac{2t}{n}} - \frac{2t}{n} e^{\frac{2(t-1)}{n}}$, implying that the expected intra-reuse is approximately $2n \cdot \left(1 - e^{-\frac{2t}{n}} - \frac{2t}{n} e^{\frac{2(t-1)}{n}}\right)$. We will use the term *scaled intra-reuse* to refer to the observed intra-reuse divided by the expected intra-reuse. Our simulations showed that the scaled intra- and inter-reuse for 21 pairs of branches are all close to 1 (data are not shown).

We also performed a similar simulation, this time varying the number of 2-breaks on the branches according to the branch lengths specified in Fig. 4. Again, the lower bounds provide accurate approximations in the case of varying branch lengths. Similar results were obtained in the case of evolutionary trees with varying topologies (data are not shown). We therefore use only lower bounds to generate Table 1(right panel) rather than showing both real distances and the lower bounds as in Table 1(left panel).

[2] We remark that $bound(e_1, e_2) = br(e_1, e_2)$ if simulations produce the shortest rearrangement scenarios on the branches e_1 and e_2. Table 1(left panel) illustrates that this is mainly the case for our simulations.

In the case when the branch lengths vary, we find it convenient to represent data as a plot that illustrates variability in the scaled inter-use. We define the *distance* between branches e_1 and e_2 in the phylogenetic tree as the distance between their midpoints, i.e., the overall length of the path, starting at e_1 and ending at e_2, minus $\frac{d(e_1)+d(e_2)}{2}$. For example, $d(M+, H+) = 56+170+58+28-\frac{56+28}{2} = 270$ (see Fig. 4). The x-axis in Fig. 5 represents the distances between pairs of branches (21 pairs total), while y-axis represents the scaled inter-reuse for pairs of branches at the distance x.

2.4 Surprising Irregularities in Breakpoint Reuse in Mammalian Genomes

The branch lengths shown in Fig. 4 actually represent the approximate numbers of rearrangements on the branches of the phylogenetic tree for *M*ouse, *R*at, *D*og, maca*Q*ue, and *H*uman genomes (represented in the alphabet of 433 "large" synteny blocks exceeding 500,000 nucleotides in human genome [4]). For the mammalian genomes, M, R, D, Q, and H, we first used MGRA [4] to reconstruct genomes of their common ancestors (denoted MR, MRD, and QH in Fig. 4) and further estimated the breakpoint inter-reuse between pairs of branches of the phylogenetic tree. The resulting Table 2 reveals some striking differences from the simulated data (Table 1, right panel) that follow a peculiar pattern: the larger is the distance between two branches, the smaller is the amount of inter-reuse between them (in contrast to RBM/FBM where the amount of inter-reuse does not depend on the distance between branches). The statement above is imprecise since we haven't described yet how to compare the amount of inter-reuse for different branches at various distances. However, we can already illustrate this phenomenon by considering branches of similar length that presumably influence the inter-reuse in a similar way (see below).

We notice that branches $M+$, $R+$, and $QH+$ have similar lengths (varying from 56 to 68 rearrangements) and construct subtables of Table 1(right panel) (for $n = 900$) and Table 2 with only three rows corresponding to these branches (Table 3). Since the lengths of branches $M+$, $R+$, and $QH+$ are similar, FBM implies that the elements belonging to the same columns in Table 3 should be similar. This is indeed the case for simulated data (small variations within each column) but not the case for real data. In fact, maximal elements in each column for real data exceed other elements by a factor of 3-5 (with an exception of the $MR+$ column). Moreover, the peculiar pattern associated with these maximal elements (maximal elements correspond to red cells) suggests that this effect is unlikely to be caused by random variations in breakpoint reuses. We remind the reader that red cells correspond to pairs of adjacent branches in the evolutionary tree suggesting that breakpoint reuse is maximal between close branches and is reducing with evolutionary time. A similar pattern is observed for the other pairs of branches of similar length: adjacent branches feature much higher inter-reuse than distant branches. We also remark that the most distant pairs of branches ($H+$ and $M+$, $H+$ and $R+$, $Q+$ and $M+$, $Q+$ and $R+$ in the yellow cells) feature the lowest inter-reuse. The only branch that shows relatively similar

inter-reuse (varying from 58 to 80) with the branches $M+$, $R+$, and $QH+$ is the branch $MR+$ which is adjacent to each of these branches.

Below we modify FBM to come up with a new model of chromosome evolution, explaining the surprising irregularities in the inter-reuse across mammalian genomes.

2.5 Turnover Fragile Breakage Model: Birth and Death of Fragile Regions

We start with a simulation of 100 rearrangements on every branch of the tree in Fig. 4. However, instead of assuming that fragile regions are fixed, we assume that after every rearrangement x fragile regions "die" and x fragile regions are "born" (keeping a constant number of fragile regions throughout the simulation). We assume that the genome has m potentially "breakable" sites but only n of them are currently fragile ($n \leq m$) (the remaining $n - m$ sites are currently solid). The dying regions are randomly selected from n currently fragile regions, while the newly born regions are randomly selected from $m - n$ solid regions.

The simplest TFBM with a fixed rate of the "birth and death" process is defined by the parameters m, n, and *turnover* rate x (FBM is a particular case of TFBM corresponding to $x = 0$, while RBM is a particular case of TFBM corresponding to $x = 0$ and $n = m$). While this over-simplistic model with a fixed turnover rate may not adequately describe the real rearrangement process, it allows one to analyze the general trends and to compare them to the trends observed in real data.

The leftmost subtable of Table 4 with $x = 0$ represents an equivalent of Table 1(left panel) for FBM and reveals that the inter-reuse is roughly the same on all pairs of branches (≈ 110 for $n = 500$, ≈ 70 for $n = 900$, ≈ 50 for $n = 1300$). The right subtables of Table 4 represent equivalents of the leftmost subtable for TFBM with the turnover rate $x = 1, 2, 3$ and reveal that the inter-reuse in yellow cells is lower than in green cells, while the inter-reuse in green cells is lower than in red cells.

Fig. 6 shows the scaled inter-reuse averaged over yellow, green, and red cells that reveals a different behavior between FBM and TFBM. Indeed, while the scaled inter-reuse is close to 1 for all pairs of branches in the case of FBM, it varies in the case of TFBM. For example, for $n = 900, m = 2000$, and $x = 3$, the inter-reuse in yellow cells is ≈ 40, in green cells is ≈ 45, and in red cells is ≈ 56. In the following sections we describe an accurate formula for estimating the breakpoint inter-reuse in the case of TFBM that accurately approximates the values shown in Fig. 6.

Our simulations demonstrate that the distribution of inter-reuses among green, red, and yellow cells differs between FBM and TFBM. We argue that this distribution (e.g., the slope of the curve in Fig. 6) represents yet another test to confirm or reject FBM/TFBM. However, while it is clear how to apply this test to the simulated data (with known rearrangements), it remains unclear how to compute it for real data when the ancestral genomes (as well as the parameters of the model) are unknown. While the ancestral genomes can be reliably approximated

using the algorithms for ancestral genome reconstruction [4, 25, 26, 45], estimating the number of fragile regions remains an open problem (see [38]). Below we develop a new test (that does not require knowledge of the number of the fragile regions n) and demonstrate that FBM does not pass this test while TFBM does, explaining the surprisingly low inter-reuse in mammalian genomes.

2.6 Multispecies Breakpoint Reuse Test

Given a phylogenetic tree describing a rearrangement scenario, we define the multispecies breakpoint reuse on this tree as follows. For two rearrangements ρ_1 and ρ_2 in the scenario, we define the distance $d(\rho_1, \rho_2)$ as the number of rearrangements in the scenario between ρ_1 and ρ_2 plus 1. For example, the distance between 2-breaks r_4 and r_6 in the tree in Fig. 3 is 4. We define the (actual) multispecies breakpoint reuse as a function

$$R(\ell) = \frac{\sum_{\rho_1,\rho_2\,:\,d(\rho_1,\rho_2)=\ell} br(\rho_1,\rho_2)}{\sum_{\rho_1,\rho_2\,:\,d(\rho_1,\rho_2)=\ell} 1}$$

that represents the total breakpoint reuse between pairs of rearrangements ρ_1, ρ_2 at the distance ℓ divided by the number of such pairs. Here $br(\rho_1, \rho_2)$ stands for the number of vertices used by both 2-breaks ρ_1 and ρ_2.

Since the rearrangements on branches of the phylogenetic tree are unknown, we use the following sampling procedure to approximate $R(\ell)$. Given genomes P and Q, we sample various shortest rearrangement scenarios between these genomes by generating random 2-break transformations of P into Q. To generate a random transformation we first randomly select a non-trivial cycle C in the breakpoint graph $G(P, Q)$ with the probability proportional to $|C|/2-1$, i.e., the number of 2-breaks required to transform such a cycle into a collection of trivial cycles ($|C|$ stands for the length of C). Then we uniformly randomly select a 2-break ρ from the set of all $\binom{|C|/2}{2} = \frac{|C|(|C|-2)}{8}$ 2-breaks that splits the selected cycle C into two and thus by Theorem 1 decreases the distance between P and Q by 1 (i.e., $d(\rho P, Q) = d(P, Q) - 1$). We continue selecting non-trivial cycles and 2-breaks in an iterative fashion for genomes $\rho \cdot P$ and Q and so on until P is transformed into Q.

The described sampling can be performed for every branch $e = (P, Q)$ of the phylogenetic tree, essentially partitioning e into $length(e) = d(P, Q)$ sub-branches, each featuring a single 2-break. The resulting tree will have $\sum_e length(e)$ sub-branches, where the sum is taken over all branches e.

For each pair of sub-branches, we compute the number of reused vertices across them and accumulate these numbers according to the distance between these sub-branches in the tree. The *empirical multispecies breakpoint reuse* (the average reuse between all sub-branches at the distance ℓ) is defined as the actual multispecies breakpoint reuse in a sampled rearrangement scenario. Our tests on phylogenetic trees with varying topologies demonstrated a good fit between the actual, empirical, and theoretical $R(l)$ curves (data are not shown).

For the five mammalian genomes, the plot of $R(\ell)$ is shown in Fig. 7. From this empirical curve we estimated the parameters $n \approx 196$, $x \approx 1.12$, and $m \approx 4017$

(see the next section) and displayed the corresponding theoretical curve. We remark that the estimated parameter n in TFBM is expected to be larger than the observed number of synteny blocks (since not all potentially breakable regions were broken in a given evolutionary scenario).

We argue that the empirical multispecies breakpoint reuse curve $R(\ell)$ complements the "exponential length distribution" [34] and "pairwise breakpoint reuse" [38] tests as the 3rd criterion to accept/reject RBM, FBM, and now TFBM. One can use the parameters n and x (estimated from empirical $R(\ell)$ curve) to evaluate the extent of the "birth and death" process and to explain why Ma et al., 2006 [25] found so few shared breakpoints between different mammalian lineages. In practice, the "multispecies breakpoint reuse test" can be applied in the same way as the Nadeau-Taylor "exponential length distribution test" was applied in numerous papers. The Nadeau-Taylor test typically amounted to constructing a histogram of synteny blocks and evaluating (often visually) whether it fits the exponential distribution. Similarly, the "multispecies breakpoint reuse test" amounts to constructing $R(\ell)$ curve and evaluating whether it significantly deviates from a horizontal line suggested by RBM and FBM. The estimated parameters of the TFBM model (see the next section) can be used to quantify the extent of these deviations.

TFBM also raises an intriguing question of what triggers the birth and death of fragile regions. As demonstrated by Zhao and Bourque, 2009 [50], the disproportionately large number of rearrangements in primate lineages are flanked by MSDs. TFBM is consistent with the Zhao-Bourque hypothesis that rearrangements are triggered by MSDs since MSDs are also subject to the "birth and death" process. Indeed, after a segmental duplication the pair of matching segments becomes subjected to random mutations and the similarity between these segments dissolves with time (a pair of segmental duplications "disappears" after ≈ 40 million years of evolution if one adopts the parameters for defining segmental duplications from [15]). The mosaic structure of segmental duplications [15] provides an additional explanation of how MSDs may promote breakpoint reuses and generate long cycles typical for the breakpoint graphs of mammalian genomes.

2.7 Computing Multispecies Breakpoint Reuse in the TFBM Model

Let *Fragile* and *Solid* be the sets of n initial fragile regions and $m-n$ initial solid regions respectively. In TFBM, the sets *Fragile* and *Solid* change in accordance with the turnover rate x, i.e., after every 2-break x randomly chosen regions (corresponding to $2x$ vertices in the breakpoint graph) from *Fragile* are moved to *Solid*, and vice versa. For a vertex in the set *Fragile*, we evaluate the probability $P(\ell)$ that this vertex still belongs to *Fragile* after ℓ 2-breaks. After every 2-break, a vertex from *Fragile* moves to *Solid* with the probability $\frac{x}{n}$, while a vertex from *Solid* moves to *Fragile* with the probability $\frac{x}{m-n}$. Therefore,

$$P(\ell+1) = P(\ell)\cdot\left(1-\frac{x}{n}\right)+(1-P(\ell))\cdot\frac{x}{m-n} = \left(1-\frac{xm}{n(m-n)}\right)\cdot P(\ell)+\frac{x}{m-n}$$

with $P(0) = 1$. Solution to this recurrence is $P(\ell) = \frac{m-n}{m}\left(1 - \frac{xm}{n(m-n)}\right)^{\ell} + \frac{n}{m}$. We now compute the expected reuse between 2-breaks ρ_1 and ρ_2 separated by ℓ other 2-breaks. Since every 2-break uses 4 vertices, the probability that it uses a particular vertex in *Fragile* is $\frac{2}{n}$. Since the 2-break ρ_1 used 4 vertices, the expected reuse between ρ_1 and ρ_2 is:

$$R(\ell) = 4 \cdot \frac{2}{n} \cdot P(\ell) = \frac{8 \cdot (m-n)}{n \cdot m}\left(1 - \frac{xm}{n(m-n)}\right)^{\ell} + \frac{8}{m}.$$

This formula fits the simulated data well, thus opening a possibility to determine the parameters m, n, and x for given real genomes. In particular, n and x can be determined from the value and slope of $R(\ell)$ at $\ell = 0$, since $R(0) = \frac{8}{n}$ and $R'(0) \approx -\frac{8x}{n^2}$ (assuming $\frac{xm}{n(m-n)} \ll 1$).

2.8 Fragile Regions in the Human Genome

Let us imagine the following gedanken experiment: 25 million years ago (time of the human-macaque split) a scientist sequences the genome of the human-macaque ancestor (QH) and attempts to predict the sites of (future) rearrangements in the (future) human genome. The only other information the scientist has is the mouse, rat, and dog genomes. While RBM offers no clues on how to make such a prediction, FBM suggests that the scientist should use the breakpoints between one of the available genomes and QH as a proxy for fragile regions. For example, there are 552 breakpoints between the mouse genome (M) and QH and 34 of them were actually used in the human lineage, resulting in only $34/552 \approx 6\%$ accuracy in predicting future human breakpoints (we use synteny blocks larger than 500K from [4]).

TFBM suggests that the scientist should rather use the *closest* genome to QH to better predict the human breakpoints. That can be achieved by first reconstructing the common ancestor (MRD) of mouse, rat, dog, and human-macaque ancestor and then using the breakpoints between MRD and QH as a proxy for the sites of rearrangements in the human lineage. 18 out 162 breakpoints between MRD and QH were used in the human lineage, resulting in $18/162 \approx 11\%$ accurate prediction of human breakpoints, nearly doubling the accuracy of predictions from distant genomes.

Now let us imagine that the scientist somehow gained access to the extant macaque genome. There are 68 breakpoints between Q and QH and 10 of them were used in the human lineage, resulting in $10/68 \approx 16\%$ accurate prediction of human breakpoints, again improving the accuracy of predictions.

These estimates indicate that TFBM can be used to improve the prediction accuracy of *future* rearrangements in various lineages and demonstrate that the sites of *recent* rearrangements in the human and other primate lineages represent the best guess for the currently active fragile regions in the human genome.

We therefore focus on the incident branches $H+$, $Q+$, and $QH+$ and construct the breakpoint graphs $G(H, QH)$, $G(Q, QH)$, and $G(QH, MRD)$. We further superimposed these three graphs to find out breakpoints that were inter-reused on the branches $H+$, $Q+$, and $QH+$. Figure 8 shows the positions of these

recently affected breakpoints (projected to the human genome) that, according to TFBM, represent the best proxy for the currently active fragile regions in the human genome. Various ongoing primate genome sequencing projects will soon result in an even better estimate for the fragile regions in the human genome.

3 Discussion

Since every species on Earth (including *Homo sapiens*) may speciate into multiple new species, one can ask a question: "How will the human genome evolve in the *next* million years?" TFBM suggests the putative sites of *future* rearrangements in the human genome. The answer to the question "Where are the (future) fragile regions in the human genome?" may be surprisingly simple: they are likely to be among the breakpoint regions that were used in various primate lineages.

Nadeau and Taylor, 1984 [34] proposed RBM based on a single observation: the exponential distribution of the human-mouse synteny block sizes. There is no doubt that jumping to this conclusion was not fully justified: there are many other models (e.g., FBM) that lead to the same exponential distribution of the "visible" synteny block sizes. Currently, there is no single piece of evidence that would allow one to claim that RBM is correct and FBM is not.

While Pevzner and Tesler, 2003 [38] revealed large breakpoint reuse (supporting FBM and contradicting RBM), Ma et al., 2006 [25] discovered low breakpoint inter-reuse (contradicting FBM), calling for yet another generalization of FBM. The proposed TFBM model not only passes both "exponential length distribution" test (motivation for RBM) and "pairwise breakpoint reuse" test (motivation for FBM) but also explains the puzzling discovery of limited breakpoint inter-reuse in [25]. We therefore argue that TFBM is a more accurate model of chromosome evolution, allowing one to approximate the currently active fragile regions in the human genome.

Needless to say, TFBM, similarly to RBM and FBM (or various models of point mutations, e.g., Jukes-Cantor model), is a simplistic model of chromosome evolution that is only an approximation of the real evolutionary process. Moreover, in the current paper we considered TFBM only for the case of 2-breaks and did not include other rearrangements such as transpositions. However, it is fair to assume that transpositions are as likely to happen on incident branches as on distant branches, implying that they cannot possibly cause the reduced breakpoint inter-reuse on distant branches. In addition to limitations of TFBM as a model, there exists a concern whether computation of empirical multispecies breakpoint reuse (that requires reconstruction of ancestral genomes) may be affected by errors in reconstruction of ancestral genomes. While various tools for ancestral genome reconstruction (such as MGRA [4] and inferCARs [25]) were shown to be quite accurate (in particular, they produce nearly identical results while using very different algorithms), it is a challenging open problem to evaluate the multispecies breakpoint reuse without explicitly computing ancestral genomes.

The key point of this paper is the birth and death process of fragile regions rather than a specific model aimed at estimating the hidden parameters of this process. TFBM is merely an initial and over-simplistic attempt to estimate these parameters. The parameters predicted by TFBM (e.g., the number of active fragile regions) are currently difficult to superimpose with scarce information about rearrangements in only 7 reliably completed mammalian genomes, not unlike the parameters of RBM derived in 1984 when no high-resolution comparative mammalian genomic architectures were available. However, similarly to comparative mapping efforts in early 1990s that confirmed the Nadeau-Taylor estimates, we believe that imminent sequencing of over 400 primate species will soon provide the detailed information about chromosomal fragility in human genome and will allow one to verify the TFBM parameters.

Similarly to the discovery of breakpoint reuse in 2003 [38], there is currently only indirect evidence supporting the birth and death of fragile regions in chromosome evolution. However, we hope that, similarly to FBM (that led to many follow-up studies supporting the existence of fragile regions), TFBM will trigger further investigations of the fragile regions longevity.

Acknowledgements

The authors thank Glenn Tesler and Jian Ma for many helpful comments.

References

[1] Alekseyev, M.A.: Multi-Break Rearrangements and Breakpoint Re-uses: from Circular to Linear Genomes. Journal of Computational Biology 15(8), 1117–1131 (2008)
[2] Alekseyev, M.A., Pevzner, P.A.: Are There Rearrangement Hotspots in the Human Genome? PLoS Computational Biology 3(11), e209 (2007)
[3] Alekseyev, M.A., Pevzner, P.A.: Multi-Break Rearrangements and Chromosomal Evolution. Theoretical Computer Science 395(2-3), 193–202 (2008)
[4] Alekseyev, M.A., Pevzner, P.A.: Breakpoint Graphs and Ancestral Genome Reconstructions. Genome Research 19(5), 943–957 (2009)
[5] Armengol, L., Pujana, M.A., Cheung, J., Scherer, S.W., Estivill, X.: Enrichment of segmental duplications in regions of breaks of synteny between the human and mouse genomes suggest their involvement in evolutionary rearrangements. Human Molecular Genetics 12(17), 2201–2208 (2003)
[6] Bailey, J., Baertsch, R., Kent, W., Haussler, D., Eichler, E.: Hotspots of mammalian chromosomal evolution. Genome Biology 5(4), R23 (2004)
[7] Bergeron, A., Mixtacki, J., Stoye, J.: On Computing the Breakpoint Reuse Rate in Rearrangement Scenarios. In: Nelson, C.E., Vialette, S. (eds.) RECOMB-CG 2008. LNCS (LNBI), vol. 5267, pp. 226–240. Springer, Heidelberg (2008)
[8] Bhutkar, A., Schaeffer, S.W., Russo, S.M., Xu, M., Smith, T.F., Gelbart, W.M.: Chromosomal Rearrangement Inferred From Comparisons of 12 Drosophila Genomes. Genetics 179(3), 1657–1680 (2008)
[9] Caceres, M., Sullivan, R.T., Thomas, J.W.: A recurrent inversion on the eutherian X chromosome. Proceedings of the National Academy of Sciences 104(47), 18571–18576 (2007)

[10] Eichler, E.E., Sankoff, D.: Structural Dynamics of Eukaryotic Chromosome Evolution. Science 301(5634), 793–797 (2003)
[11] Fertin, G., Labarre, A., Rusu, I., Tannier, E.: Combinatorics of Genome Rearrangements. MIT Press, Cambridge (2009)
[12] Gordon, L., Yang, S., Tran-Gyamfi, M., Baggott, D., Christensen, M., Hamilton, A., Crooijmans, R., Groenen, M., Lucas, S., Ovcharenko, I., Stubbs, L.: Comparative analysis of chicken chromosome 28 provides new clues to the evolutionary fragility of gene-rich vertebrate regions. Genome Research 17(11), 1603–1613 (2007)
[13] Hannenhalli, S., Pevzner, P.: Transforming men into mouse (polynomial algorithm for genomic distance problem). In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science pp. 581–592 (1995)
[14] Hinsch, H., Hannenhalli, S.: Recurring genomic breaks in independent lineages support genomic fragility. BMC Evolutionary Biology 6, 90 (2006)
[15] Jiang, Z., Tang, H., Ventura, M., Cardone, M.F., Marques-Bonet, T., She, X., Pevzner, P.A., Eichler, E.E.: Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. Nat. Genet. 39(11), 1361–1368 (2007)
[16] Kent, W.J., Baertsch, R., Hinrichs, A., Miller, W., Haussler, D.: Evolution's cauldron: Duplication, deletion, and rearrangement in the mouse and human genomes. Proceedings of the National Academy of Sciences 100(20), 11484–11489 (2003)
[17] Kikuta, H., Laplante, M., Navratilova, P., Komisarczuk, A.Z., Engstrom, P.G., Fredman, D., Akalin, A., Caccamo, M., Sealy, I., Howe, K., Ghislain, J., Pezeron, G., Mourrain, P., Ellingsen, S., Oates, A.C., Thisse, C., Thisse, B., Foucher, I., Adolf, B., Geling, A., Lenhard, B., Becker, T.S.: Genomic regulatory blocks encompass multiple neighboring genes and maintain conserved synteny in vertebrates. Genome Research 17(5), 545–555 (2007)
[18] Koszul, R., Dujon, B., Fischer, G.: Stability of Large Segmental Duplications in the Yeast Genome. Genetics 172(4), 2211–2222 (2006)
[19] Kulemzina, A., Trifonov, V., Perelman, P., Rubtsova, N., Volobuev, V., Ferguson-Smith, M., Stanyon, R., Yang, F., Graphodatsky, A.: Cross-species chromosome painting in cetartiodactyla: Reconstructing the karyotype evolution in key phylogenetic lineages. Chromosome Research 17(3), 419–436 (2009)
[20] Larkin, D.: Role of chromosomal rearrangements and conserved chromosome regions in amniote evolution. Molecular Genetics, Microbiology and Virology 25(1), 1–7 (2010)
[21] Larkin, D.M., Pape, G., Donthu, R., Auvil, L., Welge, M., Lewin, H.A.: Breakpoint regions and homologous synteny blocks in chromosomes have different evolutionary histories. Genome Research 19(5), 770–777 (2009)
[22] Larkin, D.M., Everts-van der Wind, A., Rebeiz, M., Schweitzer, P.A., Bachman, S., Green, C., Wright, C.L., Campos, E.J., Benson, L.D., Edwards, J., Liu, L., Osoegawa, K., Womack, J.E., de Jong, P.J., Lewin, H.A.: A Cattle-Human Comparative Map Built with Cattle BAC-Ends and Human Genome Sequence. Genome Research 13(8), 1966–1972 (2003)
[23] Lecompte, O., Ripp, R., Puzos-Barbe, V., Duprat, S., Heilig, R., Dietrich, J., Thierry, J.C., Poch, O.: Genome Evolution at the Genus Level: Comparison of Three Complete Genomes of Hyperthermophilic Archaea. Genome Res. 11(6), 981–993 (2001)
[24] Longo, M., Carone, D., Program, N.C.S., Green, E., O'Neill, M., O'Neill, R.: Distinct retroelement classes define evolutionary breakpoints demarcating sites of evolutionary novelty. BMC Genomics 10(1), 334 (2009)

[25] Ma, J., Zhang, L., Suh, B.B., Raney, B.J., Burhans, R.C., Kent, J.W., Blanchette, M., Haussler, D., Miller, W.: Reconstructing contiguous regions of an ancestral genome. Genome Research 16(12), 1557–1565 (2006)

[26] Ma, J., Ratan, A., Raney, B.J., Suh, B.B., Miller, W., Haussler, D.: The infinite sites model of genome evolution. Proceedings of the National Academy of Sciences 105(38), 14254–14261 (2008)

[27] Mehan, M.R., Almonte, M., Slaten, E., Freimer, N.B., Rao, P.N., Ophoff, R.A.: Analysis of segmental duplications reveals a distinct pattern of continuation-of-synteny between human and mouse genomes. Human Genetics 121(1), 93–100 (2007)

[28] Misceo, D., Capozzi, O., Roberto, R., DellOglio, M.P., Rocchi, M., Stanyon, R., Archidiacono, N.: Tracking the complex flow of chromosome rearrangements from the Hominoidea Ancestor to extant Hylobates and Nomascus Gibbons by high-resolution synteny mapping. Genome Research 18(9), 1530–1537 (2008)

[29] Mlynarski, E., Obergfell, C., O'Neill, M., O'Neill, R.: Divergent patterns of breakpoint reuse in muroid rodents. Mammalian Genome 21(1), 77–87 (2010)

[30] Mongin, E., Dewar, K., Blanchette, M.: Long-range regulation is a major driving force in maintaining genome integrity. BMC Evolutionary Biology 9(1), 203 (2009)

[31] Murphy, W.J., Larkin, D.M., van der Wind, A.E., Bourque, G., Tesler, G., Auvil, L., Beever, J.E., Chowdhary, B.P., Galibert, F., Gatzke, L., Hitte, C., Meyers, C.N., Milan, D., Ostrander, E.A., Pape, G., Parker, H.G., Raudsepp, T., Rogatcheva, M.B., Schook, L.B., Skow, L.C., Welge, M., Womack, J.E., O'Brien, S.J., Pevzner, P.A., Lewin, H.A.: Dynamics of Mammalian Chromosome Evolution Inferred from Multispecies Comparative Map. Science 309(5734), 613–617 (2005)

[32] Myers, S., Spencer, C.C.A., Auton, A., Bottolo, L., Freeman, C., Donnelly, P., McVean, G.: The distribution and causes of meiotic recombination in the human genome. Biochem. Soc. Trans. 34(Pt. 4), 526–530 (2006)

[33] Myers, S., Bottolo, L., Freeman, C., McVean, G., Donnelly, P.: A Fine-Scale Map of Recombination Rates and Hotspots Across the Human Genome. Science 310(5746), 321–324 (2005)

[34] Nadeau, J.H., Taylor, B.A.: Lengths of Chromosomal Segments Conserved since Divergence of Man and Mouse. Proceedings of the National Academy of Sciences 81(3), 814–818 (1984)

[35] Ohno, S.: Evolution by gene duplication. Springer, Berlin (1970)

[36] Peng, Q., Pevzner, P.A., Tesler, G.: The Fragile Breakage versus Random Breakage Models of Chromosome Evolution. PLoS Computational Biology 2, e14 (2006)

[37] Pevzner, P., Tesler, G.: Genome Rearrangements in Mammalian Evolution: Lessons from Human and Mouse Genomes. Genome Research 13(1), 37–45 (2003)

[38] Pevzner, P.A., Tesler, G.: Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. Proceedings of the National Academy of Sciences 100, 7672–7677 (2003)

[39] Ruiz-Herrera, A., Castresana, J., Robinson, T.J.: Is mammalian chromosomal evolution driven by regions of genome fragility? Genome Biology 7, R115 (2006)

[40] Ruiz-Herrera, A., Robinson, T.: Chromosomal instability in afrotheria: fragile sites, evolutionary breakpoints and phylogenetic inference from genome sequence assemblies. BMC Evolutionary Biology 7(1), 199 (2007)

[41] Ruiz-Herrera, A., Robinson, T.J.: Evolutionary plasticity and cancer breakpoints in human chromosome 3. BioEssays 30(11-12), 1126–1137 (2008)

[42] San Mauro, D., Gower, D.J., Zardoya, R., Wilkinson, M.: A Hotspot of Gene Order Rearrangement by Tandem Duplication and Random Loss in the Vertebrate Mitochondrial Genome. Mol. Biol. Evol. 23(1), 227–234 (2006)
[43] Sankoff, D.: The signal in the genome. PLoS Computational Biology 2(4), 320–321 (2006)
[44] Sankoff, D., Trinh, P.: Chromosomal Breakpoint Reuse in Genome Sequence Rearrangement. Journal of Computational Biology 12(6), 812–821 (2005)
[45] Swenson, K., Moret, B.: Inversion-based genomic signatures. BMC Bioinformatics 10(suppl. 1), S7 (2009)
[46] Webber, C., Ponting, C.P.: Hotspots of mutation and breakage in dog and human chromosomes. Genome Research 15(12), 1787–1797 (2005)
[47] van der Wind, A.E., Kata, S.R., Band, M.R., Rebeiz, M., Larkin, D.M., Everts, R.E., Green, C.A., Liu, L., Natarajan, S., Goldammer, T., Lee, J.H., McKay, S., Womack, J.E., Lewin, H.A.: A 1463 Gene Cattle-Human Comparative Map With Anchor Points Defined by Human Genome Sequence Coordinates. Genome Research 14(7), 1424–1437 (2004)
[48] Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)
[49] Yue, Y., Haaf, T.: 7E olfactory receptor gene clusters and evolutionary chromosome rearrangements. Cytogenetic and Genome Research 112, 6–10 (2006)
[50] Zhao, H., Bourque, G.: Recovering genome rearrangements in the mammalian phylogeny. Genome Research 19(5), 934–942 (2009)
[51] Zhao, S., Shetty, J., Hou, L., Delcher, A., Zhu, B., Osoegawa, K., de Jong, P., Nierman, W.C., Strausberg, R.L., Fraser, C.M.: Human, Mouse, and Rat Genome Large-Scale Rearrangements: Stability Versus Speciation. Genome Research 14, 1851–1860 (2004)

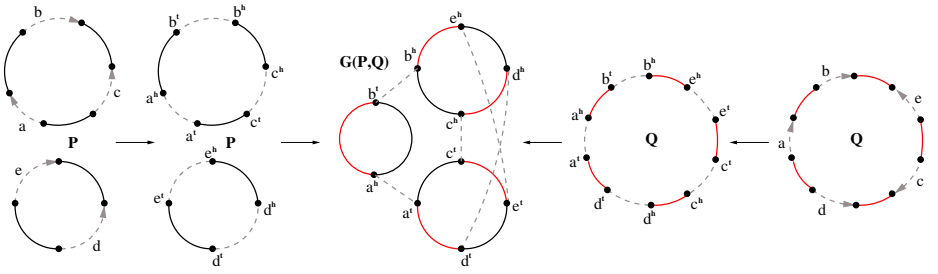


Fig. 1. The breakpoint graph $G(P, Q)$ of a two-chromosomal genome $P = (+a + b - c)(-d + e)$ and a unichromosomal genome $Q = (+a + b - e + c - d)$ represented as two black-obverse cycles and a red-obverse cycle correspondingly. The directions of obverse edges (shown as dashed edges) are not shown in the cases when they are defined by superscripts "t" and "h".

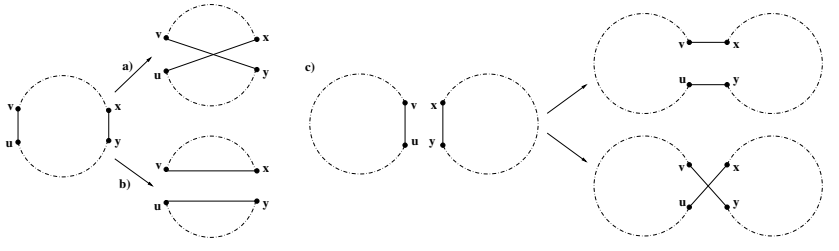


Fig. 2. A 2-break on edges (u, v) and (x, y) corresponding to a) reversal; b) fission; c) translocation/fusion

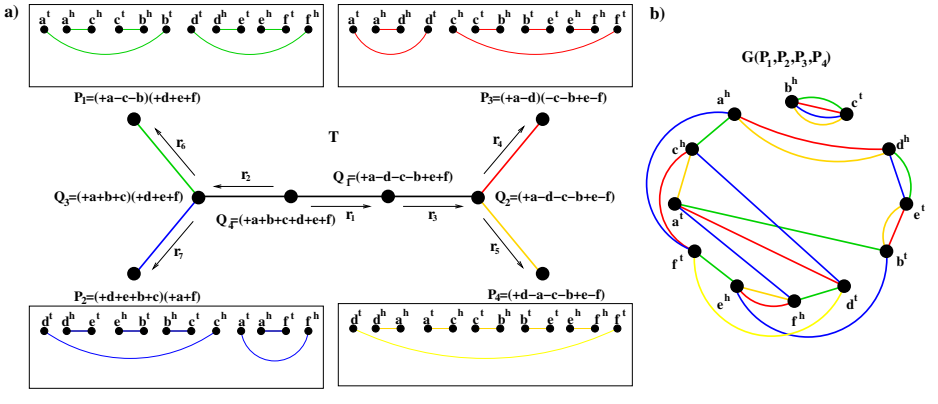


Fig. 3. a) A phylogenetic tree with four circular genomes P_1, P_2, P_3, P_4 (represented as green, blue, red, and yellow graphs respectively) at the leaves and specified intermediate genomes. The obverse edges are not shown. b) The multiple breakpoint graph $G(P_1, P_2, P_3, P_4)$ is a superposition of graphs representing genomes P_1, P_2, P_3, P_4.

Table 1. Left panel: The number of intra- and inter-reuses between 7 branches of the tree in Fig. 4, each of length 100, for simulated genomes with n fragile regions ($n = 500$, 900, 1300). The diagonal elements represent intra-reuses while the elements above diagonal represent inter-reuses. In each cell with numbers $x : y$, x represents the observed reuse while y represents the corresponding lower bound. The cells of the table are colored red (for adjacent branches like $M+$ and $R+$), green (for branches that are separated by a single branch like $M+$ and $D+$ separated by $MR+$), and yellow (for branches that are separated by two branches like $M+$ and $H+$ separated by $MR+$ and $QH+$). **Right panel:** The estimated number of intra- and inter-reuses $bound(e)$ and $bound(e_1, e_2)$ between 7 branches with varying branch lengths as specified in Fig. 4 (data simulated according to FBM with n fragile regions).

	branch lengths = 100							branch lengths as in Fig. 4						
n = 500	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+
M+	63:70	106:106	103:103	97:97	108:108	98:98	113:113	23	45	71	16	22	99	41
R+		57:70	103:103	108:108	98:98	102:102	122:122		34	83	19	25	116	49
D+			65:74	104:104	125:125	104:104	106:106			78	26	37	171	74
Q+				58:68	126:126	120:120	120:120				2	9	39	16
H+					56:62	113:113	116:116					6	51	22
MR+						71:84	104:104						186	102
QH+							54:60							25
n = 900	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+
M+	37:38	70:70	83:83	90:90	72:72	76:76	87:87	13	30	44	9	13	67	25
R+		47:50	67:67	63:63	74:74	68:68	49:49		20	53	11	16	79	31
D+			37:38	69:69	62:62	78:78	84:84			46	17	24	121	45
Q+				32:36	76:76	75:75	94:94				1	4	24	9
H+					40:44	64:64	68:68					4	34	13
MR+						42:44	64:64						113	70
QH+							28:28							14
n = 1300	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+
M+	42:46	46:46	52:52	51:51	47:47	62:62	39:39	8	21	33	7	9	52	19
R+		31:34	53:53	66:66	54:54	48:48	56:56		13	39	8	11	60	24
D+			25:26	64:64	62:62	60:60	64:64			34	12	17	91	34
Q+				22:22	58:58	50:50	50:50				1	3	19	7
H+					30:30	57:57	72:72					2	25	10
MR+						31:34	42:42						81	51
QH+							19:20							9

Table 2. The estimated number of intra- and inter-reuses $bound(e)$ and $bound(e_1, e_2)$ between 7 branches of the phylogenetic tree in Fig. 4 of five mammalian genomes (real data)

	M+	R+	D+	Q+	H+	MR+	QH+
M+	84	68	20	4	5	58	15
R+		96	22	3	6	60	17
D+			174	17	19	98	64
Q+				12	10	25	18
H+					22	23	18
MR+						292	80
QH+							70

Table 3. Subtables of Table 1 (right panel) for $n = 900$ (top part) and Table 2 (bottom part) featuring branches $M+$, $R+$, and $QH+$ as one element of the pair

	M+	R+	D+	Q+	H+	MR+	QH+
M+	13	30	44	9	13	67	25
R+	30	20	53	11	16	79	31
QH+	25	31	45	9	13	70	14

	M+	R+	D+	Q+	H+	MR+	QH+
M+	84	68	20	4	5	58	15
R+	68	96	22	3	6	60	17
QH+	15	17	64	18	18	80	70

Table 4. The breakpoint intra- and inter-reuse (averaged over 100 simulations) for five simulated genomes M, R, D, Q, H under TFBM model with $m = 2000$ synteny blocks, n fragile regions, the turnover rate x, and the evolutionary tree shown in Fig. 4 with the length of each branch equal 100

	x = 0 (FBM)							x = 1							x = 2							x = 3						
n = 500	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+
M+	67	110	109	109	110	111	108	64	93	75	66	65	92	77	63	78	57	45	46	79	57	58	69	47	36	36	68	46
R+		69	110	110	108	109	107		67	76	65	65	92	78		63	57	46	45	78	58		60	47	36	37	69	46
D+			69	109	108	109	109			66	76	77	91	90			62	56	58	78	77			58	46	47	68	69
Q+				68	108	109	110				65	92	77	93				61	79	57	76				60	68	48	66
H+					71	107	109					66	78	94					61	58	79					60	48	67
MR+						70	109						65	91						62	77						57	69
QH+							68							65							61							59
n = 900	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+
M+	42	71	72	71	71	71	71	40	64	58	53	54	65	60	39	60	51	45	45	59	51	37	55	45	39	39	56	44
R+		41	72	71	72	72	73		39	59	53	54	65	58		38	51	45	45	61	51		38	45	39	39	56	45
D+			40	73	72	70	72			41	60	59	65	65			38	50	51	58	60			38	46	46	56	54
Q+				39	73	71	73				39	64	58	64				38	59	49	60				37	55	46	56
H+					41	71	71					38	59	64					38	49	59					37	46	55
MR+						40	74						40	66						40	61						37	55
QH+							41							39							40							37
n = 1300	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+	M+	R+	D+	Q+	H+	MR+	QH+
M+	28	54	52	54	52	53	55	27	45	46	45	44	49	47	27	46	44	40	39	48	41	28	45	40	37	38	45	40
R+		28	53	53	54	53	52		29	45	44	44	48	48		28	43	40	41	46	44		27	41	38	37	44	39
D+			31	52	51	53	54			28	46	47	50	49			29	42	42	47	46			27	39	41	44	46
Q+				28	52	55	53				29	50	46	50				28	49	42	47				27	46	39	45
H+					29	53	52					28	47	49					27	42	46					27	41	44
MR+						27	53						29	49						27	48						27	46
QH+							29							28							28							27

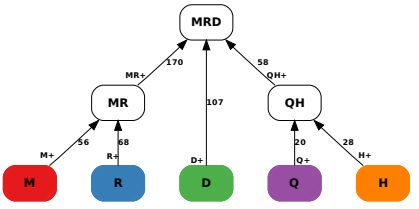


Fig. 4. The phylogenetic tree T on five genomes M, R, D, Q, and H. The branches of the tree are denoted as $M+$, $R+$, $D+$, $Q+$, $H+$, $MR+$, and $QH+$.

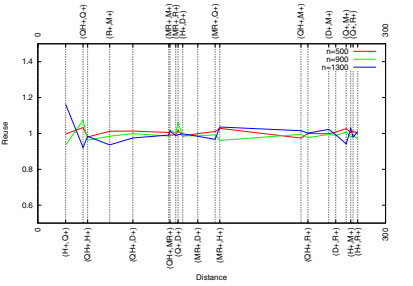


Fig. 5. The scaled inter-reuse for five simulated genomes M, R, D, Q, H (averaged over 100 simulations) on n fragile regions (for $n =$ 500, 900, and 1300) with the evolutionary tree and branch lengths shown in Fig. 4

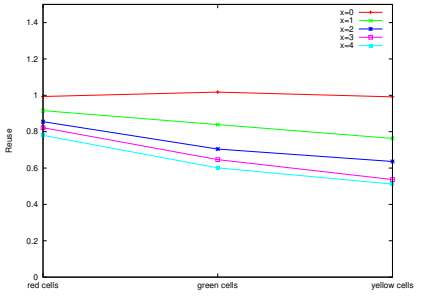


Fig. 6. The scaled inter-reuse for five simulated genomes M, R, D, Q, H on $m =$ 2000 synteny blocks, $n = 900$ fragile regions, and the turnover rate x varying from 0 to 4 with the phylogenetic tree and branch lengths shown in Fig. 4. The simulations follow FBM ($x = 0$) and TFBM (x varies from 1 to 4). The plot shows the scaled inter-reuse for only three reference points (corresponding to red, green, and yellow cells) that are somewhat arbitrarily connected by straight segments for better visualization.

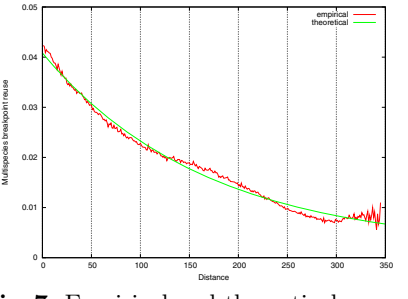


Fig. 7. Empirical and theoretical curves representing the number of reuses $R(\ell)$ as a function of distance ℓ between pairs of sub-branches of the tree in Fig. 4 of the five mammalian genomes (ancestral genomes were computed using MGRA [4]). The empirical curve is averaged over 1000 random samplings of shortest rearrangement scenarios, while the theoretical curve represents the best fit with parameters $n \approx 196$, $x \approx 1.12$, and $m \approx 4017$.

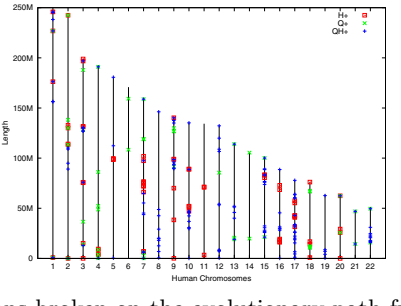


Fig. 8. Positions of regions broken on the evolutionary path from the rodent-primate-carnivore ancestor (i.e., on $H+$, $Q+$, and $QH+$ branches) projected to the human chromosomes

Mapping Association between Long-Range *Cis*-Regulatory Regions and Their Target Genes Using Comparative Genomics

Emmanuel Mongin[1], Ken Dewar[2], and Mathieu Blanchette[1]

[1] Department of human genetics, McGill University
[2] School of Computer Science, McGill University

Abstract. In chordates, long-range *cis*-regulatory regions are involved in the control of transcription initiation (either as repressors or enhancers). They can be located as far as 1 Mb from the transcription start site of the target gene and can regulate more than one gene. Therefore, proper characterization of functional interactions between long-range *cis*-regulatory regions and their target genes remains problematic. We present a novel method to predict such interactions based on the analysis of rearrangements between the human and 16 other vertebrate genomes. Our method is based on the assumption that genome rearrangements that would disrupt the functional interaction between a *cis*-regulatory region and its target gene are likely to be deleterious. Therefore, conservation of synteny through evolution would be an indication of a functional interaction. We use our algorithm to classify a set of 1,406,084 putative associations from the human genome. This genome-wide map of interactions has many potential applications, including the selection of candidate regions prior to *in vivo* experimental characterization, a better characterization of regulatory regions involved in position effect diseases, and an improved understanding of the mechanisms and importance of long-range regulation.

1 Introduction

Transcription initiation is controlled by distinct genomic regions that act as binding platform for transcription factors. Accurate regulation is crucial to many biological processes such as development, tissue specific expression or response to external stimuli. In vertebrates, distinct *cis*-regulatory regions, with their own specificity, take part in complex cross-talking processes that result in proper gene regulation. Alteration of those regions or disruption of the synteny between *cis*-regulatory regions and their target genes can have dramatic phenotypic effects often leading to diseases [1].

Long-range *cis*-regulatory regions are located over 1.5 kb upstream or downstream from the transcription start site and regulate genes at distances reaching 1Mb [2]. Those long-range regulators comprise various type of regions such as enhancers, silencers or insulators. Long-range regulatory regions may regulate many genes, act in an orientation independent manner, and more importantly

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 216–227, 2010.

regulate target genes over very large distances. Consequently, predicting the putative target gene(s) of a given *cis*-regulatory region is a difficult task. Such predictions would meet with various interests such as determining what transcription factor regulation what genes or help associate *cis*-regulatory genetic variation to expression variation.

Following the publication of the human genome sequence, we have witnessed an increasing number of vertebrate genome sequencing projects reaching completions for genomes ranging from teleost fish to mammals.This, combined with fast and accurate genomic DNA alignment programs [3,4], boosted the field of comparative genomics. Of specific interest in our context are methods allowing the identification of about 100,000 non-coding evolutionarily conserved regions (e.g. PhastCons [5]), most of which apparently regulatory regions but being located very far from the start of any annotated transcript. Although there is now ample evidence linking these regions to regulatory functions [6], computational approaches have rarely attempted to predict the gene targets of these regulatory regions.

Studies of genome evolution and genome rearrangements have also greatly benefited from the increase in genomic data. Of particular interest is the fact that evolutionary rearrangements (those rearrangements that become fixed in a population during evolution) have been shown to occur more frequently in certain genomic regions [7,8] and not uniformly at random as previously modeled [9]. The likelihood of a genome rearrangement becoming fixed in a population is strongly dependent on the fitness of the mutated individuals. In the context of long-range regulation, a rearrangement disrupting the physical link between a regulatory region and its target gene (i.e. involving a breakpoint between the two loci) will usually be deleterious to some extent, and would rarely become fixed in a population. Therefore evolutionary rearrangements are thought to largely involve breakpoints located in regions of the genome where they will not disrupt long-range regulation [10,11].

The idea of using conservation of synteny to link regulatory regions to their target genes was first introduced by Flint *et al.* who used it to map interactions within the α-globin cluster [12]. Ahituv *et al.* used the same principle to make association predictions in the complete human genome based on its comparison to the mouse, chicken, and frog genomes [13]. A similar approach was used by Sun *et al.* based on a slightly larger set of six vertebrate genomes [14], by Vavouri *et al.*, using paralogous regions [15], and by Dong *et al.* [16], who developed Synoth, a useful web server based on these ideas. Although conceptually similar to the approach proposed here, their approach requires perfect synteny conservation and is not based on a phylogenetic model of evolution, which limits its applicability in cases where the number of genomes being compared is larger (and thus more informative), as is available today. Clearly, there is a need for a new method that would allow the prediction of associations between regulatory regions and their target gene in a manner that can take advantage of large sets of phylogenetically-diverse genomes and that is applicable to the vast majority of human cis-regulatory regions that are not conserved outside mammals.

We therefore propose a new computational method based on the study of the conservation of the genomic proximity of 1,406,084 pairs of human genes and *cis*-regulatory regions in 16 other vertebrate genomes to assess the likelihood of functional interaction for each. The result is a genome-wide map of predicted functional interactions between long-range *cis*-regulatory regions and their putative target genes in the human genome.

2 Results

2.1 Orthology Mapping

We assess the functional interaction between genes and putative *cis*-regulatory regions based on the conservation of their physical proximity on chromosomes in various vertebrate genomes. The gene set is composed of 25,575 human genes (EnsEMBL genes version 54 [17], excluding pseudogenes), consisting of a total of 257,985 exons. The set contains 21,404 protein coding genes, 1664 miRNAs, 1334 snRNAs, 717 snoRNAs, and 444 rRNAs.

Various functional studies have shown that non-coding regions under purifying selection are enriched for elements with regulatory properties (e.g. [6]). Our set of putative *cis*-regulatory regions is composed of 123,905 human non-coding conserved regions (99,512 intergenic and 24,393 intronic) from the UCSC 28-way conserved regions [18,5] (see Methods). In this paper, we work under the assumption that these non-coding conserved elements (NCEs) have a regulatory function, although a small fraction of them is expected to have other functions or to be non-functional.

Genomes were selected for this analysis based on the following criteria: (i) Evolutionary distance from human: Highly diverged species have undergone more genome rearrangements and are thus more informative for this study; (ii) Genome coverage: Complete and accurate genome assemblies are required to assess synteny conservation; genomes sequenced at low coverage were thus excluded. The 16 species selected (see Figure 1 (c)) include 8 mammals, 2 birds, one reptile, one batrachian, and 5 fish. We next mapped both types of human elements (exons and NCEs) to these genomes by taking advantage of whole genome alignments ("liftover chains" [19]; see Methods). As evolutionary distance from human becomes larger, an increasing fraction of human elements fail to map to other genomes, either because they simply do not exist there or because they have diverged beyond recognition. As expected, protein coding genes exhibit a deeper overall conservation level than other types of transcribed or putative *cis*-regulatory regions. For example, 47 to 54 % of human protein coding exons map to teleost fish whereas only 1-7% of non-coding RNAs and only 2% of non-coding conserved regions map to these species. For future analyses, the level of conservation of each gene and NCE was defined as the ancestral node corresponding to the last common ancestor (e.g. eutherian, amniote, or gnathostomate ancestor) of the set of extant species where it exists.

2.2 A Map of Functional Interaction between Regulatory Elements and Target Genes

Our algorithm to identify functional NCE-gene associations is summarized in Figure 1. First, only the 1,406,084 pairs of human NCE and gene separated by at most 1 Mb are considered as potentially functional[1]. A pair of a human gene and a human NCE is labelled *associated* in a given species S if: i/ if both regions have been mapped to S, ii/ they lie on the same chromosome, and iii/ they are within at most D_S bp from each other, where D_S is a species-specific distance threshold analogous to the 1Mb threshold for human but scaled based on the size of the genome of S (see Methods). A pair that is not associated can be either *separated* (both components exist but have been separated by a rearrangement, or only the NCE remains conserved) or *incomplete* (either the NCE or both elements could not be mapped to S).

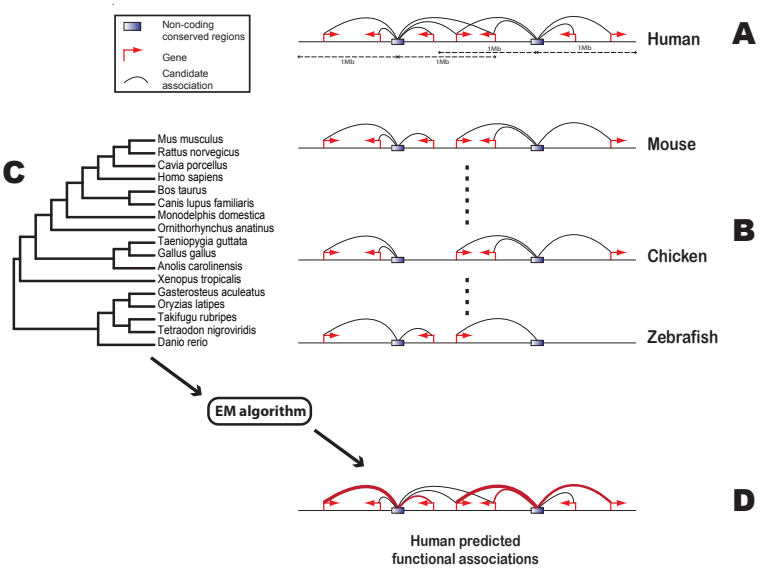


Fig. 1. Steps undertaken to calculate functional interaction scores.(A) All pairs composed of a human NCE and a human gene in the same physical proximity are retrieved (candidate associations). (B) The physical association of each candidate pair is assessed in 16 other vertebrate genomes. (C) The ancestral association status of each pair is inferred using a variant of the Fitch algorithm. (D) A likelihood ratio score (functionally associated vs non-functionally associated models) is associated to each candidate pair.

[1] We do not mean to imply that functional interactions cannot happen over larger distances; however, the vast majority of interactions are expected to be within this range.

From the mapping data, the ancestral association status (associated, separated, or incomplete) of each pair is first inferred for each ancestral node of the phylogenetic tree using a variant of the Fitch algorithm [20] (see Methods). An expectation-maximization (EM) algorithm is subsequently used to learn (in an unsupervised manner) two models: one for functionally associated pairs, and one for non-functionally associated pairs (see Figure 1, sections C and D). Each model specifies the probability of maintaining or breaking a human association at each node of the phylogenetic tree. The functional interaction score for each pair is obtained as the log-likelihood ratio of the two models. Refer to Methods for full details.

The distribution of functional interaction score is tri-modal (see Figure 2, A). The first peak (score < -10) includes 327,511 human pairs for which functional interaction can be clearly ruled out based on evolutionary evidence - we call these pairs *confidently non-associated*. Pairs scoring from -10 to 49 belong to a grey zone where evolutionary evidence is inconclusive - 910,465 pairs fall in this category. As the genomes of more vertebrate species become sequenced, the number of these inconclusive cases should be reduced. Finally, the 168,108 pairs with score over 49 are called *confidently associated* pairs.

As shown in Figure 2 (B), the functional interaction score of a NCE-gene pair depends on the conservation level of its two constituents. For example, large positive scores can only be reached by pairs where both the gene and the NCE are conserved back to gnathostomate ancestor. Indeed, confidently associated pairs almost exclusively involve NCEs and genes that are conserved at least as far back as the amniote ancestor. Pairs where either the gene or the non-coding conserved region is only conserved within eutherians generally obtain scores closer to zero.

Figure 2 (C) shows the distribution of the number of NCEs associated to a given gene, and the distribution of the number of genes associated to a given NCE. Most NCEs cannot be confidently associated to any given gene because of the lack of evolutionary evidence. Excluding those, an NCE is predicted to be linked to an average of 1.5 genes. Note that this average reflects evolutionary evidences but may not correspond to what would be observed *in vivo*.

2.3 Regulatory Complexity

Our map of predicted gene-NCE functional interactions allows studying several aspects of gene regulation. We first classified genes based on the number of NCEs predicted to be functionally interacting with them (confidently associated pairs). We say that a gene has a *complex regulation* if at least 20 NCEs are predicted to interact with it, a *simple regulation* if it is linked to 1 to 5 NCEs (we will use this set of genes as the background), and a *basic regulation* if no NCE is associated to it[2]. 619 (3.0%) genes have a basic regulation, 3921 (15.6%) genes have a simple regulation, and 2395 (11.6%) genes have a complex regulation.

[2] Genes predicted to be associated to 6-19 NCEs are of course interesting too but will not be considered in this analysis.

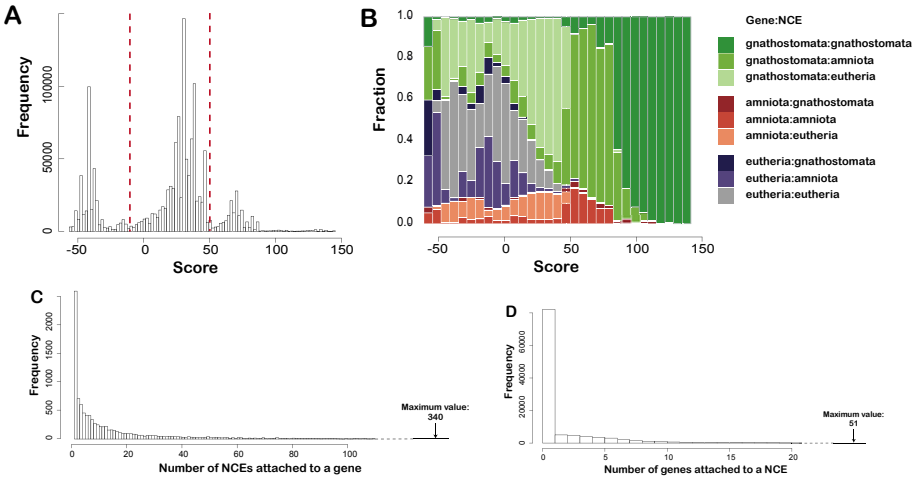


Fig. 2. Functional association score distribution. (A) Distribution of scores for all candidate associations. (B) For each score bin, the proportion of pairs at a given conservation level is given. In the legend, the conservation level is given first for the gene then for the non-coding region. (C) Distribution of the number of NCEs predicted to interact with a gene. (D) Distribution of the number of genes predicted to interact with a NCE.

Gene ontology analyses. Genes with basic and complex regulation were tested for enrichment in biological processes compared to those with simple regulation using the Babelomics platform [21] (see Table 1). Genes with complex regulation show an enrichment for genes involved in transcription and development (trans/dev) biological processes[3]. Genes involved in transcription and developmental processes have complex spatio-temporal expression pattern. Such regulation is permitted by many *cis*-regulatory regions, which allow specific expression in different tissues at different developmental time [22]. Such genes are also known to be located in the vicinity of, and remain in synteny with, gene deserts, regions known to contain a large number of *cis*-regulatory regions [23].

When compared to genes with simple regulation (1-5 associated NCEs), the genes with basic regulation (0 associated NCEs) show enrichment for GO biological processes involved in neurological processes and adaptive processes including "response to biotic stimulus", "defense response", "immune response" and "cell communication".Similar enrichments have previously been observed in highly

[3] Genes performing these functions are known to be more evolutionary conserved than other types of biological processes. Since highly conserved genes are more likely to be associated with surrounding NCEs (see Figure 2, B), to control for conservation bias, we undertook the same analysis restricting the background set to only genes predating tetrapoda divergence. The results obtained show significant p-values for similar trans/dev GO categories.

Table 1. GO analysis of the set of genes with complex regulation. Enrichment for level 3 GO biological processes is calculated with a double fisher-test on the Babelomics plateform (after correction for multiple hypothesis testing), using the set of genes with simple regulation as background.

GO category (level 3)	% of genes with complex regulation	% of genes in background set	Fold increase	Corr. p-value
reproductive process	2.56	1.19	2.2	1.1e-2
multicellular organismal dev.	25.2	15.49	1.6	8.7e-13
anatomical structure dev.	21.72	15.37	1.4	4.7e-06
cellular developmental process	19.89	15.1	1.3	5.7e-4
regulation of biological process	41.06	31.55	1.3	1.2e-08
macromolecule metabolic process	54.48	47.08	1.1	5.3e-05
primary metabolic process	62.36	55.65	1.1	2.2e-04
cellular metabolic process	63.45	56.88	1.1	2.6e-4
cellular component org. and biog.	18.06	21.79	-1.2	2.3e-2
establishment of localization	17.75	23.1	-1.3	2.9e-4

evolutionary rearranged regions [11,10]. Indeed, most genes with basic regulation lie within heavily rearranged regions, which explains why no association are detected.

Differences in levels of regulatory complexity are also observed for different types of transcripts (protein-coding, snoRNA, miRNA). Small nucleolar RNAs (snoRNA) are predicted to interact with an average of only 12.4 NCEs whereas miRNA have nearly twice as many, with 22.7 (p-value = 0.018, two-sided Wilcoxon rank sum test). Protein coding genes stand in between with a mean of 16.9. The number of other types of RNA genes was too small for this analysis.

miRNAs are short 22 nucleotides RNA molecules transcribed by RNA polymerase II that regulate the stability and translation of mRNAs . miRNAs play a key role in cellular differentiation and are tightly regulated during development.Their regulation, similarly to developmental genes, is probably under the complex control of various NCEs. In contrast, snoRNAs are mainly involved in rRNA nucleotide modifications (although it seems that some show tissue specificity and developmental regulation).Therefore, snoRNA are less likely to be tightly regulated as their main role is to participate in housekeeping functions.

Associated modules are enriched for enhancer regions. Similarly to our gene-centered analysis, we analyzed NCEs depending on the number of genes they were predicted to interact with. A certain fraction of NCEs have a function other that of being *cis*-regulatory regions (e.g. unannotated protein coding or RNA exons). However, one would expect that, if our interaction predictions are correct, these non-regulatory NCEs would not be predicted to interact with many other genes, except perhaps in the case of additional exons. On average, a NCE is associated to 1.5 genes, with a maximum of 51 genes (see Figure 2, D). We

Table 2. Overlap with histone modification data by module type. Percentage of niRNA and hiRNA overlapping regions marked with various types of histone modifications. P-values are calculated with a two-sided Fisher test.

Chromatin annotation	niNCE (%)	hiNCE (%)	Fold increase	P-value
H3K4Me1 (enhancers)	0.9	12	14	$< 2.2 \times 10^{-16}$
H3K4Me3 (promoters)	0.2	8.1	40.5	$< 2.2 \times 10^{-16}$
CTCF	0.6	2.6	4.3	1.7×10^{-13}

created two sets of modules depending on the number of genes with which they have been predicted to interact. The first set is composed of 4604 (3.7%) NCEs that are not predicted to interact with any genes - we call them non-interacting NCEs (niNCEs). The second set, highly interacting NCEs (hiNCEs), is composed of 3770 (3.0%) NCEs that are predicted to BE functionally associated to at least 10 genes.

Different types of histone modifications have been associated to active chromatin as well as to different types of *cis*-regulatory regions [24]. We tested the overlap of both hiNCE and niNCE with genomic regions exhibiting such modifications as well as with regions characterized as CTCF binding sites, as detected by Chip-Seq experiments [25].

The set of hiNCE (the most likely to be functional cis-regulatory regions) has higher overlap with H3K4Me1 and H3K4Me3 histone marks [24,25] (respectively corresponding to enhancer and promoter regions) than the niNCE dataset (see Table 2). This difference is highly significant in all cases. This result could be biased by the higher average level of conservation of hiNCEs compared to niNCEs. However, even when we control for this by taking into consideration only NCEs that predate tetrapoda divergence, the overlap with H3K4Me1 and H3K4Me3 regions remains significative with respectively fold increase of 7.8 (p-value $= 8.9 \times 10^{-11}$) and 30.3 (p-value $= 3.4 \times 10^{-11}$). These results suggest that niNCEs and hiNCEs play different roles. Since this classification is only based on our interaction predictions, this constitutes an indirect validation of our predictions.

3 Discussion and Conclusion

Linking long-range regulatory elements to the gene(s) they regulate is a challenge that remains mostly unsolved for both experimental and computational biologists. Indeed, there currently exists no high-throughpout experiment that can unambiguously identify target genes for a given long-range *cis*-regulatory region.

Conservation of synteny between a NCE and a gene may be due to the presence of a functional interaction between the two or to the lack of divergence time for genome rearrangements to have separated them. At the time of the study, we were limited to the comparison of the human genome to that of 16 other vertebrates whose genomes are assembled in supercontigs of size at least 10Mb. As

seen on Figure 2, a large number of NCE-gene pairs, especially those involving eutherian-specific NCEs, receive low-confidence scores due to the insufficient evolutionary evidence. The level of divergence (away from human) of the genomes considered impacts the amount of information a new sequence provides. Fish genomes have undergone a lot of rearrangements - which is good for our study -, but only 2% of human NCEs can be traced back to these species. Placental mammal genomes share most NCEs with human, but have typically undergone a small number of rearrangements. Improved resolution can only be obtained by increasing the number of genomes compared, at various degrees of divergence (especially marsupials, birds, and reptiles). With whole genome sequencing becoming increasingly cheap, we expect that the accuracy of our approach will quickly increase significantly.

4 Methods

4.1 Data Selection and Orthology Mapping

We retrieved human non-coding conserved regions from the human 28-way [18] alignment dataset available on the UCSC genome browser [19], excluding any region with any overlap with EnsEMBL exons, mRNAs, or repeatMasker regions. Only regions with a score over 400 and a length over 100 bp are retained for further analysis. Exons were retrieved from the Ensembl (version 54) human gene prediction dataset (but excluding predictions labelled as pseudogenes)[17].

Both human non-coding conserved regions and coding regions were mapped using liftover [19] (with blastz nets) to the following genomes: mouse,rat,guinea pig,dog cow, opossum, platypus,chicken, zebra finch, lizard, frog, zebrafish, stickelback, tetraodon, fugu and medaka.

The mapping process of both coding and non-coding regions is composed of two steps. First, all human non-coding conserved regions and exons are mapped to the 16 target genomes. Second, each mapped region is mapped back to human. Only hits which map back to the same original region in human are kept (reciprocal best hits). In the case of multi-exon genes, they are considered to be mapped to a given genome if at least one of their exons is. Each human gene and non-coding conserved region is thus mapped to either zero or exactly one position in the genome of each other species.

4.2 Predicting Functional Interaction between Genes and Non-coding Regions

Let G_S and N_S be the set of genes and non-coding conserved regions that have been mapped from human to species S. Let $P_S \subseteq G_S \times N_S$ be the set of all pairs of gene and non-coding region from species S that are located at most δ_S base pairs apart (on the same chromosome) in the genome of S. Note that genes can be paired with several non-coding regions (or to none at all), and non-coding conserved regions can be paired with several genes (or to none at all). We are

interested in classifying the pairs from P_{human} into functionally associated or non-functionally associated, based on the presence of the pair in the 16 other species. We set δ_{human} to 1 Mb and adjust the distance thresholds for other species in proportion to their genome size, relative to human: $\delta_S = 1.25Mb \cdot$ (GenomeSize(S)/GenomeSize(human)). Note that we use a constant of 1.25 Mb instead of 1 Mb to deal more gracefully with boundary cases where a pair may, for example, be located 0.99 Mb apart in human and 1.01 Mb apart in another species (if both species have similar length). For any pair $(g, n) \in P_{human}$, we define the conservation status of that pair in species S as:

$$C_S(g,n) = \begin{cases} \text{conserved} & \text{if } (g,n) \in P_S \\ \text{separated} & \text{if } (n \in N_S \text{ and } g \notin G_S) \text{ or } (g,n) \notin P_S \\ \text{missing} & \text{if } n \notin N_S \end{cases}$$

4.3 Inference of Ancestral Association Status

The ancestral status $C_u(g, n) \in$ {conserved, separated, missing} of each pair $(g, n) \in P_{human}$ is reconstructed for each ancestral node u of the phylogenetic tree, using a variant of the Fitch algorithm [20] for parsimonious reconstruction. Briefly, in an initial bottom-up phase, the set of possible states at each ancestral node is inferred based on those at its two children, following the Fitch algorithm. The state of the root of the tree is then obtained (breaking ties in favor of the "separated" state) and the information is propagated down the tree to obtain ancestral states at each node.

4.4 Expectation-Maximization Algorithm

We now present two probabilistic models for pairs of NCEs and genes. The Θ^F model describes pairs that are functionally associated and for which there is selective pressure to maintain the pairing. The Θ^{NF} model describes the evolution of pairs that are not functionally associated. Each model $M \in \{F, NF\}$ is specified as follows: $\Theta^M = (\rho_1^M, \rho_2^M, ..., \rho_{2n-2}^M)$, where ρ_u^M is the probability distribution over states at node u with model M, i.e. $\rho_u^M(a)$ is the probability of observing state a at node u ($a \in$ {conserved, separated, missing}).

Let $A(g, n) \in$ {functional, non-functional} of each pair in $(g, n) \in P_{human}$ be the true (but unknown) functional status of pair (g, n). Because the true association status of each pair is unknown, parameters of each model are estimated in an unsupervised manner using an EM-like algorithm to find maximum likelihood estimators, based on the complete set of pairs P_{human} considered. The algorithm alternates between predicting the functional status of each pair (based on the likelihood ratio of the two models) and revising the parameters of the two models based on the predicted classification. Iterating the algorithm yields estimates for the parameters and assigns log-likelihood ratio scores to each pair in P_{human}. See below for more details.

EM algorithm
Input: A phylogenetic tree T, where each leaf and internal node u is labeled with a set P_u of putatively associated pairs of genes and NCE. One leaf if labeled "human".

Output: A label $A(g,n) \in \{F, NF\}$ for each pair $(g,n) \in P_{human}$, and a model $\Theta^M = (\rho_1^M, \rho_2^M, ..., \rho_{2n-2}^M)$ that maximizes the likelihood of the observed data.
for all $(g,n) \in P_{human}$ **do**
  $A(g,n) \leftarrow F$ or NF randomly
end for
repeat
  for all $M \in \{F, NF\}$ **do**
    for all $a \in$ {conserved, separated, missing} **do**
      for all $u \in V(T)$ **do**
        $P_u^M(a) = \frac{|\{(g,n): A(g,n)=M \text{ and } C_u(g,n)=a\}|+1}{|\{(g,n): A(g,n)=M\}|+3}$
      end for
    end for
  end for
  for all $(g,n) \in P_{human}$ **do**
    $LLR(g,n) = \frac{\prod_{(u)\in V(T)} P_u^F(C_u(g,n))}{\prod_{(u)\in V(T)} P_u^{NF}(C_u(g,n))}$
    if $LLR(g,n) \geq 1$ **then** $A(G,n) \leftarrow F$
    else $A(G,n) \leftarrow NF$
  end for
until Convergence

Acknowledgements

We wish to thank three anonymous reviewers for their valuable suggestions. This work was funded in part by Genome Canada and by NSERC.

References

1. Leipoldt, M., Erdel, M., Bien-Willner, G.A., et al.: Two novel translocation breakpoints upstream of sox9 define borders of the proximal and distal breakpoint cluster region in campomelic dysplasia. Clin. Genet. 71, 67–75 (2007)
2. Lettice, L.A., Heaney, S.J.H., Purdie, L.A., Li, L., de Beer, P., et al.: A long-range shh enhancer regulates expression in the developing limb and fin and is associated with preaxial polydactyly. Hum. Mol. Genet. 12, 1725–1735 (2003)
3. Blanchette, M., Kent, W., Riemer, C., Elnitski, L., Smit, A., et al.: Aligning multiple genomic sequences with the threaded blockset aligner. Genome Res. 14, 708–715 (2004)
4. Paten, B., Herrero, J., Beal, K., Fitzgerald, S., Birney, E.: Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. Genome Res. 18, 1814–1828 (2008)
5. Siepel, A., Bejerano, G., Pedersen, J., Hinrichs, A., Hou, M., et al.: Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. Genome Res. 15, 1034–1050 (2005)
6. Pennacchio, L.A., Ahituv, N., Moses, A., Prabhakar, S., et al.: In vivo enhancer analysis of human conserved non-coding sequences. Nature 444, 499–502 (2006)

7. Pevzner, P., Tesler, G.: Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. Proc. Natl. Acad. Sci. USA 100, 7672–7677 (2003)
8. Peng, Q., Pevzner, P.A., Tesler, G.: The fragile breakage versus random breakage models of chromosome evolution. PLoS Comput. Biol. 2, e14 (2006)
9. Nadeau, J.H., Taylor, B.: A Lengths of chromosomal segments conserved since divergence of man and mouse. Proc. Natl. Acad. Sci. USA 81, 814–818 (1984)
10. Mongin, E., Dewar, K., Blanchette, M.: Long-range regulation is a major driving force in maintaining genome integrity. BMC Evol. Biol. 9, 203 (2009)
11. Larkin, D.M., Pape, G., Donthu, R., Auvil, L., Welge, M., et al.: Breakpoint regions and homologous synteny blocks in chromosomes have different evolutionary histories. Genome Res. 19, 770–777 (2009)
12. Flint, J., Tufarelli, C., Peden, J., Clark, K., Daniels, R.J., et al.: Comparative genome analysis delimits a chromosomal domain and identifies key regulatory elements in the alpha globin cluster. Hum. Mol. Genet. 10, 371–382 (2001)
13. Ahituv, N., Prabhakar, S., Poulin, F., Rubin, E.M., Couronne, O.: Mapping cis-regulatory domains in the human genome using multi-species conservation of synteny. Hum. Mol. Genet. 14, 3057–3063 (2005)
14. Sun, H., Skogerbø, G., Wang, Z., Liu, W., Li, Y.: Structural relationships between highly conserved elements and genes in vertebrate genomes. PLoS ONE 3, e3727 (2008)
15. Vavouri, T., McEwen, G.K., Woolfe, A., Gilks, W.R., Elgar, G.: Defining a genomic radius for long-range enhancer action: duplicated conserved non-coding elements hold the key. Trends Genet. 22, 5–10 (2006)
16. Dong, X., Fredman, D., Lenhard, B.: Synorth: exploring the evolution of synteny and long-range regulatory interactions in vertebrate genomes. Genome Biology 10, R86 (2009)
17. Hubbard, T.J.P., Aken, B.L., Ayling, S., Ballester, B., Beal, K., et al.: Ensembl. Nucleic Acids Res. 37, D690–D697 (2009)
18. Miller, W., Rosenbloom, K., Hardison, R.C., Hou, M., Taylor, J., et al.: 28-way vertebrate alignment and conservation track in the ucsc genome browser. Genome Res. 17, 1797–1808 (2007)
19. Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., et al.: The human genome browser at ucsc. Genome Res. 12, 996–1006 (2002)
20. Fitch, W.M., Margoliash, E.: Construction of phylogenetic trees. Science 155, 279–284 (1967)
21. Al-Shahrour, F., Carbonell, J., Minguez, P., Goetz, S., Conesa, A., et al.: Babelomics: advanced functional profiling of transcriptomics, proteomics and genomics experiments. Nucleic Acids Res. 36, 341–346 (2008)
22. Howard, M.L., Davidson, E.H.: cis-regulatory control circuits in development. Dev. Biol. 271, 109–118 (2004)
23. Ovcharenko, I., Loots, G.G., Nobrega, M.A., Hardison, R.C., Miller, W., et al.: Evolution and functional classification of vertebrate gene deserts. Genome Res. 15, 137–145 (2005)
24. Bernstein, B.E., Kamal, M., Lindblad-Toh, K., Bekiranov, S., Bailey, D.K., et al.: Genomic maps and comparative analysis of histone modifications in human and mouse. Cell 120, 169–181 (2005)
25. Mikkelsen, T.S., Ku, M., Jaffe, D.B., Issac, B., Lieberman, E., et al.: Genome-wide maps of chromatin state in pluripotent and lineage-committed cells. Nature 448, 553–560 (2007)

A New Genomic Evolutionary Model for Rearrangements, Duplications, and Losses That Applies across Eukaryotes and Prokaryotes

Yu Lin and Bernard M.E. Moret

Laboratory for Computational Biology and Bioinformatics,
Swiss Federal Institute of Technology (EPFL),
EPFL-IC-LCBB, INJ 230, Station 14, CH-1015 Lausanne, Switzerland
{yu.lin,bernard.moret}@epfl.ch

Abstract. *Background:* Genomic rearrangements have been studied since the beginnings of modern genetics and models for such rearrangements have been the subject of many papers over the last 10 years. However, none of the extant models can predict the evolution of genomic organization into circular unichromosomal genomes (as in most prokaryotes) and linear multichromosomal genomes (as in most eukaryotes). Very few of these models support gene duplications and losses—yet these events may be more common in evolutionary history than rearrangements and themselves cause apparent rearrangements.

Results: We propose a new evolutionary model that integrates gene duplications and losses with genome rearrangements and that leads to genomes with either one (or a very few) circular chromosome or a collection of linear chromosomes. Moreover, our model predictions fit observations about the evolution of gene family sizes as well as existing predictions about the growth in the number of chromosomes in eukaryotic genomes. Finally, our model is based on the existing inversion/translocation models and inherits their linear-time algorithm for pairwise distance computation.

1 Introduction

Genomic rearrangements have been studied since the beginnings of modern genetics (starting in the 1920s with the classic work of Sturtevant and Dobzhansky [19,20]) and models for such rearrangements have been the subject of many papers over the last 20 years (for a review, see [4]). However, none of the extant models predicts the evolution of genomic organization into circular unichromosomal genomes (as in most prokaryotes) and linear multichromosomal genomes (as in most eukaryotes). In addition, hardly any of these models support gene duplications and losses alongside rearrangements; yet duplications and losses may be more common in evolutionary history than rearrangements and, moreover, they themselves cause apparent rearrangements.

In this paper, we propose a new evolutionary model, based on the classical inversion/translocation (HP) [6] and double-cut-and-join (DCJ) [2,23] models, that integrates gene duplications and losses with genome rearrangements and that leads to genomes with either a single (or a very few) circular chromosome or a collection of linear chromosomes. Moreover, our model predictions fit observations (as presented by Lynch

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 228–239, 2010.

[16]) about the evolution of gene family sizes, as well as existing predictions (by Imai's group [11]) about the growth in the number of chromosomes in eukaryotic genomes. Finally, our model inherits the algorithmic results developped for previous models, such as a linear-time distance computation [1,2,3].

2 Background

Evolutionary events that affect the gene order of genomes include various rearrangements, which affect only the order, and gene duplications and losses, which affect both the gene content and, indirectly, the order. (Gene insertion, corresponding to lateral gene transfer or neofunctionalization of a gene duplicate, can be viewed as a special case of duplication.)

Rearrangements themselves include inversions, transpositions, block exchanges, circularizations, and linearizations, all of which act on a single chromosome, and translocations, fusions, and fissions, which act on two chromosomes. These operations are subsumed in the *double-cut-and-join (DCJ)* [2,23], which has formed the basis for much algorithmic research on rearrangements over the last few years. A DCJ operation makes two cuts, which can be in the same chromosome or in two different chromosomes, producing four cut ends, then rejoins the four cut ends in any of the three possible ways. The DCJ model is more general than the HP model, because it applies equally well to circular and linear chromosomes. However, the DCJ model still falls short in two respects. First, if the two cuts are in the same chromosome, one of the two nontrivial rejoinings causes a fission, creating a new circular chromosome; however, circular chromosomes do not normally arise in organisms with linear chromosomes, while most prokaryotic genomes consist of a single circular chromosome. This unrealistic operation can be corrected by forcing reabsorption of circular intermediates right after their introduction [23]. But this additional constraint creates dependencies among blocks of steps, which introduces difficulties in the estimation of the true distances (see [13]). Secondly, DCJ is a model of rearrangements: it does not take into account evolutionary events that alter the gene content and also, indirectly, the gene order, such as duplications and losses.

Genome evolution appears driven by very general mechanisms. For instance, for a wide variety of genomic properties, the number of families of a given size usually declines with the size of the family, following some asymptotic power law, the most common family size being one. Such scaling holds for gene families [7], protein folds and families (encoded in genomes) [12], and pseudogene families and pseudomotifs [15]. Several evolutionary models [5,7,17,22], all based on gene duplication, have been proposed to explain the observed biological data. More recently, Lynch [16] observed that the frequency distributions of family sizes observed in different species tend to bow downward rather than obey a power law. He gave a simple birth/death model to account for these observations. In this model, each gene (including duplicated ones) has pregeneration probability D (for duplication) of giving rise to a new copy, such that the average birth rate of a family of x members is Dx. The model also assumes that the presence of at least one member of the gene family is essential (i.e., complete loss of the gene family is not possible), but all excess copies have a probability L (for loss) of being eliminated. With D/L ratios consistent with actual estimates for eukaryotic genes, the equilibrium probability distribution of gene family sizes is close to the observations.

Our model of genomic evolution includes all of the operations from DCJ model, except the aforementioned operation that creates circular intermediates, and hence subsumes the HP model [6]; it also takes gene duplications and losses into account, all in a single step. The new evolutionary model respects the distinction between prokaryotic and eukaryotic genomes and also agrees with current predictions about genomic evolution, such as the distribution of sizes of gene families and the number of chromosomes in eukaryotic genomes. In earlier work, we described a method for estimating precisely true evolutionary distance between two genomes under this model using the independence among steps [14].

2.1 Genomes as Gene-Order Data

We denote the tail of a gene g by g^t and its head by g^h. We write $+g$ to indicate an orientation from tail to head ($g^t \rightarrow g^h$), $-g$ otherwise ($g^h \rightarrow g^t$). Two consecutive genes a and b can be connected by one *adjacency* of one of the following four types: $\{a^t, b^t\}$, $\{a^h, b^t\}$, $\{a^t, b^h\}$, and $\{a^h, b^h\}$. If gene c lies at one end of a linear chromosome, then we have a corresponding singleton set, $\{c^t\}$ or $\{c^h\}$, called a *telomere*. A *genome* can then be represented as a multiset of genes together with a multiset of adjacencies and telomeres. For example, the toy genome in Fig. 1, composed of one linear chromosome, ($+a$, $+b$, $-c$, $+a$, $+b$, $-d$, $+a$), and one circular one, ($+e$,$-f$), can be represented by the multiset of genes $\{a,a,a,b,b,c,d,e,f\}$ and the multiset of adjacencies and telomeres $\{\{a^t\}, \{a^h, b^t\}, \{b^h, c^h\}, \{c^t, a^h\}, \{a^h, b^t\}, \{b^h, d^h\}, \{d^t, a^h\}, \{a^h\}, \{e^h, f^h\}, \{e^t, f^t\}\}$. Because of the duplicated genes, there is no one-to-one correspondence between genomes and multisets of genes, adjacencies, and telomeres. For example, the genome composed of the linear chromosome $(+a, +b, -d, +a, +b, -c, +a)$ and the circular one $(+e, -f)$, would have the same multisets of genes, adjacencies and telomeres as that in Fig. 1.

2.2 Preliminaries on the Evolutionary Model

We use two parameters: the probability of occurrence of a gene duplication, p_d, and the probability of occurrence of a gene loss, p_l—the probability of occurrence of a rearrangement is then just $p_r = 1 - p_d - p_l$. The next event is chosen from the three categories according to these parameters.

For rearrangements, we select two elements uniformly *with replacement* from the multiset of all adjacencies and telomeres and then decide which rearrangement event we apply to these two elements. We have eight cases in all (refer to Fig. 2).

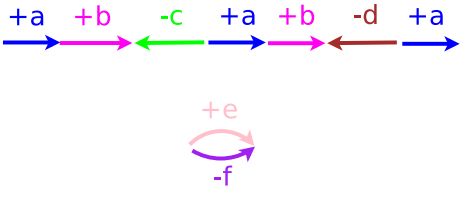


Fig. 1. A very small genome G

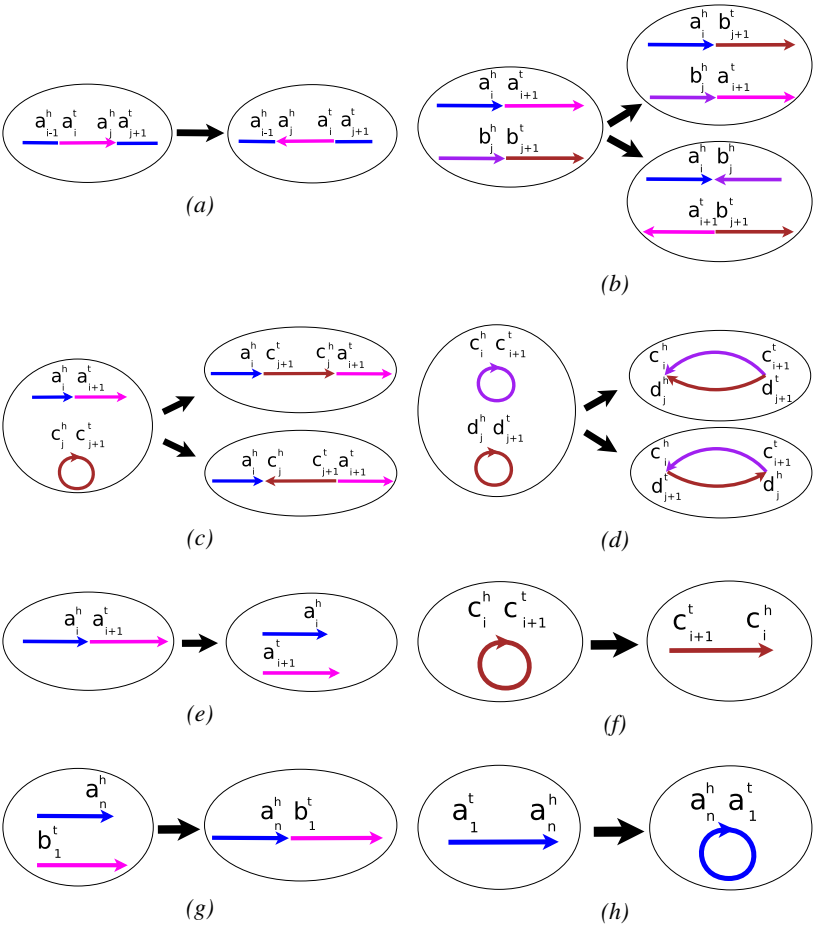


Fig. 2. Possible rearrangements

Select two different adjacencies, or one adjacency and one telomere, in the same chromosome (Fig. 2a). For example, select two different adjacencies $\{a^h_{i-1}, a^t_i\}$ and $\{a^h_j, a^t_{j+1}\}$ on one linear chromosome $A = (a_1 \ldots a_{i-1} a_i \ldots a_j a_{j+1} \ldots a_n)$. Reversing all genes between a_i and a_j yields $(a_1 \ldots a_{i-1}\text{-}a_j \ldots \text{-}a_i a_{j+1} \ldots a_n)$. Two adjacencies, $\{a^h_{i-1}, a^t_i\}$ and $\{a^h_j, a^t_{j+1}\}$, are replaced by two others, $\{a^h_{i-1}, a^h_j\}$ and $\{a^t_i, a^t_{j+1}\}$. This operation causes an inversion(another possible operation in DCJ model to creat a new circular chromosome is forbidden in our model).

Select two adjacencies, or one adjacency and one telomere, in two linear chromosomes (Fig. 2b). For example, select two adjacencies, $\{a^h_i, a^t_{i+1}\}$ from one linear chromosome $A = (a_1 \ldots a_i a_{i+1} \ldots a_n)$ and $\{b^h_j, b^t_{j+1}\}$ from another linear chromosome $B = (b_1 \ldots b_j b_{j+1} \ldots b_m)$. Now exchange the two segments between these two chromosomes C and D. There are two possible outcomes, $(a_1 \ldots a_i\ b_{j+1} \ldots b_m)$ and $(b_1 \ldots b_j\ a_{i+1} \ldots a_n)$ or $(a_1 \ldots a_i\ \text{-}b_j \ldots \text{-}b_1)$ and $(\text{-}b_n \ldots \text{-}b_{j+1}\ a_{i+1} \ldots a_n)$. Two ad-

jacencies, $\{a_i^h, a_{i+1}^t\}$ and $\{b_j^h, b_{j+1}^t\}$, are replaced by $\{a_i^h, b_{j+1}^h\}$ and $\{a_{i+1}^t, b_j^t\}$ or $\{a_i^h, b_j^h\}$ and $\{a_{i+1}^t, b_{j+1}^t\}$. This operation causes a translocation.

Select two different adjacencies, or one adjacency and one telomere, in one circular chromosome and one linear chromosome (Fig. 2c). For example, select two adjacencies, $\{a_i^h, a_{i+1}^t\}$ from one linear chromosome $A = (a_1 \dots a_i a_{i+1} \dots a_n)$ and $\{c_j^h, c_{j+1}^t\}$ from one circular chromosome $C = (c_1 \dots c_j c_{j+1} \dots c_m)$. Now merge the circular chromosome C into the linear chromosome A. There are two possible outcomes, linear chromosomes $(a_1 \dots a_i\ c_{j+1} \dots\ c_m c_1 \dots\ c_j a_{i+1} \dots a_n)$ or $(a_1 \dots a_i$ $\text{-}c_j \dots$ $\text{-}c_1 \text{-}c_m \dots$ $\text{-}c_{j+1}\ a_{i+1} \dots a_n)$. Two adjacencies, $\{a_i^h, a_{i+1}^t\}$ and $\{c_j^h, c_{j+1}^t\}$, are replaced by $\{a_i^h, c_{j+1}^h\}$ and $\{a_{i+1}^t, c_j^t\}$ or $\{a_i^h, c_j^h\}$ and $\{a_{i+1}^t, c_{j+1}^t\}$. This operation causes a fusion of a circular chromosome with a linear chromosome.

Select two adjacencies in two circular chromosomes (Fig. 2d). For example, select two adjacencies, $\{c_i^h, c_{i+1}^t\}$ from one circular chromosome $C = (c_1 \dots c_i c_{i+1} \dots c_m)$ and $\{d_j^h, d_{j+1}^t\}$ from another circular chromosome $D = (d_1 \dots d_j d_{j+1} \dots d_n)$. Now merge these two circular chromosomes C and D into one new circular chromosome. There are two possible outcomes, circular chromosomes $(c_1 \dots c_i\ d_{j+1} \dots$ $d_m d_1 \dots d_j\ c_{i+1} \dots c_m)$ or $(c_1 \dots c_i$ $\text{-}d_j \dots$ $\text{-}d_1 \text{-}d_m \dots$ $\text{-}d_{j+1}\ c_{i+1} \dots c_m)$. Two adjacencies, $\{c_i^h, c_{i+1}^t\}$ and $\{d_j^h, d_{j+1}^t\}$, are replaced by $\{c_i^h, d_{j+1}^h\}$ and $\{c_{i+1}^t, d_j^t\}$ or $\{c_i^h, d_j^h\}$ and $\{c_{i+1}^t, d_{j+1}^t\}$. This operation causes a fusion of two circular chromosomes.

Select the same adjacency twice in one linear chromosome (Fig. 2e). For example, select the adjacency $\{a_i^h, a_{i+1}^t\}$ twice from linear chromosome $A = (a_1 \dots a_i a_{i+1} \dots a_n)$. Then split C into two new linear chromosomes, $(a_1 \dots a_i)$ and $(a_{i+1} \dots a_n)$. The adjacency $\{a_i^h, a_{i+1}^t\}$ is replaced by two telomeres $\{a_i^h\}$ and $\{a_{i+1}^t\}$. This operation causes a fission of a linear chromosome.

Select the same adjacency twice in one circular chromosome (Fig. 2f). For example, select the adjacency $\{c_i^h, c_{i+1}^t\}$ twice from circular chromosome $C = (c_1 \dots$ $c_i c_{i+1} \dots c_m)$. Then linearize C into a linear chromosome, $(c_{i+1} \dots c_m c_1 \dots c_i)$. The adjacency $\{c_i^h, c_{i+1}^t\}$ is replaced by two telomeres $\{c_i^h\}$ and $\{c_{i+1}^t\}$. This operation causes a linearization of a circular chromosome.

Select two telomeres in two linear chromosomes (Fig. 2g). For example, select telomeres $\{a_n^h\}$ and $\{b_1^t\}$ from two different linear chromosomes $A = (a_1 \dots a_i a_{i+1} \dots a_n)$ and $B = (b_1 \dots b_j b_{j+1} \dots b_m)$. Then concatenate these two linear chromosomes into a single new chromosome $(a_1 \dots\ a_i a_{i+1} \dots\ a_n\ b_1 \dots\ b_j b_{j+1} \dots\ b_m)$. Two telomeres, $\{a_n^h\}$ and $\{b_1^t\}$, are replaced by one adjacency $\{a_n^h, b_1^t\}$. This operation causes a fusion of two linear chromosomes.

Select two telomeres in one linear chromosome (Fig. 2h).[1] For example, select telomeres $\{a_1^t\}$ and $\{a_n^n\}$ from linear chromosome $A = (a_1 \dots a_i a_{i+1} \dots a_n)$ (See Fig. 2h). Then circularize the linear chromosome by connecting its two ends. Two telomeres, $\{a_1^t\}$ and $\{a_n^h\}$, are replaced by by one adjacency, $\{a_1^t, a_n^h\}$. This operation causes a circularization of a linear chromosome.

As mentioned earlier, we do not include a fission that creates a circular intermediate. This decision is based on outcomes, not a mechanism; as is the DCJ model itself:

[1] Selecting one telomere twice is assimilated to selecting both telomeres of the linear chromosome.

that is, the model operations may or may not correspond to actual evolutionary events, but running the model produces simulated genomes that more closely resemble actual genomes.

For gene duplication, we uniformly select a position to start duplicating a short segment of chromosomal material and place the new copy to a new position within the genome. We set L_{max} as the maximum number of genes in the duplicated segment and assume that the number of genes in that segment is a uniform random number between 1 and L_{max}. For example, select one segment $a_{i+1} \ldots a_{i+L}$ to duplicate and insert the copy between one adjacency $\{b^h_j, b^t_{j+1}\}$. Such an operation duplicates L genes and $L-1$ adjacencies, removes one adjacency, and adds two new adjacencies; thus genes a_{i+1}, …, a_{i+L-1} and a_{i+L} are added to the multiset of genes, the adjacency $\{b^h_j, b^t_{j+1}\}$ is removed, and $L+1$ new adjacencies, $\{b^h_j, a^t_{i+1}\}$, $\{a^h_{i+1}, a^t_{i+2}\}$, …, $\{a^h_{i+L}, b^t_{j+1}\}$, are added. For gene loss, we uniformly select one gene from the set of all candidate genes and delete it, restricting gene loss to the deletion of a single gene copy at a time, following Lynch [16]. For example, if we delete gene a_i in the chromosome $(\ldots a_{i-1} a_i a_{i+1} \ldots)$, one copy of a_i is removed from the multiset of genes, while two adjacencies, $\{a^h_{i-1}, a^t_i\}$ and $\{a^h_i, a^t_{i+1}\}$, are replaced by one adjacency, $\{a^h_{i-1}, a^t_{i+1}\}$.

3 Results

3.1 Model Restricted to Rearrangements

Edit distance computation

The edit distance between two genomes is the minimum number of allowed evolutionary operations necessary to transform one genome into the other.

Consider two genomes with equal gene content and no duplicate genes. If both genomes consist of only linear chromosomes, the model of Hannenhalli and Pevzner [6] allows the computation of the edit distance under inversions, translocations, fusions, and fissions, hereafter the *HP distance*. The edit distance in our model can be no larger than the HP distance, since our model includes all operations in the HP model and more (circularizations and linearizations).

In fact, the two edit distances are equal for genomes composed of only linear chromosomes. Suppose there are intermediate circular chromosomes in some sorting path in our model; then we can always find pairs of operations, one to circularize a linear chromosome and the other to linearize that circular chromosome, that can be replaced by a fission and a fusion in the HP model. So any optimal sorting path in our model can be transformed into an optimal sorting path of equal length in the HP model; therefore the edit distance in our model is equal to the HP distance—although the number of optimal sorting paths may be different.

If two genomes have both linear and circular chromosomes, the edit distance in our model can be no smaller than the DCJ distance [2], since the DCJ model includes all operations in our model and more. Bergeron *et al.* [3] gave a linear-time algorithm to compute the extra cost of not resorting to "forbidden" DCJ operations to compute the HP distance; their algorithm also applies to our model. Thus for any two genomes with equal gene content and no duplicate genes, the edit distance in our model can be computed in linear time.

3.2 Genome Structure Prediction

In this section, we prove that our new model respects the distinction between eukaryotic and prokaryotic genomes.

Theorem 1. *Let the ancestral genome have one circular chromosome with n genes. After $O(n)$ rearrangements events, with probability $1-n^{-O(1)}$, the final genome contains a single circular chromosome or a collection of $O(\log n)$ linear chromosomes.*

Proof. We examine the effect of rearrangements on the genome structure. Given the original genome with one circular chromosome, only one of our eight cases can result in a linearization: *select the same adjacency twice* (Fig. 2f). Once we have only linear chromosomes, two cases can directly result in a change in the number of linear or circular chromosomes: *select the same adjacency twice* (Fig. 2e) and *select two telomeres* (Fig. 2h). The probability for selecting the same adjacency twice is $O(1/n)$; that for selecting two telomeres is $O(t^2/n^2)$, where t is the number of telomeres. Every time we select the same adjacency twice, we increase the number of linear chromosomes by 1. Let the indicator variable X_i represent whether or not we select the same adjacency twice at the ith step and write k for the number of evolutionary events. Set $X = \sum_{i=1}^{k} X_i$ and let μ be the expectation of X. The Chernoff bound shows

$$Pr(X > (1+\delta)\mu) < \left(e^{\delta}/(1+\delta)^{1+\delta}\right)^{\mu}$$

In our case, $k = O(n)$, $\mu = O(1)$, $\delta = O(logn)$, so that we get

$$Pr(X > O(\log n)) < n^{-O(1)}$$

Let the indicator variable Y_i represent whether or not we select two telomeres at the ith step. Since $t = 2X$, t is bounded by $O(\log n)$ with probability $1-n^{-O(1)}$. Thus, with probability $1-n^{-O(1)}$, we have

$$Pr(Y_i = 1) < O((\log n)^2/n^2),$$

Now set $Y = \sum_{i=1}^{k} Y_i$. We have

$$Pr(Y > 0) \leqslant \sum_{i=1}^{k} Pr(Y_i = 1) < n^{-O(1)}.$$

Overall, then, with probability $1-n^{-O(1)}$, $X < O(\log n)$ and $Y = 0$, which means that the final genome structure has either a collection of $O(\log n)$ linear chromosomes or a single circular chromosome. □

Thm. 1 tells us that, if the original genomic structure starts from a circular chromosome, most current genomes will contain a single circular chromosome or a collection of linear chromosomes. However, if the initial genome structure was, e.g., a mix of linear and circular chromosomes, would such a structure be stable through evolution? We can characterize all stable structures in our model under some mild conditions.

Theorem 2. *Let the ancestral genome have n genes and assume that there are positive constants c_1 and α such that each chromosome in the ancestral genome has at least $c_1 n^{\alpha}$ genes. Let c_2 be some constant obeying $c_2 > 2c_1$. After $c_2 n^{1-\alpha}\log n$ rearrangements, with probability $1 - O(n^{-\alpha}\log n)$, the final genome contains either a single circular chromosome or a collection of linear chromosomes.*

Proof. In our evolutionary model, consider the case of selecting two adjacencies or one adjacency and one telomere in two different chromosomes. If one of the two chromosomes is circular, a fusion will merge the circular chromosome into the linear chromosome (Fig. 2c). If both chromosomes are circular, a fusion will merge the two chromosomes into a single circular chromosome (Fig. 2d). We use a graph representation, G, for the genome structure, where each circular chromosome is represented by a vertex A_i and all of the linear chromosomes (if any) are represented by a single vertex B . In the evolutionary process, if two adjacencies or one adjacency and one telomere are selected in two different chromosomes, connect the vertices of these two chromosomes. We first ignore circularizations of linear chromosomes (Fig. 2h), then the genome ends up with a single circular chromosome or a collection of linear chromosomes if and only if the corresponding graph G is connected finally. We therefore bound the probability that the graph G is not connected after $c_2 n^{1-\alpha}\log n$ rearrangements. If G is not connected, there is at least one bipartition of the vertices into S_1 and S_2 in which no edge has an endpoint in each subset. Assume there are g_1 and g_2 genes in S_1 and S_2, respectively; then $min\{g_1, g_2\} \geqslant c_1 n^{\alpha}$ and $g_1 + g_2 = n$. Since there are at most $\frac{1}{c_1} n^{1-\alpha}$ chromosomes, we can write

$$\begin{aligned} Pr(G \text{ is not connected}) &< \binom{\binom{g_1}{2} + \binom{g_2}{2}}{c_2 n^{1-\alpha}\log n} \Big/ \binom{\binom{g_1+g_2}{2}}{c_2 n^{1-\alpha}\log n} \\ &< (1 - c_1 n^{1-\alpha})^{c_2 n^{1-\alpha}\log n} < O(n^{-2\alpha}) \end{aligned}$$

Let indicator variable X_i represent whether or not we select the same adjacency twice at the ith step (Fig. 2e,f) and set $X = \sum_{i=1}^{c_2 n^{1-\alpha}\log n} X_i$. We have

$$\begin{aligned} Pr(X_i = 1) &\leqslant 1/n \\ Pr(X > 0) &\leqslant \sum_{i=1}^{c_2 n^{1-\alpha}\log n} Pr(X_i = 1) = O(n^{-\alpha}\log n). \end{aligned}$$

Now we bound the probability of selecting two telomeres in the same linear chromosome (Fig. 2h), which causes circularization of this chromosome. For each linear chromosome, there are four possible ways of selecting two corresponding telomeres. Since the number of linear chromosomes l is bounded by $\frac{1}{c_1} n^{1-\alpha}$, there are at most $\frac{4}{c_1} n^{1-\alpha}$ ways to circularize one linear chromosome in all $(n+l)^2$ ways of selecting two adjacencies or telomeres. Again, let indicator variable Y_i represent circularization of one linear chromosome at the ith step and set $Y = \sum_{i=1}^{c_2 n^{1-\alpha}\log n} Y_i$. We have

$$\begin{aligned} Pr(Y > 0) &\leqslant \sum_{i=1}^{c_2 n^{1-\alpha}\log n} Pr(Y_i = 1) \\ &\leqslant 4c_2 \log n / c_1 n^{2\alpha} < O(n^{-2\alpha}\log n) \end{aligned}$$

Thus, with probability $1 - O(n^{-\alpha}\log n)$, we have: G is connected, $X = 0$, and $Y = 0$, so that the final genome contains either a single circular chromosome or a collection of linear chromosomes. □

The restriction on the minimum size of chromosomes in the ancestral genomes is very mild, since the parameter α can be arbitrarily small.

Our model also predicts, for genomes composed of a collection of linear chromsomes, convergence to a certain number of chromosomes, which depends on the total number of genes.

Theorem 3. *Assume there are n genes and fewer than* $\frac{1+\sqrt{1+4n}}{2}$ *linear chromosomes in the original genome. The number of linear chromosomes increases during rearrangements, converging to* $\frac{1+\sqrt{1+4n}}{2}$.

Proof. Assume there are l linear chromosomes in the original genome. In our model, the number of linear chromosomes increases by 1 with probability $\frac{1}{n+l}$ and decreases by 1 with probability $(\frac{l}{n+l})^2$. Since we have $l < \frac{1+\sqrt{1+4n}}{2}$, an increase is more likely. The stable equilibrium follows from the equation $\frac{1}{n+l} = (\frac{l}{n+l})^2$. □

These theorems are not affected by duplications and losses, as long as the latter are reflected in the sizes of chromosomes and the total number of genes.

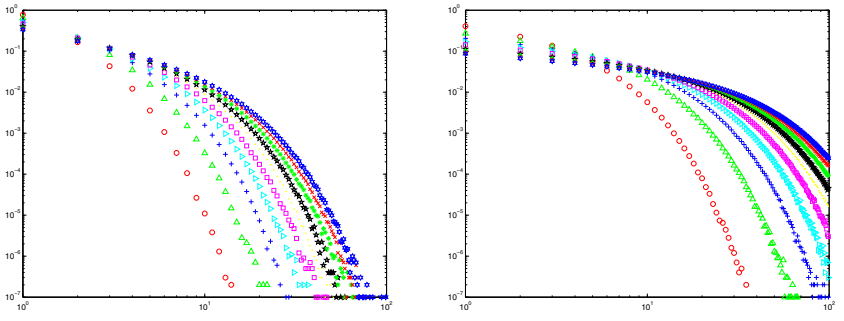
3.3 Sizes of Gene Families

Of most concern in a duplication and loss model is the distribution of the sizes of the gene families, since that is one of the few aspects of the process that has been observed to obey general laws. Our sole aim in this section is to demonstrate through simulations that our model, which uses the duplication/loss model of Lynch, yields distributions consistent with what Lynch suggested [16].

Our experiments start with a genome with no duplicated genes. This genome is then subjected to a prescribed number k, varying from from 0 to 10 times the number of genes, of evolutionary events chosen according to p_d and p_l to obtain different genomes G^k. We test a large number of different choices of parameters on varying sizes of genomes; as the results are consistent throughout, we report two cases: (a) 1'000 genes with $L = 10$, $p_d = 0.2$, and $p_l = 0.8$; and (b) 10'000 genes with $L = 10$, $p_d = 0.4$, and $p_l = 0.6$. The data in Fig. 3 summarizes 1'000 runs for each parameter setting. (The parameters chosen correspond to those used in our distance estimation model [14].) The shape of the distributions of gene family sizes is generally similar to the observations presented by Lynch [16].

4 Discussion and Conclusions

Thm. 1 and Thm. 2 together show that our model respects the distinction between the organization of most prokaryotic genomes (one circular chromosome) and that of most eukaryotic genomes (multiple linear chromosomes). In contrast, the HP model [6] deals



(a) 1'000 genes *(b) 10'000 genes*

Fig. 3. Probability distribution of the size of gene families, for various numbers of events, increasing from the leftmost ($\#events = \#genes$) to the rightmost ($\#events = 10 \times \#genes$)

with only linear chromosomes, while the DCJ model [2,23] (assuming uniform distribution of all possible DCJ operations) predicts that over half of modern genomes consisting of only circular chromosomes will have more than one circular chromosome.

There is evidence about the linearization of circular chromosomes during bacterial evolution [21] and the increase in the number of chromosomes of eukaryotic groups by centric fission [8,9], both of which accord with Thm. 3. According to the minimum interaction theory of Imai *et al.* [10], genome evolution in eukaryotes proceeds as a whole toward increasing the number of chromosomes. Their theory predicts that the highest number of chromosomes in mammals should be 166, while their simulations yield a range of 133–138 for this number [11]. The latter range agrees with our model (as well as the models in [6,2,23], if we assume that the two cuts are uniformly selected) if the number of genes is around 20'000, a fairly typical value for mammals.

Fig. 3 shows that our model of gene duplications and losses readily generates distributional forms close to the observations presented by Lynch [16]. Different parameters for gene duplications and losses, and the number of evolutionary events, influence the the distributions of gene family sizes: such information can help us improve the estimation of the actual number of evolutionary events as well as infer the parameters for duplications and losses in our model [14].

According to our model, more rearrangements, gene duplications, and gene losses will linearize circular chromosomes, increase the number of linear chromosomes, and increase the number of genes—i.e., will favor a shift from a prokaryotic architecture to a eukaryotic one. However prokaryotic architectures exist in large numbers today. The reason is to be found in population sizes. In a large population, as with most prokaryotic organisms, most alleles are likely to be eliminated by purifying selection, whereas, in a small population, neutral or even deleterious mutations can be fixated more easily. Thus many forms of mutant alleles that are able to drift to fixation in multicellular eukaryotes are eliminated by purifying selection in prokaryotes as population sizes decreased dramatically in the transition from prokaryotes to multicellular eukaryotes [16]. Similarly, the fixation of rearrangements, gene duplications, and gene losses (all "rare genomic events" [18]) in prokaryotic species is also more difficult compared to that

in eukaryotes. Thus, in our model, prokaryotes tend to have one circular chromosome and a small number of genes, while eukaryotes tend to have multiple linear chromosomes and a large number of genes, in response to a reduction in purifying selection. Our model of gene rearrangement, duplication, and loss is the first to give rise naturally to such a structure; and it does so independently of the choice of parameters, which influence only the tapering rate of the size of gene families.

References

1. Bader, D.A., Moret, B.M.E., Yan, M.: A fast linear-time algorithm for inversion distance with an experimental comparison. J. Comput. Biol. 8(5), 483–491 (2001)
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
3. Bergeron, A., Mixtacki, J., Stoye, J.: A new linear time algorithm to compute the genomic distance via the double cut and join distance. Theor. Computer Science 410(51), 5300–5316 (2009)
4. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. MIT Press, Cambridge (2009)
5. Hahn, M.W., De Bie, T., Stajich, J.E., Nguyen, C., Cristianini, N.: Estimating the tempo and mode of gene family evolution from comparative genomic data. Genome Research 15(8), 1153–1160 (2005)
6. Hannenhalli, S., Pevzner, P.A.: Transforming mice into men (polynomial algorithm for genomic distance problems). In: Proc. 36th IEEE Symp. Foundations of Comput. Sci. (FOCS 1995), pp. 581–592. IEEE Computer Society Press, Piscataway (1995)
7. Huynen, M.A., van Nimwegen, E.: The frequency distribution of gene family sizes in complete genomes. Mol. Biol. Evol. 15(5), 583–589 (1998)
8. Imai, H.T.: On the origin of telocentric chromosomes in mammals. J. Theor. Biol. 71(4), 619–637 (1978)
9. Imai, H.T., Crozier, R.H.: Quantitative analysis of directionality in mammalian karyotype evolution. American Naturalist 116(4), 537–569 (1980)
10. Imai, H.T., Maruyama, T., Gojobori, T., Inoue, Y., Crozier, R.H.: Theoretical bases for karyotype evolution. 1. the minimum-interaction hypothesis. American Naturalist 128(6), 900–920 (1986)
11. Imai, H.T., Satta, Y., Wada, M., Takahata, N.: Estimation of the highest chromosome number of eukaryotes based on the minimum interaction theory. J. Theor. Biol. 217(1), 61–74 (2002)
12. Koonin, E.V., Wolf, Y.I., Karev, G.P.: The structure of the protein universe and genome evolution. Nature 420, 218–223 (2002)
13. Lin, Y., Moret, B.M.E.: Estimating true evolutionary distances under the dcj model. In: Proc. 16th Conf. Intelligent Systems for Mol. Biol. (ISMB 2008). Bioinformatics, vol. 24(13), pp. i114–i122 (2008)
14. Lin, Y., Rajan, V., Swenson, K.M., Moret, B.M.E.: Estimating true evolutionary distances under rearrangements, duplications, and losses. In: Proc. 8th Asia Pacific Bioinformatics Conf, APBC 2010 (accepted, to appear, 2010)
15. Luscombe, N.M., Qian, J., Zhang, Z., Johnson, T., Gerstein, M.: The dominance of the population by a selected few: power-law behaviour applies to a wide variety of genomic properties. Genome Biology 3(8) (2002)
16. Lynch, M.: The Origins of Genome Architecture. Sinauer (2007)

17. Qian, J., Luscombe, N.M., Gerstein, M.: Protein family and fold occurrence in genomes: power-law behavior and evolutionary model. J. Mol. Biol. 313, 673–681 (2001)
18. Rokas, A., Holland, P.W.H.: Rare genomic changes as a tool for phylogenetics. Trends in Ecol. and Evol. 15, 454–459 (2000)
19. Sturtevant, A.H.: A case of rearrangement of genes in Drosophila. Proc. Nat'l Acad. Sci., USA 7, 235–237 (1921)
20. Sturtevant, A.H., Dobzhansky, T.: Inversions in the third chromosome of wild races of Drosophila pseudoobscura, and their use in the study of the history of the species. Proc. Nat'l Acad. Sci., USA 22, 448–450 (1936)
21. Volff, J.-N., Altenbuchner, J.: A new beginning with new ends: linearisation of circular chromosomes during bacterial evolution. FEMS Microbiol. Lett. 186, 143–150 (2000)
22. Yanai, I., Camacho, C.J., DeLisi, C.: Predictions of gene family distributions in microbial genomes: evolution by gene duplication and modification. Phys. Rev. Lett. 85(12), 2641–2644 (2000)
23. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21(16), 3340–3346 (2005)

A Simulation Tool for the Study of Symmetric Inversions in Bacterial Genomes

Ulisses Dias[1,2,⋆], Zanoni Dias[1,⋆], and João C. Setubal[3]

[1] Institute of Computing
Unicamp - Campinas - SP - Brazil
{udias,zanoni}@ic.unicamp.br
[2] Virginia Bioinformatics Institute
Virginia Tech - Blacksburg - VA - USA
udias@vbi.vt.edu
[3] Virginia Bioinformatics Institute and Department of Computer Science
Virginia Tech - Blacksburg - VA - USA
setubal@vbi.vt.edu

Abstract. We present the tool SIB that simulates genomic inversions in bacterial chromosomes. The tool simulates symmetric inversions but allows the appearance of nonsymmetric inversions by simulating small syntenic blocks frequently observed on bacterial genome comparisons. We evaluate SIB by comparing its results to real genome alignments. We develop measures that allow quantitative comparisons between real pairwise alignments (in terms of dotplots) and simulated ones. These measures allow an evaluation of SIB in terms of dendrograms. We evaluate SIB by comparing its results to whole chromosome alignments and maximum likelihood trees for three bacterial groups (the *Pseudomonadaceae* family and the *Xanthomonas* and *Shewanella* genera). We demonstrate an application of SIB by using it to evaluate the ancestral genome reconstruction tool MGR.

1 Introduction

Genome rearrangements are one of the most important large-scale evolutionary phenomena in genomes. In bacteria, inversions have long been recognized as one of the most frequently observed rearrangements. In particular, inversions symmetric to the origin of replication (meaning that the endpoints of the inversion are equally distant from the origin of replication) have been proposed as the primary mechanism that explains the 'X' patterns that are frequently seen when two circular chromosomes of closely related species are aligned [3]. A recent study using *Yersinia* genomes [2] has added evidence that such symmetric inversions are "over-represented" with respect to other kinds of inversions.

An emerging area in the computational analysis of evolutionary phenomena is that of ancestral genome reconstruction. Several programs have been proposed

⋆ This work is partially sponsored by CNPq (472504/2007-0, 479207/2007-0 and 483177/2009-1), CAPES (BEX 1757/09-1), and FAPESP (2007/05574-4).

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 240–251, 2010.

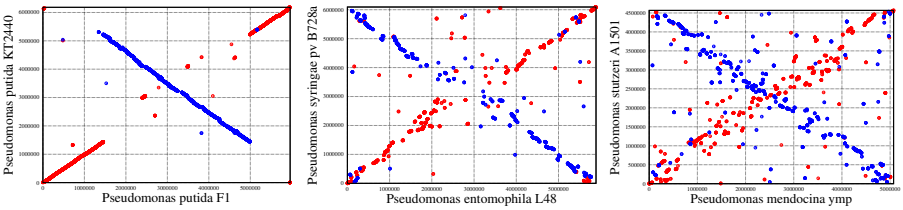


Fig. 1. MUMmer pairwise alignment of *Pseudomonadaceae* chromosomes. Dots represent matches between the chromosome sequences. Red dots depict matches in the same orientation in both chromosomes, whereas blue dots depict matches in opposite orientation.

such as BPAnalysis [12], GRAPPA [9], BADGER [7], and MGR [1]. Because generally we do not have access to ancestral genomes, in order to evaluate whether a reconstruction is good or not simulation tools are essential. In this paper we propose a novel simulation tool for the study of symmetric inversions in bacterial circular chromosomes; we call our tool SIB.

A simulation tool such as the one we propose evidently needs to model the evolution of bacterial genomes. This modeling is difficult for several reasons: in spite of the apparent predominance of symmetric inversions, bacterial genomes undergo other kinds of rearrangements, as well as insertions, deletions, and duplications. Moreover, genome alignments of different groups of bacteria reveal differing patterns of rearrangements. For these reasons we decided to create a simulation tool whose aim is to reproduce the 'X' pattern in alignments more clearly observed in certain bacterial groups. We chose the following groups as a basis for our model: the *Pseudomonadaceae* family, the *Xanthomonas* genus, and the *Shewanella* genus, all of them part of the γ-proteobacteria taxonomical class. Another criterion we used in this choice was the availability of reasonably distinct fully sequenced genomes in each group. Figure 1 shows examples of genome alignments for members of the *Pseudomonadaceae* family.

2 Pairwise Chromosome Comparisons and Dotplot Measures

Table 1 contains information about the genome groups we consider in this study. All genomes in this list have just one circular chromosome. Some members have small (less than 100 kbp) plasmids; we do not take plasmids into consideration.

For pairwise comparisons of chromosomes we used the tool MUMmer [6]. MUMmer searches for maximal exact matches between two DNA sequences. It is possible to plot the results using a standard 'dotplot' format (see Figure 1).

In order to characterize in a quantitative fashion a pairwise alignment from the point of view of symmetric inversions our primary reference is the disposition

Table 1. Bacterial groups used in this work. In each group all fully sequenced genomes available from GenBank as of May, 2010 were used, with the exception of *Cellvibrio japonicus* Ueda107 (NC_010995.1). Even though the NCBI taxonomy places it under the *Pseudomonadaceae* family, the pairwise comparisons we obtained suggest that it is (relatively) very distant from the other members: its MUMmer alignment to other members results in too few matches.

Bacterial group	number of genomes	Chromosome size range (Mbp)
Pseudomonadaceae	18	4.6 – 7.1
Xanthomonas	9	3.8 – 5.2
Shewanella	9	4.3 – 5.9

of matches in these dotplots. The dotplots show that most of the matches occur near the primary diagonal ($y = x$) or near the secondary anti-diagonal ($y = L-x$, where L is the chromosome length) as shown in Figure 1. The basic explanation we use for these patterns is the one proposed by Eisen et al. [3].

In the genome rearrangement literature that focuses on mathematical models of genomes, a chromosome is usually represented as a permutation $\pi = (\pm\pi_1 \ \pm\pi_2 \ \ldots \ \pm\pi_n)$, for $\pi_i \in \mathbb{I}$, $0 < \pi_i \leq n$ and $i \neq j \leftrightarrow \pi_i \neq \pi_j$, where each π_i represents a gene, and that gene is assumed to be shared by the genomes being compared, with n being the total number of genes shared. Our inputs are DNA sequences, not strings of genes; on the other hand, our primary reference for modeling purposes are the dotplots, thus we also need to deal with discrete entities present in chromosomes, but not necessarily genes. We handle this situation by converting MUMmer matches into *points* in the following way. Because bacterial genomes have on average one gene for every 1,000 bp (or 1 kbp), we split all matches reported by MUMmer into 1 kbp segments (and each of them becomes a point). Matches that are at least 0.5 kbp long are considered to have 1 kbp. Matches shorter than 0.5 kbp are discarded. Another simplification we adopt is to discard duplications (matches have to be unique). Thus, the dotplots on which we based the evolutionary model of SIB are the ones that result from this process (*processed dotplots*). The removal of duplicated matches is not a concern for the genomes we studied. The duplication removal step did not remove more than 10 elements in every pairwise comparison we analyzed.

Eisen et al. [3] proposed a measure for MUMmer alignment plots to determine if faint 'X' patterns were stastistically significant or not. Their measure takes into account the density of points near the two main diagonals. In order to compare simulated scenarios with our processed dotplots originating from real data we developed measures that also take into account density of points near the two main diagonals. We define three regions of interest. Region 1 is the strip along the main diagonal defined by two parallel diagonals, one on each side. We define similarly a Region 2 to be the analogous strip for the anti-diagonal. We require that each of these strips contain 25% of the total dotplot area. We further

define Region 3 to be everything that is outside Regions 1 and 2. We define density as the ratio between the number of points found in a region to the total number of points present.

MUMmer results in two kind of matches: *forward matches*, in which regions on the plus strands of both chromosomes are matched, and *reverse matches*, in which a region in the plus strand of one chromosome is matched to a region in the minus strand in the other chromosome. In Figure 1 forward matches are colored red, and reverse matches are colored blue. Thus we define d_1 to be the density of points of the forward kind that are in Region 1; d_2 to be the density of points of the reverse kind that are in Region 2; and d_3 to be the density of points of both kinds that are in Region 3.

We have used the densities d_1, d_2 and d_3 to compare processed dotplots to dotplots resulting from simulations, as described next.

3 The Simulation Tool

The evolutionary model that is built-in in SIB resulted from careful observation of pairwise comparisons between *Pseudomonadaceae* chromosome sequences. We now explain the details of this model.

Inversions. Although symmetric inversions seem to be the dominant genomic rearrangement event in bacteria, it is known that nonsymmetric inversions can also happen. There are two kinds of nonsymmetric inversions: one that spans the origin of replication and one that does not span the origin. Darling et al. [2] call these *inter-replichore inversions* and *intra-replichore inversions*, respectively. In results presented here we have used only symmetric inversions, but nonsymmetric inter-replichore inversions can be generated because of *unbreakable segments* (see below).

For symmetric inversions, SIB has as a parameter the distance between the endpoints and the origin, *the inversion length*. We model these lengths by a normal distribution, $N_\ell(\mu_\ell, \sigma_\ell)$, where μ_ℓ is the mean length and σ_ℓ is its standard deviation. In the simulations presented here we have used the empirically derived values of $\mu_\ell = n/3$ and $\sigma_\ell = n/5$, where n is the number of points in the simulation.

Unbreakable segments. Another fact that emerges from chromosome sequence comparisons of bacterial species is that even widely separated species will generally contain small *syntenic blocks*, that is, groups of genes (generally between 2 to 10) whose order is conserved across species [18,5,14]. We model this phenomenon by determining that each genome contains consecutive points that form an "unbreakable segment". When a symmetric inversion endpoint is inside an unbreakable segment, we create an asymmetry by extending the endpoint to the border of the segment in order to include the segment as a whole. The (desirable) side effect of this feature is that some points, after the inversion, will appear in areas outside the main diagonal strips.

We model number and size of unbreakable segments by a normal distribution $N_{\text{un}}(\mu_{\text{un}}, \sigma_{\text{un}})$, where μ_{un} is the mean size of an unbreakable segment and σ_{un} is the respective standard deviation. We have used the values $\mu_{\text{un}} = 3$ and $\sigma_{\text{un}} = 10$. These values were derived from the *Pseudomonadaceae* processed dotplots.

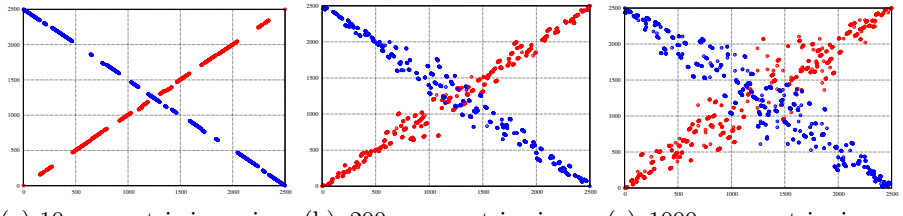
Number of points. The number of points is also a parameter of the simulation. For the results we present we have adopted the value 2,500, which is (in round numbers) the average number of points we have observed in processed *Pseudomonadaceae* dotplots.

Simulated scenarios. SIB can simulate the evolution of all nodes in a tree whose topology is provided by the user using the *Newick* format. Parameters that control these evolutionary scenarios are: the depth of the tree and the number of inversions between a parent node and its children.

4 Comparison between Real Dotplots and Simulations

For the purposes of comparing our simulated results with processed dotplots, we create a *simulation scenario* in the following way. It is a 2-branch simulation, and the number of generations between the root and the two leaf nodes is 4,000. Between any two levels (two generations) 5 inversions take place on each branch. Thus we can obtain a pairwise comparison of the left and right branches at any level along the simulation. Because the simulation incorporates pseudo-randomness in the lengths of inversions and size of unbreakable segments, each simulation run with the same parameter values can produce a different result (and hence a different scenario).

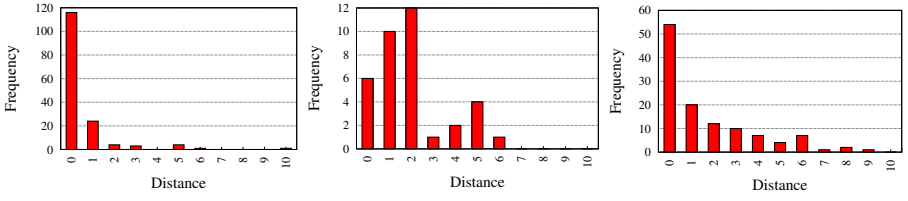
We have carried out all pairwise MUMmer comparisons between all pairs of genomes in each of the three bacterial groups of Table 1. Each one of these dotplots was converted to a point in 3-dimensional space given by the values of d_1, d_2 and d_3 (defined in the previous section). Each level in a simulated scenario also results in a point in 3D space. We can thus determine the closest simulated dotplot to any given processed dotplot by simply finding the simulated dotplot with smallest Euclidean distance to the processed dotplot.



(a) 10 symmetric inversions (b) 200 symmetric inversions (c) 1000 symmetric inversions

Fig. 2. Simulated dotplots generated by SIB

In Figure 2 we show three dotplots obtained from one of these scenarios. Using the method described in the previous paragraph we associate each processed dotplot to the closest simulated dotplot obtained from 10 different simulated scenarios. In Figure 3 we show the resulting histograms of distances for each of the three bacterial groups. The best histogram is the one for the *Pseudomonadaceae*, since the vast majority of distances was zero. The histograms for the other two groups were not as good, but most distances obtained were small.



(a) *Pseudomonadaceae* (b) *Xanthomonas* (c) *Shewanella*

Fig. 3. Histograms of Euclidean distances between processed dotplots and the closest simulated dotplot

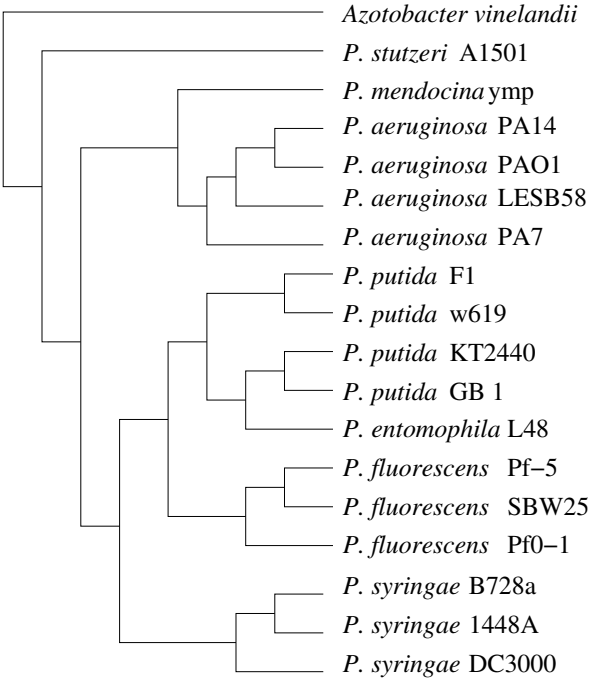


Fig. 4. *Pseudomonadaceae* dendrogram

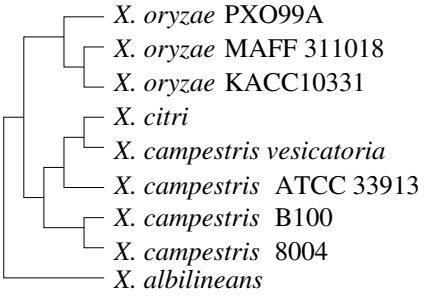


Fig. 5. *Xanthomonas* dendrogram

As another test of our simulations we inferred a dendrogram from simulated data in the following way. The association between a processed dotplot and its closest simulated dotplot allows us to associate each processed dotplot with the number of inversions used to generate the simulated dotplot that is closest. That number of inversions can be seen as a distance between the two chromosomes that originated the processed dotplot. Using the resulting distance matrix as input for the program `neighbor` (which implements the neighbor-joining algorithm [11]) available in the package PHYLIP [4] we generated a dendrogram for each of the three bacterial groups. These dendrograms are shown in Figures 4, 5 and 6.

Are these dendrograms good or not? We have compared each to the best phylogenetic tree that we could find for each of the three bacterial groups. These trees are shown in Figures 7, 8, and 9 (sources are given in figure legends). We show only the topology of each tree. These three trees were generated by maximum likelihood methods using concatenation of protein sequences. Note also that the sets of genomes in the *Pseudomonadaceae* and *Xanthomonas* trees are not exactly the same as the sets we used (more genomes have become available since the *Pseudomonadaceae* tree was published [13] and the published *Xanthomonas* tree [8] contains two unfinished genomes). A comparison between each dendrogram and its respective tree shows that we obtained best results for

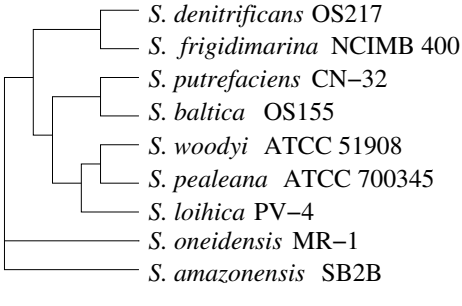


Fig. 6. *Shewanella* dendrogram

the *Pseudomonadaceae* family. The discrepancies we observed include: the *fluorescens* clade grouped with the *putida* clade instead of the *syringae* clade; and *P. mendocina* paired with the *aeruginosa* clade, rather than being an outgroup for all except *P. stutzeri* and *A. vinelandii*. The *Xanthomonas* comparison shows a discrepancy in the grouping of the *X. citri* clade with the *campestris* clade rather than the *oryzae* clade. Finally a noteworthy discrepancy in the *Shewanella* dendrogram is that *S. oneidensis* was placed at the base of the tree.

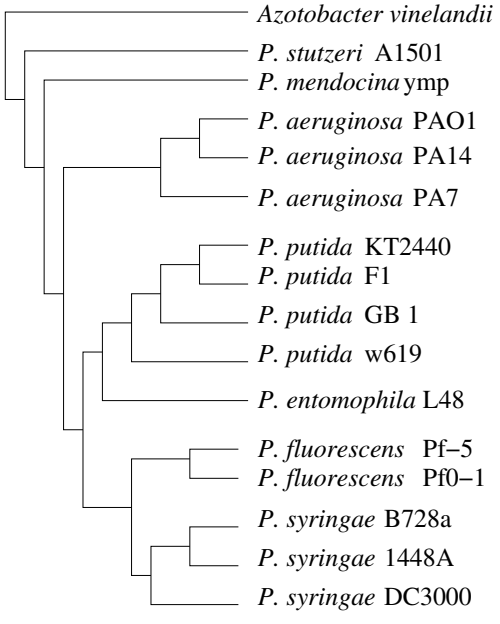


Fig. 7. *Pseudomonadaceae* phylogenetic tree. Adapted from [13]

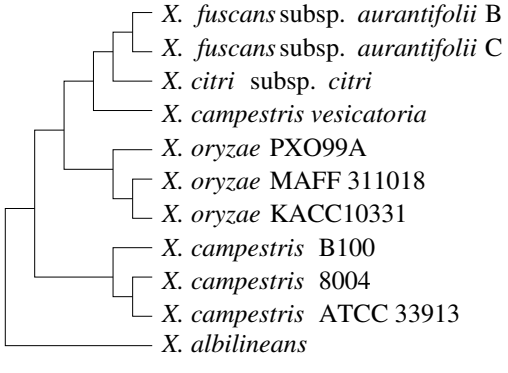


Fig. 8. *Xanthomonas* phylogenetic tree. Adapted from [8]

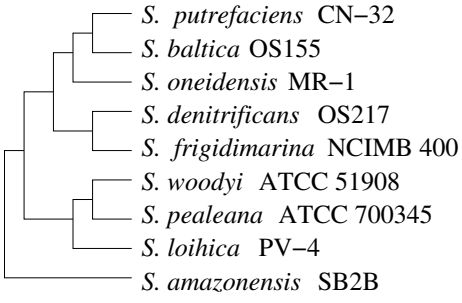


Fig. 9. *Shewanella* phylogenetic tree. Adapted from [17]

The topological similarity between our dendrograms and the maximum likelihood trees was also assessed by using the method presented by Vienne et al. [16]. This methods tests the null hypothesis that trees are not more topologically similar than expected by chance. In the case of *Pseudomonadaceae* the resulting p-value is 7×10^{-6}; the p-values for both *Xanthomonas* and *Shewanella* is 1.3×10^{-2}. This confirms that our results for *Pseudomonadaceae* were much better than for *Xanthomonas* and *Shewanella*. In presenting these comparisons we are taking the maximum likelihood trees as ground truth, but it is of course possible that even they may contain errors.

It is important to mention that the creation of the dendrograms is an indirect way of computing the inversion distance between any two real genomes for which there is a processed dotplot. It is also possible to compute the distance directly using our notion of points and an algorithm for signed inversion distance [15]. However, because of the differences between various members of each bacterial group (differences in size and in number of matched regions) it is far from clear how these direct distances are to be related to one another in an uniform way so as to be usable for dendrogram creation.

Based on the histogram of distances and on the dendrograms obtained we conclude that SIB does a good (albeit not perfect) job of simulating the evolution of inversion rearrangements in the chromosomes of the three bacterial groups under study. The results for the *Pseudomonadaceae* family were better than for the other two. We believe that this was the case primarily because our empirical parameter values were derived from *Pseudomonadaceae* comparisons.

SIB was implemented using the Python programming language. The simulations presented here were run on a standard desktop computer (2.4 GHz Intel core 2 Duo machine with 2 GB memory). The approximate time spent to generate a complete binary tree simulation with depth 4 and a total number of $4,000$ symmetric inversions between the root and the leaves is about 15 seconds. Simulations of 2-branch scenarios with 4,000 inversions also take 15 seconds. Distance calculation was the most expensive computation, taking on average about two hours, but that of course is not part of the simulation *per se*.

5 Evaluation of One Reconstruction Program

In this section we present results of an evaluation of the MGR program [1] using SIB. MGR tries to reconstruct ancestral chromosomes assuming that the only possible rearrangement event is a general unconstrained inversion and using a parsimony criterion to infer inversion events.

Because MGR takes time prohibitively long with 2,500 points, we decided to evaluate it under a simple evolutionary case, described in Figure 10. In this case, there are just 150 inversions between the root and leaves A and B, and 300 inversions between the root and leaf C. The reconstruction problem for this case is also known as the *median problem* [10].

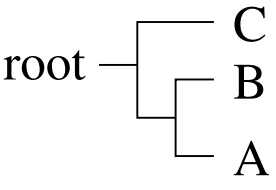


Fig. 10. Simple evolutionary scenario used to evaluate the program MGR

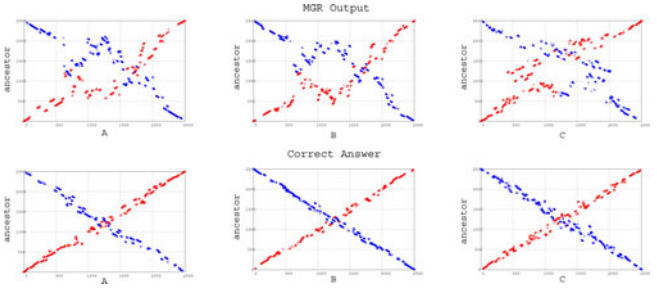


Fig. 11. Dotplots that result from comparing the reconstructed ancestor and the three descendants A, B and C (above) and corresponding correct dotplots (below)

We generated 10 different scenarios for the case described above. On average, the MGR reconstruction requires 380 inversions along all branches, as compared to the total of 600 inversions actually performed. This suggests that the parsimony criterion may not be a good criterion when reconstructing inversion history in bacteria.

In figure 11, we show the dotplots relative to the comparison of the reconstructed root with all leaves, for one specific scenario (the results are similar for other scenarios). Below each dotplot we show the actual dotplot that should have been obtained.

We can also compare the roots (in different scenarios) reconstructed by MGR to the actual roots in terms of breakpoint distance. If the reconstruction were perfect that distance would be zero. On average we obtained the value 119.

These results shows that MGR fails to reconstruct correctly the ancestors in our simulation. To the extent that our simulation is faithful to actual evolution in bacteria this suggests that MGR would not reconstruct correctly chromosomal ancestors either. This result is not surprising given the evolutionary model embedded in our simulation and the evolutionary model assumed by MGR, with its emphasis on the reconstruction of the most parsimonious scenario.

We have tried to test other reconstruction tools with SIB data, but MGR was the only one that was able to handle the input sizes described here.

6 Conclusion

In this work we presented the first simulation tool for the study of genomic inversions in bacterial genomes. The tool is in a preliminary stage of development. There are numerous improvements that can be made. Our method for analyzing X patterns in dotplots can be substantially improved by actually detecting the main matched backbone. We plan to model the backbone by a Longest Increasing Subsequence problem and combine the result with a regression analysis. This can be done for both forward and reverse matches. Such a method will be able to deal with situations when there are significant nonsymmetric inversions or when large deletions are present. This will allow us in turn to improve the evolutionary model, in particular explicitly modeling nonsymmetric inversions. Detailed study of the *Xanthomonas* and *Shewanella* results will probably result in revised models for inversion lengths and unbreakable segment number and lengths, and their associated parameter values. Care must be exercised in implementing all these changes because our experience so far has already shown that better modeling of one specific aspect sometimes yields worse overall results.

More detailed comparisons of our dendrograms and published phylogenetic trees may result in additional improvements, which could incorporate some notion of clade confidence. This would be akin to what is obtained from bootstrap analysis in phylogenetic reconstruction. The tool and its code are freely available at `http://www.ic.unicamp.br/~udias/sib`.

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. Darling, A.E., Miklós, I., Ragan, M.A.: Dynamics of genome rearrangement in bacterial populations. PLoS Genetics 4(7), e1000128 (2008)
3. Eisen, J.A., Heidelberg, J.F., White, O., Salzberg, S.L.: Evidence for symmetric chromosomal inversions around the replication origin in bacteria. Genome Biology 1(6), research 0011.1–0011.9 (2000)
4. Felsenstein, J.: PHYLIP (Phylogeny Inference Package) version 3.5c. Distributed by the author. Department of Genetics, University of Washington, Seattle

5. Guerrero, G., Peralta, H., Aguilar, A., et al.: Evolutionary, structural, and functional relationships revealed by comparative analysis of syntenic genes in rhizobiales. BMC Evolutionary Biology 5(55) (2005)
6. Kurtz, S., Phillippy, A., Delcher, A.L., et al.: Versatile and open software for comparing large genomes. Genome Biology 5(2), R12 (2004)
7. Larget, B., Simon, D.L., Kadane, J.B., Sweet, D.: A bayesian analysis of metazoan mitochondrial genome arrangements. Molecular Biology and Evolution 22(3), 486–495 (2005)
8. Moreira, L.M., Almeida, N.F., Potnis, N., et al.: Novel insights into the genomic basis of citrus canker based on the genome sequences of two strains of Xanthomonas fuscans subsp. aurantifolii. BMC Genomics 11(1), 238 (2010)
9. 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)
10. Ohlebusch, E., Abouelhoda, M.I., Hockel, K., Stallkamp, J.: The median problem for the reversal distance in circular bacterial genomes. In: Proceedings of Combinatorial Pattern Matching, pp. 116–127 (2005)
11. Saitou, N., Nei, M.: The neighbor-joining method: A new method for reconstructing phylogenetic trees. Molecular Biology and Evolution 4, 406–425 (1987)
12. Sankoff, D., Blanchette, M.: Multiple Genome Rearrangements. In: Istrail, S., Pevzner, P.A., Waterman, M. (eds.) Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB 1998), pp. 243–247 (1998)
13. Setubal, J.C., dos Santos, P., Goldman, B.S., et al.: Genome sequence of Azotobacter vinelandii, an obligate aerobe specialized to support diverse anaerobic metabolic processes. Journal of Bacteriology 191(14), 4534–4545 (2009)
14. Slater, S.C., Goldman, B., Goodner, B.: et al. Genome sequences of three agrobacterium biovars help elucidate the evolution of multi-chromosome genomes in bacteria. Journal of Bacteriology 191(8), 2501–2511 (2009)
15. Tesler, G.: Grimm: genome rearrangements web server. Bioinformatics 18(3), 492–493 (2002)
16. de Vienne, D.M., Giraud, T., Martin, O.C.: A congruence index for testing topological similarity between trees. Bioinformatics 23, 3119–3124 (2007)
17. Williams, K.P., Gillespie, J.J., Sobral, B., et al.: Phylogeny of gammaproteobacteria. Journal of Bacteriology 192(9), 2305–2314 (2010)
18. Wong, K., Golding, G.B.: A phylogenetic analysis of the pSymb replicon from the Sinorhizobium meliloti genome reveals a complex evolutionary history. Canadian Journal of Microbiology 49, 269–280 (2003)

Consistency of Sequence-Based Gene Clusters

Roland Wittler[1,2] and Jens Stoye[1]

[1] Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada
[2] Technische Fakultät, Universität Bielefeld, Germany
roland@cebitec.uni-bielefeld.de,
stoye@techfak.uni-bielefeld.de

Abstract. In comparative genomics, various combinatorial models can be used to specify gene clusters — groups of genes that are co-located in a set of genomes. Several approaches have been proposed to reconstruct putative ancestral gene clusters based on the gene order of contemporary species. One prevalent and natural reconstruction criterion is *consistency*: For a set of reconstructed gene clusters, there should exist a gene order that comprises all given clusters.

In this paper, we discuss the consistency problem for different gene cluster models on sequences with restricted gene multiplicities. Our results range from linear-time algorithms for the simple model of *adjacencies* to NP completeness for more complex models like *common intervals*.

1 Introduction

The exploration of the ancestral history of different species can give valuable information about their evolution. In whole-genome comparison, one commonly considers the order of the genes or other markers within the genome to study changes and similarities in the structure of different genomes.

Genes belonging to the same gene family are represented by the same identifier. To simplify matters, the term 'gene' will be used to refer to the corresponding gene family identifier. One simple way to model genomes is to use permutations. However, this approach includes the assumption that every gene occurs exactly once in each considered genome. To allow for duplications and deletions, a relaxation to sequences of genes is necessary. A convenient way to account for the orientation of a gene within the genome is to use signed permutations or signed sequences, respectively.

Evolutionary processes can rearrange a gene order. The gene composition of some regions, however, is preserved and can be found in several related genomes. These segments, denoted as *gene clusters*, often contain functionally or evolutionarily associated genes [14, 16]. Whenever the genomes of several species comprise the same gene cluster, it was presumably inherited from a common ancestor. Recent studies [1, 2, 7, 19] build on this idea to reconstruct ancient gene clusters and to infer ancient gene orders. More precisely, the internal nodes of a given phylogenetic tree are labeled with sets of gene clusters, based on the gene orders of contemporary species at the leaves of the tree. Beside the pure identification of gene clusters, such reconstructed scenarios for the origin of the clusters and the development of the gene order can give valuable information about underlying evolutionary processes, the ancestral history of the species, and functional and evolutionary relations of genes.

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 252–263, 2010.

Proposed reconstruction approaches differ in the underlying models for gene order and gene clusters, and in the applied methodology. However, a general aim is to ensure *consistency*: For a set of putative ancient gene clusters, there should exist at least one gene order that comprises all given clusters. Otherwise, the reconstruction result would be inconsistent with respect to the genome model.

The goal of reconstructing consistent labelings was first introduced by Bergeron *et al*. [2] who presented an algorithm that reconstructs sets of framed common intervals on permutations. Adam *et al*. [1] applied the parsimony principle as an objective function to reconstruct common intervals on permutations. A heuristic is used to reach consistency. Recently, Chauve and Tannier [7] proposed a methodology to reconstruct the gene order of the amniote genome, based on consistent labelings of common intervals and adjacencies. In our previous work [19], we introduced an algorithmical framework that is not restricted to a specific model but instead follows an oracle-based approach to compute most parsimonious consistent labelings for various models.

All of the above methods have been successfully applied to real data and proven to yield reasonable and valuable results. They all rely on permutation-based models, which enable efficient algorithms and data structures. In particular, the verification of consistency can be solved in polynomial time and space using data structures like PQ-trees or PC-trees [10]. Some reconstruction approaches could be easily adapted to the model of *sequences without duplications* which allows genes to be missing in some genomes but still requires each gene to occur at most once in each genome.

In this paper, we discuss consistency for *sequence-based* gene cluster models. Particularly, we consider the simple model of *adjacencies*, the classical model of *common intervals* [20], and two variants of the latter. For each of these models we address the problem: Given a set of gene clusters and a maximum copy number for each gene, decide whether there exists a valid gene order that contains all the clusters. Our results range from algorithms that verify consistency for adjacencies in linear time to the confirmation of NP completeness for the more complex models.

The paper has been organized in the following way. First, we formally introduce the *Consistency Problem* in Section 2. Then, in Section 3, we give an efficient solution for the gene cluster model of both signed and unsigned adjacencies. In Section 4, we present NP completeness results for the model of common intervals and its variants, before we finish with some discussions and conclusions in Section 5. The technical details of the NP completeness proofs can be found in [21, Appendix A].

2 The Consistency Problem

Assume a set of putative gene clusters, assigned to an ancestral node in a given tree. These ancient clusters in turn imply a set of putative ancient genomes: all those which contain all the given clusters. Depending on the gene cluster model used, this set of genomes can be empty if some of the clusters derived from different contemporary species are in contradiction with others. For example, when we model gene order as permutations, there is no valid gene order comprising the three adjacencies $\{a, b\}$, $\{a, c\}$ and $\{a, d\}$, because, according to the model, gene a can only occur once and thus only be neighbor of two other genes.

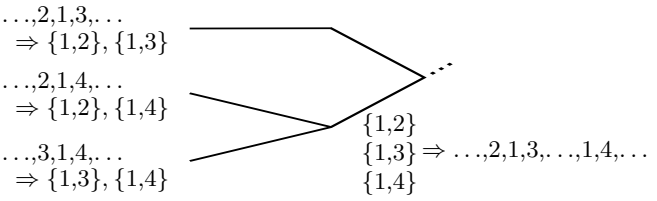


Fig. 1. Example for an artifact that arises when gene clusters are reconstructed on the basis of gene orders, where any gene can occur arbitrarily often. Any most parsimonious labeling would assign all three adjacencies to the lowermost internal node, implying at least two copies of gene 1.

In a more general case, we represent genomes as *sequences* of genes or other genomic markers. In a sequence, any element can occur multiple times or not at all, which in the context of gene order comparison corresponds to paralogous genes and gene deletions, respectively.

If we allow each gene to appear arbitrarily often in any genome, the question for consistency would become redundant: Any set of gene clusters is consistent since there is a valid gene order containing all assigned clusters. For instance, we can simply create a short sequence of genes according to each cluster separately and then concatenate these sequences to an absurd yet valid gene order. Such a construction is possible for any gene cluster model. As a consequence, consistency always holds and does not contribute to a specification of reasonable reconstruction results.

Even if we replace the naive concatenation approach and instead construct preferably compact valid gene orders, we cannot avoid to include some genes multiple times. In some cases, this causes side effects. In the example given in Figure 1, the classical parsimony principle is applied to assign gene clusters to the inner nodes of a given tree, minimizing the number of gains and losses of clusters. Although a gene is contained in all input genomes only once, it is reconstructed to occur multiple times for ancestral nodes. In this simple example, we consider a subsequence of only three genes in each input genome and obtain a segment of five genes for the examined internal node. In general, such artifacts imply unnaturally long genomes for higher levels in the tree.

To preclude this unwanted effect, we refine the concept of consistency. Instead of simply restricting the total length of a genome, we limit the multiplicity of each individual gene.

In the following problem definition, we intentionally refrain from specifying a concrete model of gene clusters and instead use the imprecise notion of a *sequence containing a cluster*. For instance, in the simple model of gene adjacencies, a sequence g *contains* a gene cluster $\{a, b\}$ if and only if the genes a and b occur adjacently in g.

Definition 1 (Consistency Problem). *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be the set of genes and $m : \mathcal{G}_N \to \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of gene clusters. The* consistency problem *is to decide if C is* consistent *with respect to m, i.e. whether there exists a sequence s over $\mathcal{G}_N$ for which the following properties hold:*

(i) s contains each gene g at most $m(g)$ times, and
(ii) s contains all gene clusters $c \in C$.

Whenever we want to consider consistency as a reconstruction criterion, we have to provide a solution for the above problem. As we will see in the following sections, the problem complexity highly depends on the specific cluster definition.

In our framework, we assume that the gene multiplicities are given. Nevertheless, we want to sketch some ways to specify $m(g)$ for the internal nodes in the phylogenetic tree.

For some specific datasets, we can rely on knowledge about the genomic history. For instance, several studies suggest two whole genome duplications in the evolution of the Chordate genome in the teleost fishes lineage [13]. Such information can be used to deduce the ancestral number of genes.

Otherwise, the most accurate but also elaborate approach would be to deploy *gene-tree species-tree reconciliation* [17] to reconstruct the history of the genes in terms of speciation events, gene duplications and gene losses. Less extensive and more suitable for our needs, one could also utilize approaches which do not require any further data or pre-knowledge. Probability-based methods [8] could be applied to effectively and reliably infer ancestral gene multiplicities $m_v(g)$ for all internal nodes v, given the copy number at the leaves. Or, we could apply the concept of parsimony and minimize the amount of copy number differences. A less restrictive solution is to define the multiplicity of a gene g for node u in a bottom-up fashion as the maximum over the multiplicities of its child nodes $v_1, \ldots, v_k$: $m_u(g) := \max_{i=1,\ldots,k} \left(m_{v_i}(g)\right)$.

Instead of performing a separate preprocessing step to fix the thresholds in advance, one could also try to include the gene multiplicity into the overall objective of the reconstruction. However, in general, optimizing for a combination including an original objective, consistency, and the gene copy number would be an intricate task due to the strong interdependencies of the subcriteria.

3 An Efficient Solution for Adjacencies

Probably the simplest formalization of co-localization of genes is the concept of *adjacencies*, i.e. two directly neighboring genes. This elementary pattern of gene order conservation, also known as *gene pairs* or *neighboring genes*, has been widely used in whole genome comparison. Especially in the field of gene order reconstruction, this model is one of the most prevalent concepts [4,7,15].

3.1 Unsigned Adjacencies

In the following, we formalize the concept of adjacencies and present a method to efficiently solve the consistency problem for adjacencies on sequences, i.e. to decide if there exists a sequence that contains a set of given adjacencies while considering each gene g at most $m(g)$ times. To model the problem, we use a graph theoretic approach.

Definition 2 (Unsigned Adjacencies on Sequences). *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be a set of genes. An adjacency $\{a, b\}$ of the genes $a, b \in \mathcal{G}_N$ is contained in a sequence s over $\mathcal{G}_N$ if and only if a and b occur adjacently at least once in s.*

Definition 3 (Gene Order Graph). *Let $\mathcal{G}_N = \{1, \ldots, N\}$ be a set of genes and C be a set of pairs $\{a, b\}$ with $a, b \in \mathcal{G}_N$. Then, the* gene order graph *of C, denoted by $G_N(C)$, is the graph with the vertex set $\{v_g \mid g \in \mathcal{G}_N\}$ and the edge set $\big\{\{v_a, v_b\} \mid \{a, b\} \in C\big\}$.*

The gene order graph of a set of adjacencies C can be constructed in $\mathcal{O}(N + |C|)$ time and space. In this process, we keep track of the degree of each node v_g, denoted by $\deg(v_g)$. Then, the following lemma allows us to check for consistency of C in $\mathcal{O}(N)$ steps and thus in a total running time and with a space requirement of $\mathcal{O}(N + |C|)$.

Lemma 1. *Let $\mathcal{G}_N = \{1, \ldots, N\}$ be a set of genes and let $m : \mathcal{G}_N \to \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of pairs $\{a, b\}$ with $a, b \in \mathcal{G}_N$ and $G_N(C) = (V, E)$ be the gene order graph of C. Then, C is consistent with respect to m if and only if the following conditions hold:*

(i) $\deg(v_g) \leq 2m(g)$ *for all vertices $v_g \in V$, and*
(ii) $\sum_{v_g \in c} \big(2m(g) - \deg(v_g)\big) > 0$ *for each connected component c in $G_N(C)$.*

Proof. Assume we have given $\mathcal{G}_N$, m, C and $G_N(C)$ as required by the lemma. We extend the gene order graph $G_N = (V, E)$ to a multigraph $H_N = (V', E')$, where the new vertex set contains one additional node v_0, i.e. $V' = V \cup \{v_0\}$. The multiset of edges E' contains all edges in E with multiplicity one and further auxiliary edges: For each vertex $v_g \neq v_0$ with $deg(v_g) < 2m(g)$ we add the edge $\{v_0, v_g\}$ with multiplicity $2m(g) - deg(v_g)$ to E'.

If condition (i) of the lemma holds, then all nodes in the obtained extended graph have even degree: All vertices $v_g \neq v_0$ are filled up to a degree of $2m(g)$ and v_0 is incident to $\sum_{v_g \in V} \big(2m(g) - deg(v_g)\big) = \sum_{v_g \in V} 2m(g) - 2|C|$ edges. Further, condition (ii) implies that for each connected component of G_N, in the extended graph, at least one edge connects this subgraph to v_0. Hence, H_N is connected.

Conditions (i) and (ii) imply that H_N is Eulerian. That means, there is a path starting and ending in v_0 which contains all edges, especially the edges of the original gene order

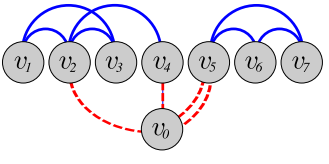


Fig. 2. An example to illustrate the proof of Lemma 1. Consider the set of genes $\mathcal{G}_7$ with the multiplicities $m(g) = 2$ for $g \in \{2, 5\}$ and otherwise $m(g) = 1$, and the set $C = \big\{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\big\}$ of unsigned adjacencies. The gene order graph $G_7(C)$ is depicted including the extensions described in the proof. The solid edges correspond to the original edges as defined by the given adjacencies, and the dashed lines represent the auxiliary edges. The obtained extended graph contains, for instance, the Eulerian cycle $(v_0, v_2, v_1, v_3, v_2, v_4, v_0, v_5, v_6, v_7, v_5, v_0)$, which corresponds to the valid gene order $(2, 1, 3, 2, 4, 5, 6, 7, 5)$.

graph, exactly once. Since each node $v_g \neq v_0$ has a degree of $2m(g)$, it is traversed exactly $m(g)$ times. Each such Eulerian path corresponds to a sequence of genes that contains all adjacencies in C and each gene g exactly $m(g)$ times, as exemplified in Figure 2. Thus, C is consistent with respect to m.

On the contrary, if condition (i) is not satisfied, there is at least one gene g that is contained in more given adjacencies than its multiplicity $m(g)$ allows. And, if condition (ii) does not hold for any connected component c, the maximal number of adjacencies of all genes in c is exhausted and the genes cannot be put into a linear order, i.e. a cycle containing v_0, with the remaining genes. In both cases, the existence of a valid gene order is precluded and thus, consistency is disproven. □

3.2 Signed Adjacencies

A slightly more sophisticated variant of the adjacency model is motivated by the observation that the orientation of genes can play a role in co-expression and also in gene order conservation [11].

Definition 4 (Signed Adjacencies on Signed Sequences). *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be a set of genes. A signed adjacency $\{a, b\}$ of the genes $a, b \in \{g, -g \mid g \in \mathcal{G}_N\}$ is contained in a sequence s over $\mathcal{G}_N$ if and only if a is directly followed by $-b$, or b by $-a$ at least once in s.*

Example 1. Consider the model of signed adjacencies for $N = 4$. The signed adjacency $\{2, -3\}$ is contained in both sequences $s_1 = (1, 2, 3, 4)$ and $s_2 = (4, 1, -3, -2)$. No other signed adjacencies of the genes 2 and 3 are contained in any of the two sequences.

We transfer the general idea from the unsigned to the signed case. To this end, we adjust the definition of the gene order graph. Now, each gene g is represented by two nodes in the graph, where each such pair is connected by $m(g)$ edges.

Definition 5 (Signed Gene Order Graph). *Let $\mathcal{G}_N = \{1, \ldots, N\}$ be a set of genes and let $m : \mathcal{G}_N \to \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of pairs $\{a, b\}$ with $a, b \in \{g, -g \mid g \in \mathcal{G}_N\}$. Then, the* signed gene order graph of C, *denoted by $G_N^s(C)$, is the multigraph with the vertex set $\{v_g, v_{-g} \mid g \in \mathcal{G}_N\}$ and the multiset of edges $\big\{\{v_g, v_{-g}\}$ with multiplicity $m(g) \mid g \in \mathcal{G}_N\big\} \cup \big\{\{v_a, v_b\}$ with multiplicity one $\mid \{a, b\} \in C\big\}$.*

Similarly to the unsigned case, we can construct the graph in $\mathcal{O}(N + |C|)$ time.

Lemma 2. *Let $\mathcal{G}_N = \{1, \ldots, N\}$ be a set of genes and let $m : \mathcal{G}_N \to \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of signed adjacencies $\{a, b\}$ with $a, b \in \{g, -g \mid g \in \mathcal{G}_N\}$ and $G_N^s(C) = (V, E)$ be the signed gene order graph of C. Then, C is consistent with respect to m if and only if the following conditions hold:*

(i) $deg(v_g) \leq 2m(|g|)$ *for all vertices* $v_g \in V$, *and*
(ii) $\sum_{v_g \in c} \big(2m(|g|) - deg(v_g)\big) > 0$ *for each connected component c in* $G_N^s(C)$.

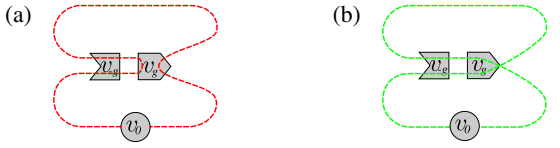


Fig. 3. Illustration of the relation of improper and proper Eulerian cycles in an extended signed gene order graph as described in the proof of Lemma 2

Proof. We proceed analogously to the unsigned case described in the proof of Lemma 1: We extend the signed gene order graph $G_N^s = (V, E)$ to a multigraph $H_N^s = (V', E')$, where the new vertex set contains one additional node v_0, i.e. $V' = V \cup \{v_0\}$. The multiset of edges E' contains all edges in E with multiplicity one and further auxiliary edges: For each vertex $v_g \neq v_0$ with $deg(v_g) < 2m(|g|)$ we add the edge $\{v_0, v_g\}$ with multiplicity $2m(|g|) - deg(v_g)$ to E'.

Then, again, the conditions (i) and (ii) of Lemma 2 imply the existence of an Eulerian path in $H_N^s(C)$. But in this case, the correspondence of such a path to a valid gene order is not trivial. When the pair of nodes representing gene g is traversed by a path $(\ldots, v_{-g}, v_g, \ldots)$, this relates to a signed gene order $(\ldots, g, \ldots)$, whereas a path $(\ldots, v_g, v_{-g}, \ldots)$ correlates to a signed gene order $(\ldots, -g, \ldots)$. By definition, $H_N^s(C)$ includes $m(|g|)$ edges $\{v_g, v_{-g}\}$. An Eulerian cycle passes each of these edges, but not necessarily in the above mentioned way. It might also be of the form $(\ldots, v_f, v_{-g}, v_g, v_{-g}, v_h \ldots)$ with $f \neq g \neq h$, which does not represent a signed gene order. In this case, $m(|g|) \geq 2$ and due to the construction of the extended graph, there are $m(|g|)$ edges $\{v_g, v_{-g}\}$ and at least $m(|g|)$ edges $\{v_g, v_h\}$ with $h \neq -g$. Hence, the considered Eulerian cycle has to pass node v_g again in the form $\ldots, v_i, v_g, v_j, \ldots$ with $i \neq -g \neq j$, as shown in Figure 3(a). However, whenever this situation arises, it is always possible to construct an alternative Eulerian cycle $(v_0, \ldots, v_{-g}, v_g, \ldots, v_{-g}, v_g, \ldots, v_0)$, as depicted in Figure 3(b). If this modification is performed for all such improperties, the obtained Eulerian cycle is proper in the sense that it represents a signed gene order $(\ldots, g, \ldots, g, \ldots)$. Thus, conditions (i) and (ii) imply not only the existence of an Eulerian path but also the existence of a valid signed gene order and hence consistency of C with respect to m. The reverse direction of the lemma holds analogously to Lemma 1. □

Based on the definition of a gene order graph, Lemmas 1 and 2 provide algorithms to solve the consistency problem on adjacencies on sequences in time and space linear in the number of genes and in the number of given adjacencies. Both the models and the lemmas can easily be modified to allow one circular gene order or even several circular chromosomes. Only the connectivity requirement has to be relaxed correspondingly.

4 NP Completeness for Common Intervals

To find larger conserved regions, we now address a model for gene clusters that, in contrast to adjacencies, generally span more than two genes: *Common intervals*, segments of the genome containing the same set of genes in an arbitrary order but not interrupted by other genes.

4.1 Basic Common Intervals

In line with other studies, we base our definition on the notion of *character sets*, which enables us to formalize the cluster model in a straightforward way. Since we will utilize this term for models on signed sequences later on, we directly define it for the general, signed case. Although, in our framework, a common interval is defined on a *single* gene order, we stick to the term *common* to not confuse the reader familiar with this gene cluster model by redefining the same concept under a different name.

Definition 6 (Character Set). *Let $s = (a_1, \ldots, a_{|s|})$ be a signed sequence. Then, the* character set *of s, denoted $\mathcal{CS}(s)$, is the set of all elements in s: $\mathcal{CS}(s) := \left\{|a| \mid a \in \{a_1, \ldots, a_{|s|}\}\right\}$.*

Definition 7 (Common Intervals on Sequences). *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be a set of genes. Then, a common interval $c \subseteq \mathcal{G}_N$ with $|c| > 1$ is contained in a sequence s over $\mathcal{G}_N$ if and only if s contains a substring s' such that $\mathcal{CS}(s') = c$.*

Recall that we want to find an answer to the question: Given a set of common intervals C and a multiplicity threshold function m, is there a valid gene order that contains all elements of C and meets the restrictions imposed by m? As we will show now, this problem is NP complete.

Theorem 1. *The consistency problem for common intervals on sequences is NP complete.*

Proof (Sketch). One can easily formulate an algorithm that verifies a given solution, i.e. a proper gene order, for correctness in polynomial time, which shows that the problem belongs to the complexity class NP.

NP hardness is proven by reduction from the following variant of the Hamiltonian cycle problem: Let $G = (V \cup W, E)$ be a connected, undirected, bipartite graph with $|V| = |W| \geq 3$, $E \subseteq \left\{\{v, w\} \mid v \in V, w \in W\right\}$ and $\deg(u) = |\{e \in E | u \in e\}| \leq 3$ for all $u \in V \cup W$. Decide whether there exists a Hamiltonian cycle in G, i.e. a path in G that starts and ends in the same vertex $v' \in V$ and in-between contains each vertex $v \in V \backslash \{v'\}$ exactly once. This problem is known to be NP complete [12]. By reducing it to the consistency problem for common intervals in polynomial time, we get NP hardness of the latter problem.

For a given graph as stated in the problem definition, we construct an instance of the consistency problem as follows: For each node, depending on its degree, and for each edge, depending on the graph structure, we add certain elements to the set of genes and define certain common intervals. We restrict the multiplicity of the genes such that common intervals have to overlap in a valid gene order in specific substrings. Each of these substrings corresponds to a node or an edge in the graph, and overlapping substrings correspond to paths in the graph. Finally, we show that the substrings can be combined to a complete, valid sequence if and only if there is a Hamiltonian cycle in the graph. Details are given in [21, Appendix A.2]. □

4.2 Variants of Common Intervals

Beside its classical definition, there are different generalizations of common intervals on sequences discussed in the literature, such as r-window clusters and max-gap clusters [9], or approximate gene clusters [6,18]. Since the consistency problem is NP complete for basic common intervals, any generalization is NP hard as well.

In contrast to generalizations, there are also other cluster models which are restricted variants of common intervals. In the following, we will discuss such models, in particular framed and nested common intervals.

Framed Common Intervals

This gene cluster model, common intervals framed by two genes whose orientations have to be conserved, has first been introduced on permutations as *conserved intervals* [3]. In gene order reconstruction, framed common intervals on permutations have been the first model to formally state the problem of finding putative ancestral sets of gene clusters preserving consistency [2].

Definition 8 (Framed Common Intervals on Signed Sequences). *Let* $\mathcal{G}_N := \{1, \ldots, N\}$ *be a set of genes. A framed common interval* $[a\ I\ b]$ *consists of two* extremities *a and b with* $|a|, |b| \in \mathcal{G}_N$*, and a set of* inner elements $I \subseteq \mathcal{G}_N$*. We say that* $[a\ I\ b]$ *is contained in a signed sequence s, if and only if in s, a is followed by b or $-b$ by $-a$, and the character set of the substring between the extremities is equal to I.*

According to this definition, a gene can be extremity and inner element, or even left and right extremity at the same time. Apart from that, analogously to basic common intervals, a cluster can occur multiple times in one genome, and one gene can be contained several times in one cluster occurrence, as illustrated by the following example.

Example 2. Consider the model of framed common intervals for $N = 6$ and sequence $s = (5, 4, -2, -1, 2, -3, 6)$. Beside others, the framed common interval $[4\,\{1, 2\}\,{-3}]$, is contained in s as illustrated by the *box diagram* below, where the occurrences of the extremeties and the inner elements are surrounded by rectangles:

$$s = (\ +5, \boxed{+4,} \boxed{-2,\ -1,\ +2,} \boxed{-3,} +6\ ) .$$

The obvious relationship of basic and framed common intervals allows to infer an important correlation of these models with respect to the consistency problem: Any instance of this problem for common intervals can be reduced to an instance for framed common intervals. Based on this, we can deduce the following statement.

Theorem 2. *The consistency problem for framed common intervals on signed sequences is NP complete.*

Proof (Sketch). To show NP hardness, we reduce the consistency problem for common intervals to the consistency problem for framed common intervals in polynomial time. To this end, we introduce two additional genes with a certain multiplicity and, for each

basic common interval, we create a framed common interval including these genes as framing and inner elements. Then, any valid gene order for one problem instance can be transformed into a valid gene order for the other instance by removing occurrences of the additional genes or inserting them, respectively. Details are given in [21, Appendix A.3]. □

Nested Common Intervals

Hoberman and Durand [9] discussed nestedness as a desired property of gene clusters and proposed a first algorithm to identify respective clusters. Recently, *nested common intervals* were formally defined and studied in [5].

Definition 9 (Nested Common Intervals on Sequences). *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be a set of genes. The structure of a nested common interval is defined recursively. A nested common interval is either*

(i) an unordered pair of genes $\{a, b\}$ with $a \neq b$, which is contained in a sequence s over $\mathcal{G}_N$ if and only if a and b are adjacent in s, or

(ii) a tupel (c, a) of a nested common interval c and a gene a, which is contained in a sequence s if and only if, in s, a is adjacent to a substring s' of s such that $\mathcal{CS}(c) = \mathcal{CS}(s')$ and c is contained in s',

where the character set of a nested common interval is the set of all contained genes: $\mathcal{CS}(\{a, b\}) := \{a, b\}$ and $\mathcal{CS}((c, a)) := \mathcal{CS}(c) \cup \{a\}$.

Similar to the other cluster models discussed above, any nested common interval may occur multiple times in one genome and one gene may be contained multiple times in the occurrence of a cluster in one genome. Analogously to framed common intervals, one gene may be incorporated in the definition of one cluster several times.

Example 3. Consider the model of nested common intervals for $N = 6$ and sequence $s = (5, 4, 2, 1, 2, 3, 6)$. Then, beside others, the nested common interval $((\{2, 3\}, 1), 4)$ is contained in s as illustrated below, where the occurrences of the subclusters are indicated by lines:

$$(5, \underline{4}, \underline{2, 1, 2, 3}, 6)\,.$$

Even the strict assumption of nestedness is not strong enough to allow an efficient verification of consistency.

Theorem 3. *The consistency problem for nested common intervals on sequences is NP complete.*

Proof (Sketch). NP hardness is proven similarly to Theorem 1. The common intervals used in that proof can be replaced by certain nested common intervals such that the argumentation holds similarly. Details are given in [21, Appendix A.4]. □

Further Variations and Restrictions

Our NP completeness results also hold for further variations of the above models: defined on circular sequences, restricted in size, basic and nested common intervals containing each gene at most once etc. These results are detailed in [21] and will be discussed elaborately in the full version of this paper.

5 Conclusion

In this paper, we have discussed the consistency problem, i.e. the decision whether there exists a valid gene order comprising a given set of gene clusters. We have discussed this question for different gene cluster models on sequences with restricted gene multiplicities. In summary, we identified a severe border between gene cluster models for which we can verify consistency efficiently and those for which we cannot. The complexity rises drastically from linear for adjacencies to NP hard for more general cluster models, even if they are strongly restricted.

This raises the question for a sequence-based gene cluster model that, on the one hand, allows some degree of flexibility and, on the other hand, offers a polynomial-time algorithm to verify consistency. The integration of such a model into any of the existing reconstruction methods could increase sensitivity. Actually, first results on both simulated and real data indicate that within segments of conserved gene content, the order of the genes is conserved almost exactly [21]. Thus, a model covering only single missing or additional genes, or the reversal of two neighboring genes could already enhance reconstruction results strongly.

Another way to find a practical solution for flexible models is not to be deterred by the NP hardness results. In fact, the reduction procedures on which the proofs are based produce instances of the consistency problem where the multiplicity for some genes grows with the instance size. This suggests to reconsider the problems in the context of *fixed parameter tractability*. However, our first investigations in this direction were not promising. Moreover, the maximum copy number of genes observed in real data can be large in general.

We implemented the gene order graph to model adjacencies on sequences and integrated this gene cluster model into our unified reconstruction framework, presented in [19], available from the web site `bibiserv.techfak.uni-bielefeld.de/rococo/`. An elaborate description of the method and the results can be found in [21]. We refrain from reporting detailed results here because these are concerned more with the reconstruction method than with the general concept of consistency discussed in this paper. Nevertheless, we would like to mention the following overall findings. Simulations showed that estimating the gene multiplicities using the simple maximum approach does not significantly decrease the accuracy of the reconstruction compared to using the "real" simulated copy numbers. Furthermore, we applied our method on genomic data of Corynebacteria using different gene cluster models: Common intervals on permutations and adjacencies on sequences. A comparison of the results revealed a large overlap. Nevertheless, many conserved segments could only be identified by either of the approaches. This highlights the importance of studying gene cluster reconstruction with respect to different, especially flexible models for gene clusters and the relaxed model of sequences for gene order.

Acknowledgments

The authors wish to thank Cedric Chauve for discussions on an earlier version of this manuscript. The work of Roland Wittler was supported by the DFG Graduiertenkolleg Bioinformatik (GK 635).

References

1. Adam, Z., Turmel, M., Lemieux, C., Sankoff, D.: Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution. J. Comp. Biol. 14(4), 436–445 (2007)
2. Bergeron, A., Blanchette, M., Chateau, A., Chauve, C.: Reconstructing ancestral gene order using conserved intervals. In: Jonassen, I., Kim, J. (eds.) WABI 2004. LNCS (LNBI), vol. 3240, pp. 14–25. Springer, Heidelberg (2004)
3. Bergeron, A., Stoye, J.: On the similarity of sets of permutations and its applications to genome comparison. J. Comp. Biol. 13(7), 1340–1354 (2006)
4. Bhutkar, A., Gelbart, W.M., Smith, T.F.: Inferring genome-scale rearrangement phylogeny and ancestral gene order: A drosophila case study. Genome Biol. 8(11), R236 (2007)
5. Blin, G., Stoye, J.: Finding nested common intervals efficiently. J. Comp. Biol. (to appear 2010); A preliminary version appeared in Proc. of RECOMB-CG 2009. LNCS (LNBI), vol. 5817, pp. 59–69 (2009)
6. Böcker, S., Jahn, K., Mixtacki, J., Stoye, J.: Computation of median gene clusters. J. Comp. Biol. 16(8), 1085–1099 (2009)
7. Chauve, C., Tannier, E.: A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. PLoS Comput. Biol. 4(11), e1000234 (2008)
8. Csűrös, M., Miklós, I.: Mathematical framework for phylogenetic birth-and-death models. Arxiv preprint arXiv:0902.0970 (2009)
9. Hoberman, R., Durand, D.: The Incompatible Desiderata of Gene Cluster Properties. In: McLysaght, A., Huson, D.H. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3678, pp. 73–87. Springer, Heidelberg (2005)
10. Hsu, W.-L., McConnell, R.: PQ-trees, PC-trees, and planar graphs. In: Mehta, D.P., Sahni, S. (eds.) Handbook of Data Structures and Applications (2004)
11. Huynen, M.A., Snel, B., Bork, P.: Inversions and the dynamics of eukaryotic gene order. Trends Genet. 17(6), 304–306 (2001)
12. Itai, A., Papadimitriou, C.H., Szwarcfiter, J.L.: Hamilton paths in grid graphs. SIAM J. Computing 11(4), 676–686 (1982)
13. Jaillon, O., Aury, J.-M., Brunet, F., Petit, J.-L., Stange-Thomann, N., et al.: Genome duplication in the teleost fish tetraodon nigroviridis reveals the early vertebrate proto-karyotype. Nature 431(7011), 946–957 (2004)
14. Lawrence, J.G., Roth, J.R.: Selfish operons: Horizontal transfer drive the evolution of gene clusters. Genetics 143(4), 1843–1860 (1996)
15. Ma, J., Zhang, L., Suh, B.B., Raney, B.J., Burhans, R.C., Kent, J.W., Blanchette, M., Haussler, D., Miller, W.Z.: Reconstructing contiguous regions of an ancestral genome. Genome Res. 16(12), 1557–1565 (2006)
16. Overbeek, R., Fonstein, M., D'Souza, M., Pusch, G.D., Maltsev, N.: The use of gene clusters to infer functional coupling. Proc. Natl. Acad. Sci. USA 96(6), 2896–2901 (1999)
17. Page, R.D.M., Charleston, M.A.: From gene to organismal phylogeny: Reconciled trees and the gene tree/species tree problem. Mol. Phyl. and Evol. 7(2), 231–240 (1997)
18. Rahmann, S., Klau, G.W.: Integer linear programs for discovering approximate gene clusters. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 298–306. Springer, Heidelberg (2006)
19. Stoye, J., Wittler, R.: A unified approach for reconstructing ancient gene clusters. IEEE/ACM Trans. Comput. Biol. Bioinf. 6(3), 387–400 (2009)
20. Uno, T., Yagiura, M.: Fast algorithms to enumerate all common intervals of two permutations. Algorithmica 26(2), 290–309 (2000)
21. Wittler, R.: Phylogeny-based Analysis of Gene Clusters. Ph.D. Thesis, Faculty of Technology, Bielefeld University (2010), `http://bieson.ub.uni-bielefeld.de/volltexte/2010/1627/`

Efficient Computation of Approximate Gene Clusters Based on Reference Occurrences

Katharina Jahn

Institut für Informatik, Technische Fakultät, Universität Bielefeld, Germany
kjahn@cebitec.uni-bielefeld.de

Abstract. Whole genome comparison based on the analysis of gene cluster conservation has become a popular approach in comparative genomics. While gene order and gene content as a whole randomize over time, it is observed that certain groups of genes which are often functionally related remain co-located across species. However, the conservation is usually not perfect which turns the identification of these structures, often referred to as approximate gene clusters, into a challenging task. In this paper, we present a polynomial time algorithm that computes approximate gene clusters based on reference occurrences. We show that our approach yields highly comparable results to a more general approach and allows for approximate gene cluster detection in parameter ranges currently not feasible for non-reference based approaches.

1 Introduction

Whole genome comparison on the level of gene order has become an important field of comparative genomics. It is well known that genomes evolve not only on the level of nucleotide sequence but also by means of large-scale rearrangements operations, such as inversions and transpositions, as well as changes in the gene content. Focusing on this large-scale structure, genomes are usually modeled as strings of integers so that genes belonging to the same gene family are represented by the same integer. If no selective pressure was acting on whole genome evolution, gene order and gene content would randomize over time. In practice, we observe a low overall gene order conservation between species that is contrasted by a number of small, well-conserved segments, often referred to as gene clusters. Such local aberrations from genome randomization are known to provide highly informative signals for functional analysis. The identification of these structures can be a challenging task as conservation patterns may be highly variable across species. Due to micro-rearrangements, gene order can vary across cluster occurrences, and due to gene insertions and gene losses, cluster occurrences may be interrupted by genes that do not belong to the cluster, and contain only a subset of the clustered genes. To cope with such variations, different approximate gene cluster models have been proposed in the last years.

One of the first formal gene cluster models are *common intervals* which allow for variable gene order and multiple gene copies within cluster occurrences, but

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 264–277, 2010.

not for differences in the set of contained genes. Due to this restriction, the computation of gene clusters under the common intervals model runs in polynomial time with respect to the maximum genome length n.

The most-widely used gene cluster model that covers gene insertions are *max-gap clusters* [2,4]. This model allows for an arbitrary number of gaps in every cluster occurrence, each up to a fixed length, that can be filled with intermittent genes. The asymptotic complexity of identifying max-gap clusters increases exponentially with the number of compared genomes, but practical running times were shown to be feasible [6]. The treatment of gene losses in the max-gap model is as follows: Any gene that is lost in a cluster occurrence is counted as an intermittent gene in those occurrences where it is still present. Therefore, the set of genes representing a gene cluster reduces to the minimal consensus of genes that occur in all cluster occurrences. Moreover, gap sizes may have to be increased artificially to bridge seemingly intermittent genes.

Recently, a number of set distance based models arose, e. g. *median gene clusters* [3], that extend the concept of common intervals towards approximate conservation of gene content. The basic idea is to define a maximum distance δ between a consensus gene set and its approximate occurrences that can be freely distributed over gene losses and insertions located anywhere in the approximate cluster occurrences. A reported gene cluster is not a minimal consensus but a set of genes that is optimized in the sense that the total distance to its approximate occurrences is minimized. In principle, this approach allows for the detection of gene clusters with diverse conservation patterns in a large number of genomes. However, the search space grows exponentially, either with distance threshold δ [9] or the number of compared genomes k [3]. Practical computation times are feasible for many, but not all, interesting parameter ranges. The exponential time complexity is caused by the optimality criterion imposed on the consensus gene set. If a gene cluster is not represented by the optimal consensus set but a close set that has a reference occurrence in the given genomes, i. e. one without intermittent or missing genes, the search space becomes polynomially bounded. The computation of such *reference* based conservation patterns was studied by Amir et al [1]. However, it is possible to construct a counter example for which their graph-based, $O(kn^3 + \textit{output size})$ time and $O(kn^3)$ space, algorithm does not detect the complete solution set [5].

In this paper, we show that the complete set of reference-based approximate cluster occurrences can be identified in time $O(k^2n^2(\delta+1)^2+\textit{output size})$, $\delta \ll n$, using $O(kn^2)$ space. To assess the relevance of our reference-based gene cluster model and the performance of our algorithm, we compare it to a related approach that solves the general approximate gene cluster problem.

2 Basic Definitions and Notation

We model a genome as a *string* over a finite alphabet $\Sigma = \{1, \ldots, \sigma\}$ of gene family ids, such that genes belonging to the same homology family are represented by the same integer. Given a string S, we denote its *length* by $|S|$ and

refer to its ith character by $S[i]$, $1 \leq i \leq |S|$. We say that character $c \in \Sigma$ *occurs* at position i in S if and only if $S[i] = c$. To capture the character content of S regardless of sequential arrangement and multiple character occurrences, we define the *character set* of S as $\mathcal{CS}(S) = \{S[i] \mid 1 \leq i \leq |S|\}$. To simplify notation, we assume that a string S is extended on both ends by a terminal character $0 \notin \Sigma$, i.e. $S[0] = S[|S| + 1] = 0$. We define $S[i, j]$ to be the *substring* of S that starts with its ith and ends with its jth character, $1 \leq i \leq j \leq |S|$. The corresponding index interval $[i, j]$ is called a *location* of $C \subseteq \Sigma$ if and only if $C = \mathcal{CS}(S[i, j])$. We distinguish different types of intervals in a string S. We call an interval $[i, j]$ with $j \geq i$ *left-maximal* if $S[i - 1] \notin \mathcal{CS}(S[i, j])$, *right-maximal* if $S[j + 1] \notin \mathcal{CS}(S[i, j])$ and *maximal* if it is both left- and right-maximal.

Definition 1. *Given a set of strings $\mathcal{S} = \{S_1, \ldots, S_k\}$, $k \geq 2$, we call a k-tuple of maximal intervals* $([i_1, j_1], \ldots, [i_k, j_k])$ common intervals *of $\mathcal{S}$ if and only if there is a $C \subseteq \Sigma$ with:*

$$C = \mathcal{CS}(S_1[i_1, j_1]) = \ldots = \mathcal{CS}(S_k[i_k, j_k]).$$

Such character sets correspond to perfectly conserved gene clusters. To quantify differences in the gene content of approximate gene cluster occurrences, we use the *symmetric set distance* which defines the distance between two sets C and C' as the cardinality of their *symmetric difference*: $D(C, C') = |C \setminus C'| + |C' \setminus C|$. This measure constitutes a metric and therefore meets all intuitive notions of a distance measure such as validity of the triangle inequality. We extend the concept of character set locations towards approximate conservation: Given an integer $\delta \geq 0$, we say an interval $[i, j]$ in a string S is a *δ-location* of a character set C if and only if $D(C, \mathcal{CS}(S[i, j])) \leq \delta$ and $C \cap \mathcal{CS}(S[i, j]) \neq \emptyset$. We distinguish additional subtypes of maximal intervals: Given a character set C, we say a maximal interval $[i, j]$ in S with $\mathcal{CS}(S[i, j]) \cap C \neq \emptyset$ is *closed* with respect to C, or *C-closed* for short, if and only if $S[i - 1] \notin C$ and $S[j + 1] \notin C$. For the next subtype of maximal intervals, we define the *left-most essential position* i^* of $[i, j]$ with respect to C as the smallest index i', $i \leq i' \leq j$, such that $S[i'] \in C$. Analogously, we define the *right-most essential position* j^* of $[i, j]$ with respect to C as the largest index j', $i \leq j' \leq j$, such that $S[j'] \in C$. Interval $[i^*, j^*]$ is called the *C-essential subinterval* of $[i, j]$. The characters at positions i^* and j^* are called *left-most essential character*, respectively *right-most essential character*, with respect to C. These concepts are used to define the second subtype of maximal intervals which comprises all maximal intervals $[i, j]$ with $\mathcal{CS}(S[i, j]) \cap C \neq \emptyset$ for which $\mathcal{CS}(S[i, j]) = \mathcal{CS}(S[i^*, j^*])$ holds. Intervals of the form of $[i, j]$ are called *compact* with respect to C or *C-compact* for short. If a maximal $[i, j]$ is both closed and compact with respect to C, we call it *optimal* with respect to C, or *C-optimal* for short. A C-optimal δ-location of C is called an optimal δ-location. The idea behind this definition is to define an optimal placement of an approximate cluster occurrence. In particular, one can show the following:

Observation 1. *The number of optimal δ-locations of a character set $C \subseteq \Sigma$ in a string S of length n is in $O(n(\delta + 1))$ for all $\delta \geq 0$.*

Proof. We count how many optimal δ-locations of C can have the same left-most position a in S. For that purpose, we show that every two such intervals $[a, b_1]$ and $[a, b_2]$, $b_1 < b_2$, contain a different number of characters from $\Sigma \setminus C$. We have $S[b_1 + 1] \notin C$ and $S[b_1 + 1] \notin \mathcal{CS}(S[a, b_1])$, while $S[b_1 + 1] \in \mathcal{CS}(S[a, b_2])$. However, it also holds that every $c \in \mathcal{CS}(S[a, b_1]) \setminus C$ is contained in $\mathcal{CS}(S[a, b_2])$. Therefore, the number of characters from $\Sigma \setminus C$ cannot be the same in the two intervals. Moreover, a δ-location of C can have only between 0 and δ characters from $\Sigma \setminus C$. Thus, there are at most $\delta+1$ optimal δ-locations of C with left-most position a and $O(n(\delta + 1))$ optimal δ-locations of C in the complete string. □

We now have all the prerequisites to define the problems studied in this paper. At first, we study the detection of optimal δ-locations:

Problem 1. Given two strings S_1 and S_2 over an alphabet Σ and a distance threshold δ, find for each maximal $[i, j]$ in S_1 all optimal δ-locations of $\mathcal{CS}(S_1[i, j])$ in S_2.

For $\delta = 0$, this is problem is equivalent to common intervals detection. For approximate gene cluster detection in multiple genomes the definition can be extended as follows:

Problem 2. Given a set of strings $\mathcal{S} = \{S_1 \ldots, S_k\}$ over an alphabet Σ, a distance threshold δ and a quorum parameter $q \leq k$, find each $C \subseteq \Sigma$ with $C = \mathcal{CS}(S_\ell[i_\ell, j_\ell])$ for some $1 \leq \ell \leq k$, $1 \leq i_\ell \leq j_\ell \leq |S_\ell|$ that has δ-locations in at least q different strings and report all its optimal δ-locations in $S_1 \ldots, S_k$.

We call character sets of the form of C *conserved reference sets* and their optimal δ-locations *reference-based approximate common intervals.*

3 Computation of Optimal δ-Locations

The algorithm presented in the following adopts the basic search strategy of the Connecting Intervals (CI) Algorithm for the computation of common intervals [10]. Therefore, we begin with a short review of this algorithm.

3.1 The Connecting Intervals Algorithm

The CI Algorithm uses two static data structures that are computed in a preprocessing step: an array of length $|\Sigma|$ called POS that lists for each character $c \in \Sigma$ its occurrences in S_2 from left to right, and a $|S_2| \times |S_2|$ table named NUM that stores for every interval in S_2 how many different characters are contained, i. e. $\text{NUM}[i, j] = |\mathcal{CS}(S_2[i, j])|$. For an example, see Figure 1.

The basic idea of the CI Algorithm is that while going systematically through all maximal intervals $[i, j]$ in S_1, referred to as *reference intervals* in the following, one iteratively generates and extends marked intervals in S_2 that consist only of characters occurring in the current *reference character set* $\mathcal{CS}(S_1[i, j])$ using array POS for their identification. An example of this procedure is given in

Pos[1] = ∅
Pos[2] = {4, 7}
Pos[3] = {3, 6, 10}
Pos[4] = {5, 12}
Pos[5] = {1, 9}
Pos[6] = {8, 11}
Pos[7] = {2}

Num:

$i\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	5	5	6	6	6	6	6
2		1	2	3	4	4	4	5	6	6	6	6
3			1	2	3	3	3	4	5	5	5	5
4				1	2	3	3	4	5	5	5	5
5					1	2	3	4	5	5	5	5
6						1	2	3	4	4	4	5
7							1	2	3	4	4	5
8								1	2	3	4	5
9									1	2	3	4
10										1	2	3
11											1	2
12												1

Fig. 1. Example of data structures Pos and Num for alphabet $\Sigma = \{1, 2, 3, 4, 5, 6, 7\}$ and two strings $S_1 = 1\ 2\ 4\ 6\ 4\ 3\ 1\ 5\ 6\ 2$ and $S_2 = 5\ 7\ 3\ 2\ 4\ 3\ 2\ 6\ 5\ 3\ 6\ 4$

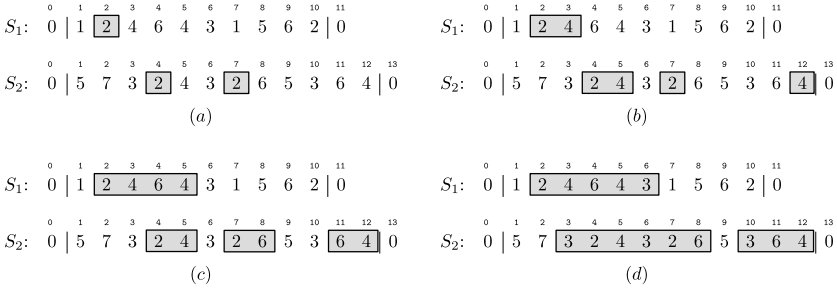


Fig. 2. Iterative generation of reference intervals in S_1 for start position 2 and corresponding interval marking in S_2 for $S_1 = 1\ 2\ 4\ 6\ 4\ 3\ 1\ 5\ 6\ 2$ and $S_2 = 5\ 7\ 3\ 2\ 4\ 3\ 2\ 6\ 5\ 3\ 6\ 4$

Figure 2. Common intervals are retrieved by comparing the character content of $\mathcal{CS}(S_1[i, j])$ and the marked intervals in S_2. Since by construction the character sets of the marked intervals are subsets of $\mathcal{CS}(S_1[i, j])$, this can be tested by comparing their sizes, using the table Num, and the current size of $\mathcal{CS}(S_1[i, j])$. Only those intervals in S_2 that were extended by an occurrence of the most recent element of the reference character set need to be considered for this test. Other intervals do not contain this character and thus have a smaller character set. It can be shown that this algorithm detects all common intervals in time $O(n^2)$ using $O(n^2)$ space.

3.2 Extension of the Connecting Intervals Algorithm

The changes necessary to the CI Algorithm to compute optimal δ-locations are presented together with the pseudocode given in Algorithm 1. The presented algorithm is a further development of the filter step in the median gene cluster

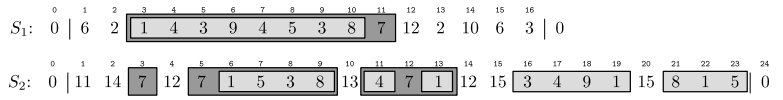
Algorithm 1. Computation of optimal δ-locations

```
 1: for i = 1, ..., |S_1| do
 2:   j ← i , C ← ∅
 3:   while j ≤ |S_1| and [i, j] is left-maximal do
 4:     c ← S_1[j]
 5:     while [i, j] is not right-maximal do
 6:       j ← j + 1
 7:     end while
 8:     for each interval [a, b] ∈ C do
 9:       remove [a, b] from C unless it is optimal δ-location of CS(S_1[i, j])
10:     end for
11:     for each position p in S_2 with S_2[p] = c do
12:       mark position p in S_2
13:       find unmarked positions l_1, ..., l_{δ+1} and r_1, ..., r_{δ+1} around p
14:       for each interval [l_x+1, r_y−1] with 1 ≤ x, y ≤ δ+1 do
15:         if [l_x+1, r_y−1] is optimal δ-location and not contained in C then
16:           add [l_x+1, r_y−1] to C
17:         end if
18:       end for
19:     end for
20:     if C ≠ ∅ then
21:       output ([i, j], C)
22:     end if
23:     j ← j+1
24:   end while
25:   unmark all positions in S_2
26: end for
```

computation scheme where for a given δ only the existence of a δ-location is tested. To solve Problem 1 we need to enumerate all optimal δ-locations.

We adopt the iterative generation of reference intervals from the original CI Algorithm, where for a fixed i the maximal intervals starting at i are processed one after the other for increasing values of j (lines 1 to 7). Also the marking of intervals in S_2 that consist only of characters from the current reference character set $C = \mathcal{CS}(S_1[i, j])$ (line 12) is useful for this purpose: Since approximate locations need to have character sets that intersect with C, these intervals are starting points for detecting C-optimal δ-locations. However, unlike with perfect locations, it is not sufficient to consider only recently extended maximal marked intervals. For $\delta > 0$, intervals that are partially unmarked and/or contain no occurrence of c, the character most recently added to C, can as well be C-optimal δ-locations.

However, we observe that it is not necessary to compute optimal δ-locations from scratch for every reference character set. For a fixed left border i successive reference intervals $[i, j']$ and $[i, j]$ with $\mathcal{CS}(S_1[i, j]) = \mathcal{CS}(S_1[i, j']) \cup \{c\}$ can share some optimal δ-locations as the example in Figure 3 shows. Therefore, we store the optimal δ-locations of one reference character set in a list $\mathcal{C}$ and test for



reference character sets: $\mathcal{CS}(S_1[3,10]) = \{1,3,4,5,8,9\}$, $\mathcal{CS}(S_1[3,11]) = \{1,3,4,5,7,8,9\}$

optimal δ-locations in S_2:

	with occurrence(s) of 7	without occurrence of 7
$\mathcal{CS}(S_1[3,10])$:	$[\mathbf{5},\mathbf{13}]$, $[\mathbf{11},\mathbf{23}]$	$[6,9]$, $[6,11]$, $[13,23]$, $[\mathbf{15},\mathbf{23}]$, $[\mathbf{16},\mathbf{19}]$, $[21,23]$
$\mathcal{CS}(S_1[3,11])$:	$[3,9]$, $[3,14]$, $[3,23]$, $[5,9]$, $[\mathbf{5},\mathbf{13}]$, $[\mathbf{11},\mathbf{23}]$	$[\mathbf{15},\mathbf{23}]$, $[\mathbf{16},\mathbf{19}]$

Fig. 3. Optimal δ-locations of two successive reference character sets in two example strings S_1 and S_2 for distance threshold $\delta = 3$. Shared optimal δ-locations are printed bold

the following reference character set which of these intervals can be inherited. Only in the case where the left border of the reference interval is shifted, $\mathcal{C}$ is emptied as the generation of reference character sets starts anew (line 2). The optimal δ-locations that can be inherited to the next reference character set can be characterized as follows:

Observation 2. *Let C, $C' \subseteq \Sigma$ with $C = C' \cup \{c\}$ and $c \notin C'$. Every interval $[a,b]$ in a string S over Σ that is an optimal δ-location of C' is also an optimal δ-location of C if and only if either:*

(i) $c \in \mathcal{CS}(S[a,b])$, *or*

(ii) $c \notin \mathcal{CS}(S[a,b])$, $c \neq S[a-1]$, $c \neq S[b+1]$ *and* $D(C', \mathcal{CS}(S[a,b])) < \delta$.

Proof. (i) Since $[a,b]$ is C'-optimal and contains an occurrence of c, it is C-optimal. Moreover, $D(\mathcal{CS}(S[a,b]), C) < D(\mathcal{CS}(S[a,b]), C') < \delta$ holds. (ii) $\Rightarrow$: From $[a,b]$ being C-closed and from $c \in C$ follows directly that $S[a-1] \notin C$ and $S[b+1] \notin C$. Furthermore, we have $D(C, \mathcal{CS}(S[a,b])) \leq \delta$. Removing a single character from C that is not contained in $\mathcal{CS}(S[a,b])$ reduces the distance by one. Therefore, $D(C', \mathcal{CS}(S[a,b])) < \delta$ holds. $\Leftarrow$: It follows from $c \neq S[a-1]$, $c \neq S[b+1]$ and $C = C' \cup \{c\}$ that $[a,b]$ is C-optimal. Furthermore, we have $D(C', \mathcal{CS}(S[a,b])) < \delta$. Adding a single character to C' increases this distance by at most one so that it cannot exceed δ. Therefore, $[a,b]$ is an optimal δ-location of C. □

Testing the elements of $\mathcal{C}$ for the conditions given in Observation 2, we can remove all non-inheritable intervals (line 9). Next, we show how we can compute the optimal δ-locations that cannot be inherited from the previous reference character set. These intervals are characterized as follows:

Observation 3. *Let C, $C' \subseteq \Sigma$ with $C = C' \cup \{c\}$ and $c \notin C'$. Every interval $[a,b]$ in a string S over Σ that is an optimal δ-location of C is an optimal δ-location of C' if and only if either:*

(i) $c \notin \mathcal{CS}(S[a,b])$, *or*

(ii) $c \in \mathcal{CS}(S[a,b])$, $D(\mathcal{CS}(S[a,b]),C) < \delta$ *and* $\exists\, p_\ell, p, p_r : a \le p_\ell < p < p_r \le b$, $S[p_\ell] \in C'$, $S[p] = c$, *and* $S[p_r] \in C'$.

Proof. *(i)* With $[a,b]$ being C-optimal, $S[a-1]$ and $S[b+1]$ are not in C and therefore not in C'. Due to $c \notin \mathcal{CS}(S[a,b])$, $[a,b]$ has the same left- and right-most essential positions with respect to C and C'. Thus $[a,b]$ is C'-optimal. Also, $D(\mathcal{CS}(S[a,b]),C') = D(\mathcal{CS}(S[a,b]),C) - 1 \le \delta - 1$ holds. Hence, $[a,b]$ is an optimal δ-location of C'.
(ii) $\Rightarrow$: From $D(\mathcal{CS}(S[a,b]),C') \le \delta$ follows $D(\mathcal{CS}(S[a,b]),C) < \delta$ because $c \in C$ and $c \in \mathcal{CS}(S[a,b])$, but $c \notin C'$. Let p_ℓ and p_r be left-most/right-most essential positions of $[a,b]$ with respect to C'. Then $S[p_\ell] \neq c$, $S[p_r] \neq c$ and $\exists p$ with $p_\ell < p < p_r$ and $S[p] = c$ due to C'-optimality and $c \in \mathcal{CS}(S[a,b])$. $\Leftarrow$: $D(\mathcal{CS}(S[a,b]),C') \le \delta$ holds, as $D(\mathcal{CS}(S[a,b]),C') = D(\mathcal{CS}(S[a,b]),C) + 1$ and $D(\mathcal{CS}(S[a,b]),C) < \delta$. It follows from $p_\ell < p < p_r$ that c is contained in the C'-essential subinterval of $[a,b]$. Since C and C' differ only in c, $[a,b]$ has to be C'-optimal □

An important result of this observation is that only intervals with an occurrence of c need to be considered for the computation of non-inheritable optimal δ-locations. Moreover, we can infer that there are exactly two types of them: intervals whose character set has exactly distance δ to C, and intervals with left-most and/or right-most essential character c that contain no "inner occurrences" of c, i.e. positions that are separated from both interval boundaries by characters from C other than c. Only these intervals need to be computed anew for every reference interval.

To detect these non-inheritable optimal δ-locations, we identify for each occurrence p of c in S_2 all intervals around p that contain at most δ different unmarked characters. All other intervals either contain no occurrence of c or have a distance to C greater than δ. To find these intervals, we compute positions to the left and right of p with increasing numbers $x, y \ge 1$ of unmarked characters (line 13):

$$l_x = l_x(p) = \max(\{l \mid S_2[l,p] \text{ contains } x \text{ different unmarked characters}\} \cup \{0\})$$
$$r_y = r_y(p) = \min(\{r \mid S_2[p,r] \text{ contains } y \text{ different unmarked characters}\} \cup \{|S_2|+1\}).$$

By definition, the substrings $S_2[l_x+1, r_y-1]$ contain at most $x+y-2$ different characters not occurring in $S_1[i,j]$. (The number is smaller if the same unmarked characters occur left and right of p.) Clearly, not all of them are C-optimal or fulfill the distance constraint, as the example in Figure 4 illustrates. But together, they form a superset of the C-optimal δ-locations around p. Thus, we only need to test every interval of the form $[l_x+1, r_y-1]$ for being an optimal δ-location for the reference interval $[i,j]$ and add it to $\mathcal{C}$ if it passes this test and cannot be inherited from the previous reference character set (line 16).

It follows from Observations 2 and 3 that once all occurrences of c have been processed, $\mathcal{C}$ contains all optimal δ-locations of C. To avoid that intervals with multiple occurrences of c are redundantly inserted, we add a rule by which only

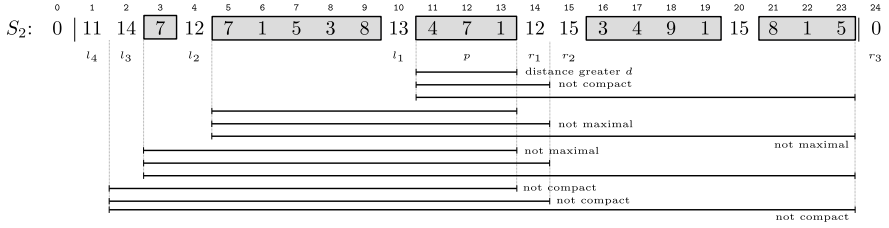


Fig. 4. All intervals of the form $[l_x+1, r_y-1]$ for $p = 12$, $\delta = 3$. For intervals that are not an optimal δ-location of $\mathcal{CS}(S_1[3, 11])$ the missing property is given.

intervals in which p is the left-most occurrence of c are added to $\mathcal{C}$. From the previous considerations, it follows directly that the presented algorithm solves Problem 1.

3.3 Implementation Details and Data Structures

Before we can analyze the time complexity of Algorithm 1, we need to have a closer look at some of the involved operations and the data structures that can be employed for their efficient implementation. A more detailed description can be found in [5].

Precomputation of $l_1, \ldots, l_{\delta+1}$ and $r_1, \ldots, r_{\delta+1}$: We begin with the identification of unmarked positions around the newly marked positions in S_2 (line 13). An efficient approach to detect these positions, was recently introduced in the context of median gene cluster computations [3]. It was shown that the values l_x and r_y, $1 \leq x, y \leq \delta+1$, can be precomputed for all positions p in S_2 once the left border i of the next class of reference intervals $[i, j]$, $j \geq i$, is fixed. These values are stored in two $\delta \times |S_2|$ tables L and R such that $L[p][x] = l_x(p)$ and $R[p][y] = r_y(p)$ holds. Once i is shifted to $i + 1$, L and R need to be updated which was shown to be possible in time $O(n(\delta + 1))$. For a complete run of Algorithm 1 the time spent on computing and maintaining L and R accumulates to $O(n^2(\delta + 1))$.

Identification of left- and right-most essential positions: To test the candidate intervals $[l_x+1, r_y-1]$ for being C-compact, we need to know their C-essential subintervals. We observe that the left-most essential position of an interval depends only on its left border and C while the right-most essential position depends only on the right border and C. Therefore, we need to determine for each l_x and r_y only one left-most (respectively right-most) essential position that is valid for all intervals with left border l_x+1, respectively right border r_y-1. We precompute and update these values parallel to the values of l_x and r_y using two additional tables L' and R' that are of the same format as L and R. It can be shown that these tables can be maintained along with L and R without increasing the asymptotic time complexity [5].

Testing intervals for being optimal δ-locations: Next, we show how a candidate interval $[l_x+1, r_y-1]$ can be tested for being optimal δ-location of C (line 15). First, we test for interval maximality. This test can be done in constant time if we compute in a preprocessing for every position p in S_2 the next and previous occurrence of $S_2[p]$ in S_2 and store this information in two static arrays of length $|S_2|$. Every maximal interval of the form $[l_x+1, r_y-1]$ is automatically C-closed, because neither $S_2[l_x]$ nor $S_2[r_y]$ can be contained in C. C-compactness can be tested in constant time by comparing the entries in NUM for $[l_x+1, r_y-1]$ and its C-essential subinterval. To test the distance constraint, we compute $D(\mathcal{CS}(S_1[i,j]), \mathcal{CS}(S_2[l_x+1, r_y-1]))$ which equals $|C| - |\mathcal{CS}(S_2[l_x+1, r_y-1])|$ plus twice the number of different unmarked characters in $S_2[l_x+1, r_y-1]$. $|\mathcal{CS}(S_2[l_x+1, r_y-1])|$ can be looked-up in NUM and $|C|$ can be tracked during reference interval generation. If candidate intervals $[l_x+1, r_y-1]$ are enumerated systematically, also the number of unmarked characters is available such that the complete distance computation takes constant time. Finally, we need to test whether $[l_x+1, r_y-1]$ is already contained in $\mathcal{C}$. Rather than testing condition (ii) of Observation 3, we generate first a separate list of the newly detected C-optimal δ-locations and then merge it with the inherited $\mathcal{C}$. Both lists can be kept sorted based on the index positions of their elements if the occurrences of c in S_2 and the corresponding $[l_x+1, r_y-1]$ are always processed from left to right.

The second step where intervals are tested for being optimal δ-locations is in line 9 of Algorithm 1. To identify intervals in $\mathcal{C}$ that contain an occurrence of c, we perform a combined iteration through $\mathcal{C}$ and POS$[c]$. For the intervals that contain no occurrence of c, condition (ii) of Observation 2 can be tested in constant time if we track for each interval the distance between its character set and the current C. To achieve this, we remember the initial distance when the element is added to $\mathcal{C}$ an increment it by one each time it does not contain an occurrence of the recent c.

3.4 Complexity Analysis

We now have all prerequisites to analyze the complexity of Algorithm 1. It follows from the analysis of the CI Algorithm that the generation of reference intervals in S_1 and the marking of character occurrences in S_2 is in $O(n^2)$. The first interesting step in Algorithm 1 is the iteration through $\mathcal{C}$ to remove intervals that are not an optimal δ-location of the new reference character set (lines 8 to 10). The complete cost for the combined iterations through $\mathcal{C}$ and POS$[c]$ is $O(n^2 + \textit{output size})$. This is because every $c \in \Sigma$ is $O(n)$ times the most recent element in C and $n \cdot \sum_{c\in\Sigma} |\text{POS}[c]| = n^2$, while the elements of $\mathcal{C}$ belong to the output for the previous reference character set. The next step is the computation of the non-inherited optimal δ-locations (lines 11 to 19). Every position p in S_2 gets at most $O(n)$ times newly marked, such that in total the **for**-loop in line 11 is executed $O(n^2)$ times. Using tables L and R, the unmarked positions $l_1, \ldots, l_{\delta+1}$ and $r_1, \ldots, r_{\delta+1}$ are immediately available. As we have seen above, the precomputation and maintenance of these data structures is in

time $O(n^2(\delta+1))$. It follows the processing of candidate intervals of the form $[l_x+1, r_y-1]$. For each occurrence p, there are $O((\delta+1)^2)$ many of these intervals, while there are $|\text{POS}[c]|$ many occurrences of each c in S_2. With every position p in S_2 being marked at most n times, $O\big(n\cdot\sum_{c\in\Sigma}\big(|\text{POS}[c]|\cdot(\delta+1)^2\big)\big) = O(n^2(\delta+1)^2)$ intervals are considered during the complete run of Algorithm 1. Using the data structures described above, testing a single candidate for being an optimal δ-location of C takes constant time. Finally, the list of new optimal δ-locations is merged with $\mathcal{C}$. As both lists are sorted, the time spent on this operation equals the length of the new list plus the length of $\mathcal{C}$. Since all elements of $\mathcal{C}$ are optimal δ-locations of the current C, this accumulates to $O(n^2(\delta+1)^2 + \textit{output size})$ for the complete run of the algorithm. Concerning space complexity, table NUM is the most costly data structure. Based on these findings, we claim the following theorem:

Theorem 1. *Using the data structures described above, Algorithm 1 solves Problem 1 in time* $O(n^2(\delta+1)^2 + \textit{output size})$ *using* $O(n^2)$ *space.*

4 Reference-Based Approximate Common Intervals

The extension of Algorithm 1 to solve Problem 2 is straight-forward. As the search space of conserved reference sets is limited to character sets that have a location in one of the input genomes, we can reuse the reference interval generation to traverse the search space. We only need to run it one after the other on all strings in $\mathcal{S}$. To compute optimal δ-locations of a reference character set and to decide whether they are distributed over enough strings, we need to run the rest of the algorithm in parallel on the remaining strings tracking not only the optimal δ-locations in each string but also a counter for the number of strings that contain a δ-location of the current reference character set C. If this number is at least q, C is reported along with its optimal δ-locations. The time complexity of the adapted algorithm increases to $O(k^2n^2(\delta+1)^2 + \textit{output size})$, while the space complexity increases to $O(kn^2)$.

5 Experimental Results

The experimental evaluation of the presented approach was performed on a test set containing the genomes of five γ-proteobacteria that we downloaded from the NCBI database [8]: *Buchnera aphidicola APS*, *Escherichia coli K12*, *Haemophilus influenzae Rd*, *Pasteurella multocida Pm70*, and *Xylella fastidiosa 9a5c*.

For grouping genes into homology families, we employed the GHOSTFAM tool [11]. Using the standard parameters, this program distributed the 11,184 genes occurring in the studied genomes into 5086 gene families of sizes between 1 and 63. All computations were performed on an 8 × 2.6 GHz AMD Opteron 8218 Dual-Core processor with 32 GB main memory.

The main focus of our evaluation is on the impact that the requirement of a reference occurrence has on approximate gene cluster prediction. To this end, we

estimated how many approximate gene clusters with a general consensus set for a given distance threshold have a reference-based consensus set that complies with the same distance threshold. To increase comparability to our pair-wise distance constrained reference gene clusters, we modified the sum-distance based median gene cluster algorithm [3] such that it computes for a combination of approximate gene cluster occurrences from k genomes, $([i_1, j_1], \ldots, [i_k, j_k])$, the optimal consensus set that minimizes the largest pairwise distance between the gene contents of the approximate occurrences and the consensus set C.

$$\max_{1 \leq \ell \leq k} D(C, \mathcal{CS}(S_\ell[i_\ell, j_\ell])) \leq \max_{1 \leq \ell \leq k} D(C', \mathcal{CS}(S_\ell[i_\ell, j_\ell])) \text{ for all } C' \subseteq \Sigma.$$

This corresponds to the computation of a so-called *center set* that is equivalent to the *closest string* problem [7]. Such a center set does not necessarily have a reference occurrence in the given genomes which leads to an exponential search space size as basically all combinations of the form $([i_1, j_1], \ldots, [i_k, j_k])$ need to be processed. However, in practice, filter techniques can be employed that rule out a large number of interval combinations without testing them explicitly if it is clear that their consensus will exceed the pre-defined distance threshold δ.

We ran the modified program for different combinations of δ and s, a minimum size threshold for the center-based consensus set. To estimate how far off the closest reference occurrence is, we computed for each "successful" interval combination, i. e. one whose center set complies with the distance threshold δ, the smallest distance to a reference-based consensus chosen among the character sets of the k intervals:

$$dist = \min_{\ell} \max_{m} D(CS(S_\ell[i_\ell, j_\ell]), CS(S_m[i_m, j_m]))$$

Afterwards, we computed the fraction of successful interval combinations for which such a reference based consensus set complies with the center distance threshold δ or with a slightly increased threshold. We observe that for the vast majority of interval combinations the best reference-based consensus complies with the initial distance threshold. All other combinations have a reference-based consensus that complies with a slightly increased distance threshold.

To assess the practical performance of our program for reference gene cluster computation, we ran it for all distance threshold values between 1 and 12 and a fixed minimum cluster size $s = 4$. As the minimum cluster size is not used in the main part of the algorithm, but only to filter afterwards too small reference gene clusters, we do not distinguish runtimes for different values of s. We observe that runtimes increase slowly with increasing values of δ and range from 36s to 42s. The number of detected reference-based gene clusters ranges from 339 to 5410 when counting only those reference occurrences that are not nested into a larger one. Among the predicted clusters are many known gene clusters like the operon for ATP biosynthesis and the cell division and cell wall biosynthesis gene

Table 1. Computation of approximate gene clusters with a center-based consensus. The output size (# non nested interval combinations) refers to the number of interval combinations that have a center-based consensus for the given distance threshold δ and are not completely nested into another interval combination. Additionally, the fraction of these interval combinations that have a reference-based consensus for the initial δ or slightly higher values is shown. No results are given for computations that did not finish within 24 hours.

		$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$
$s = 4$	running time	4s	59m 16s	$>$ 24h	$>$ 24h
	# non nested interval combinations	162	$2.6 \cdot 10^4$	-	-
	reference-based consensus for δ	100.0%	99.8%	-	-
	reference-based consensus for $\delta + 1$	-	99.9%	-	-
	reference-based consensus for $\delta + 2$	-	100.0%	-	-
$s = 6$	running time	4s	32s	6h 0m	$>$ 24h
	# non nested interval combinations	61	856	$1.6 \cdot 10^4$	-
	reference-based consensus for δ	100.0%	99.6%	97.8%	-
	reference-based consensus for $\delta + 1$	-	99.9%	99.8%	-
	reference-based consensus for $\delta + 2$	-	100.0%	100.0%	-
$s = 8$	running time	3s	6s	12m 51s	$>$ 24h
	# non nested interval combinations	16	181	2560	-
	reference-based consensus for δ	100.0%	100.0%	98.0%	-
	reference-based consensus for $\delta + 1$	-	-	100.0%	-
$s = 10$	running time	3s	5s	14s	3h 21m
	# non nested interval combinations	9	71	1120	$1.2 \cdot 10^4$
	reference-based consensus for δ	100.0%	100.0%	97.1%	90.3%
	reference-based consensus for $\delta + 1$	-	-	100.0%	96.8%
	reference-based consensus for $\delta + 2$	-	-	-	99.4%
	reference-based consensus for $\delta + 3$	-	-	-	100.0%

cluster, but also some clusters for which no obvious classification exists. We are currently doing deeper analysis of these predictions.

6 Conclusion

In this paper, we presented a polynomial-time algorithm for the detection of approximate gene clusters in multiple genomes. The limitation of the search space to polynomial size was achieved by the use of a gene cluster model that requires the existence of a reference occurrence. Unlike a previous algorithm, our algorithm detects the complete solution set of reference-based approximate common intervals.

We evaluated the relevance and performance of our approach on real genome data. In this initial study, we have shown that our gene cluster predictions are highly comparable to a more general approach that does not rely on reference occurrences, while using only a fraction of the running time. Only for a very small fraction of gene cluster predictions, the reference-based cluster version can only be found if the distance threshold is slightly increased. Since runtimes increase only slowly with δ, this is not a problem in practice. In general, the polynomial time complexity of our algorithm raises the opportunity to search for approximate gene clusters with very diverse conservation patterns that are not detectable with those parameter settings currently feasible for non-reference based approaches.

References

1. Amir, A., Gasieniec, L., Shalom, R.: Improved approximate common interval. Inf. Process. Lett. 103(4), 142–149 (2007)
2. Bergeron, A., Corteel, S., Raffinot, M.: The algorithmic of gene teams. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 464–476. Springer, Heidelberg (2002)
3. Böcker, S., Jahn, K., Mixtacki, J., Stoye, J.: Computation of median gene clusters. J. Comp. Biol. 16(8), 1085–1099 (2009)
4. He, X., Goldwasser, M.H.: Identifying conserved gene clusters in the presence of homology families. J. Comp. Biol. 12, 638–656 (2005)
5. Jahn, K.: Approximate Common Intervals Based Gene Cluster Models. PhD thesis, Bielefeld University, Bielefeld (2010)
6. Ling, X., He, X., Xin, D.: Detecting gene clusters under evolutionary constraint in a large number of genomes. Bioinformatics 25(5), 571 (2009)
7. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. In: Vingron, M., Wong, L. (eds.) RECOMB 2008. LNCS (LNBI), vol. 4955, pp. 396–409. Springer, Heidelberg (2008)
8. Pruitt, K.D., Tatusova, T., Maglott, D.R.: NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. Nucleic Acids Res. (2006)
9. Rahmann, S., Klau, G.W.: Integer linear programs for discovering approximate gene clusters. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 298–309. Springer, Heidelberg (2006)
10. Schmidt, T., Stoye, J.: Quadratic time algorithms for finding common intervals in two and more sequences. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 347–358. Springer, Heidelberg (2004)
11. Schmidt, T., Stoye, J.: Gecko and GhostFam – rigorous and efficient gene cluster detection in prokaryotic genomes. In: Bergman, N. (ed.) Comparative Genomics, ch. 12. Methods in Molecular Biology, vol. 2, pp. 165–182. Humana Press, Totowa (2007)

The Complexity of the Gapped Consecutive-Ones Property Problem for Matrices of Bounded Maximum Degree

Ján Maňuch[1,2,⋆] and Murray Patterson[2,⋆⋆]

[1] Department of Mathematics, Simon Fraser University, Burnaby, BC, Canada
[2] Deptartment of Computer Science, UBC, Vancouver, BC, Canada

Abstract. The Gapped Consecutive-Ones Property (C1P) Problem, or the (k, δ)-C1P Problem is: given a binary matrix M and integers k and δ, decide if the columns of M can be ordered such that each row contains at most k blocks of 1's, and no two neighboring blocks of 1's are separated by a gap of more than δ 0's. This problem was introduced in [3]. The classical polynomial-time solvable C1P Problem is equivalent to the $(1, 0)$-C1P problem. It has been shown that for every unbounded or bounded $k \geq 2$ and unbounded or bounded $\delta \geq 1$, except when $(k, \delta) = (2, 1)$, the (k, δ)-C1P Problem is NP-complete [10,6].

In this paper we study the Gapped C1P Problem with a third parameter d, namely the bound on the maximum number of 1's in any row of M, or the bound on the maximum *degree* of M. This is motivated by problems in comparative genomics and paleogenomics, where the genome data is often sparse [4]. The (d, k, δ)-C1P Problem has been shown to be polynomial-time solvable when all three parameters are fixed [3]. Since fixing d also fixes k $(k \leq d)$, the only case left to consider is the case when δ is unbounded, or the (d, k, ∞)-C1P Problem. Here we show that for every $d > k \geq 2$, the (d, k, ∞)-C1P Problem is NP-complete.

1 Introduction

Let M be a binary matrix with m rows and n columns. A *block* in a row of m is a maximal sequence of consecutive entries containing 1. A *gap* is a sequence of consecutive 0's that separates two blocks, where the size of a gap is the length of this sequence of 0's. The *degree* of a row of m is the number of 1's in that row. Matrix M is said to have the *Consecutive-Ones Property* (C1P) if its columns can be permuted such that each row contains one block (there are no gaps in this case). Deciding if a binary matrix has the C1P can be done in linear time [11].

Among its many applications, the C1P has been widely used in molecular biology, in relation with physical mapping [1] and the reconstruction of ancestral genomes [4]. However, a common problem in such applications is that matrices obtained from experiments do not have the C1P, often due to small errors in

⋆ Research supported by NSERC Discovery Grant.
⋆⋆ Research supported by NSERC PGS-D.

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 278–289, 2010.

the data [6,4]. It is this lack of robustness of the C1P to small changes that led researchers to consider relaxing the consecutivity condition of the 1's in each row, but rather to allow gaps, with some restriction on the nature of these gaps [6,3]. In [3], the authors defined the Gapped C1P Problem, or the (k, δ)-C1P Problem: given binary matrix M and two integers k and δ, to decide if the columns of M can be ordered such that each row contains at most k blocks, and no two neighboring blocks of 1's are separated by a gap of size more than δ. They showed that for every $k \geq 2, \delta \geq 1, (k, \delta) \neq (2, 1)$, the (k, δ)-C1P Problem is NP-complete [10], leaving open only the complexity of the $(2, 1)$-C1P case. The problem remains NP-complete even if one of the two parameters is unbounded: (i) for every $k \geq 2$, the (k, ∞)-C1P Problem is NP-complete [6], and (ii) for every $\delta \geq 1$, the (∞, δ)-C1P Problem is NP-complete [10].

The above NP-completeness results involve constructions with rows of large degree, however matrices obtained from experiments are often sparse [4]. This could give hope that problems are polynomial-time solvable for real data. Formally, we have the (d, k, δ)-C1P Problem: given matrix M where the bound on the maximum degree of any row of M is d, decide if it has the (k, δ)-C1P. We call a permutation π of the columns of M that witness this property a (d, k, δ)-*consecutive order*; that the matrix M' resulting from this permutation is (d, k, δ)-*consecutive*, or that it is *consecutive w.r.t.* π; and that M is (d, k, δ)-C1P, or has the (d, k, δ)-C1P. If all three parameters are fixed, the problem is related to the classical Graph Bandwidth Problem, and thus can be solved in polynomial time using a variant of an algorithm of Saxe [13,3].

In this paper we study the complexity of the (d, k, δ)-C1P Problem when one or more of these parameters are unbounded. The cases with d unbounded were considered in [10]. Since fixing d also fixes k $(k \leq d)$, the only case left to consider is the case when δ is unbounded, or the (d, k, ∞)-C1P Problem. Here we show that in every non-trivial case, this problem is NP-complete, i.e., for every $d > k \geq 2$, the (d, k, ∞)-C1P Problem is NP-complete. Since d is the bound on the maximum number of 1's in any row of M, it is enough to show that for every $d \geq 3$, the $(d, d-1, \infty)$-C1P Problem is NP-complete. Indeed, a matrix of bounded maximum degree d is also of bounded maximum degree d', for $d' \geq d$. However, for every $d > k \geq 2$, the complexity of the (d, k, ∞)-C1P Problem for d-uniform matrices (each row has degree d) remains open. Note that if $d = 2$, the problem becomes the C1P Problem, and if $d \leq k$, then any order of the columns of M is a valid solution, since no row can have more than d blocks of 1's.

This paper is structured as follows. In Section 2, we show an auxiliary result on fixing the order of selected columns of a matrix with one swap allowed, which we then use in Section 3 to show that for all $d \geq 4$, the $(d, d-1, \infty)$-C1P Problem is NP-complete, by giving a reduction from 3SAT. In Section 4, we show that the $(3, 2, \infty)$-C1P Problem is NP-complete, by showing equivalence to a new hypergraph covering problem, and then showing that this problem is NP-complete. Note that the first proof does not work for $d = 3$ and the second proof is not easily adaptable for $d \geq 4$. Finally, in Section 5, we conclude the paper with some remarks and discuss future work.

2 Fixing a Block of Columns with One Swap

In this section we present a theorem that provides a key building block in our NP-completeness constructions: namely that a set of rows can be added to a binary matrix M that ensures that a selected set S of columns in M remains "together" and in fixed order, except that exactly two columns of S may swap positions, in any $(d, d-1, \infty)$-consecutive order of M. We will use this in the following section in the reduction from 3SAT, where these two possible orders of the columns in S will represent the truth values of a variable.

To state the result we need the following definitions. We say that columns $S = \{c_1, \ldots, c_k\}$ appear *together* in an order π of the columns of M if between any two columns in S there are only columns from S in π. To specify a row in binary matrix M, we use the convention of only listing in the square brackets, the columns that contain `1` in this row. For example, $[1, 5, 8]$ represents a row with ones in columns 1, 5 and 8, and zeroes everywhere else. This list might also contain a set of columns. For example, if $S = \{2, 4, 6\}$ then $[S, 7]$ is equivalent to $[2, 4, 6, 7]$.

Theorem 1. *For every $d \geq 3$ and every $1 < j < 2d-2$, given matrix M on $n \geq 2d$ columns, $\binom{2d}{d} - d - 2$ rows of degree d can be added to M to force $2d$ selected columns to appear together and in fixed order (or the reverse order), with the exception that the j-th and $(j+2)$-nd selected columns may swap positions, in any $(d, d-1, \infty)$-consecutive order of M.*

Proof. Let $S = \{\overline{1}, \ldots, \overline{2d}\}$ be the set of $2d$ selected columns. An order of the columns of M is called *normal* if the selected columns appear together and in the order $\overline{1}, \ldots, \overline{2d}$ or its reversal, and is called $(\overline{j}, \overline{j+2})$-*swapped* if the selected columns appear together and in the order $\overline{1}, \ldots, \overline{j-1}, \overline{j+2}, \overline{j+1}, \overline{j}, \overline{j+3}, \ldots, \overline{2d}$ or its reversal. Let $\mathcal{N}_d$ be the set of all normal orders of M and $\mathcal{S}_{d,j}$ be the set of all $(\overline{j}, \overline{j+2})$-swapped orders. The basic idea of the construction is very simple: we will construct matrix $M_{d,j}$ (on $n \geq 2d$ columns) by adding all rows which (i) have d `1`'s in the columns in S and `0`'s everywhere else and (ii) are $(d, d-1, \infty)$-consecutive w.r.t. all normal and $(\overline{j}, \overline{j+2})$-swapped orders of $M_{d,j}$. Note that since in any normal ($(\overline{j}, \overline{j+2})$-swapped) order the selected columns appear together, if a row is $(d, d-1, \infty)$-consecutive w.r.t. one normal ($(\overline{j}, \overline{j+2})$-swapped) order then it is $(d, d-1, \infty)$-consecutive w.r.t. all such orders. The difficult part will be to show that every $(d, d-1, \infty)$-consecutive order of the constructed matrix is either normal or $(\overline{j}, \overline{j+2})$-swapped. In what follows, we will first construct the set of rows which satisfy the above conditions (i) and (ii), and then prove the claim for this set of rows.

First, we will identify the rows with d `1`'s in the selected columns which are not $(d, d-1, \infty)$-consecutive w.r.t. at least one normal order. We will call such rows $\mathcal{N}_d$-*forbidden*. In general, given a set of orders $\mathcal{P}$ of a matrix, a row is $\mathcal{P}$-*forbidden* if it contains d `1`'s in selected columns and zeroes everywhere else, and it is not $(d, d-1, \infty)$-consecutive w.r.t. at least one order in $\mathcal{P}$. Let $F_{\mathcal{P}}$ be the set of all $\mathcal{P}$-forbidden rows. There are $\binom{2d}{d}$ unique rows containing `1`'s in d selected columns and zeros in all other columns. Let R_d be the set of all

of these rows. Note that, by definition, for any $\mathcal{P}$, $F_{\mathcal{P}} \subseteq R_d$. The $\mathcal{N}_d$-forbidden rows are exactly the rows which do not contain 1's in any pair $(\overline{i}, \overline{i+1})$ of consecutive selected columns. Since they contain d 1's in selected columns and selected columns appear together in any normal order, they can have at most one gap of size 2, and the remaining gaps are of size 1 (in any normal order). In addition, if they have a gap of size 2, they contain 1's in columns $\overline{1}$ and $\overline{2d}$. Hence, in each $\mathcal{N}_d$-forbidden row, the columns containing 1's form a (possibly empty) progression of odd numbered selected columns followed by a (possibly empty) progression of even numbered selected columns. If in an $\mathcal{N}_d$-forbidden row both progressions are non-empty, then there is a gap of size 2 between the last 1 of the first progression, and the first 1 of the second progression. Formally, the $\mathcal{N}_d$-forbidden rows are $f_i^{(d)} = [\overline{1}, \overline{3}, \ldots, \overline{2i-1}, \overline{2i+2}, \overline{2i+4}, \ldots, \overline{2d}]$ (a progression of i odd numbered columns followed by a progression of $d-i$ even numbered columns of S), where $i = 0, \ldots, d$, i.e., $F_{\mathcal{N}_d} = \{f_i^{(d)} \mid i = 0, \ldots, d\}$. Note that $f_0^{(d)} = [\overline{2}, \overline{4}, \ldots, \overline{2d}]$ and $f_d^{(d)} = [\overline{1}, \overline{3}, \ldots, \overline{2d-1}]$.

Next, we are going to identify the $\mathcal{S}_{d,j}$-forbidden rows. These rows can be easily obtained from the $\mathcal{N}_d$-forbidden rows by swapping values in columns $\overline{j}$ and $\overline{j+2}$ in each of these rows. Note that if row $f_i^{(d)} \in F_{\mathcal{N}_d}$ contains the same value in columns $\overline{j}$ and $\overline{j+2}$ then the row stays the same after swapping these values. The values of row $f_i^{(d)}$ in columns $\overline{j}$ and $\overline{j+2}$ differ if and only if the gap of size 2 in $f_i^{(d)}$ contains either $\overline{j}$ or $\overline{j+2}$. This happens only if $j = 2i-1$ or $j = 2i$ (depending on parity of j), i.e., if $i = \lceil j/2 \rceil$. Hence, there is exactly one row in $F_{\mathcal{S}_{d,j}} \setminus F_{\mathcal{N}_d}$ and it can be obtained from $f_{\lceil j/2 \rceil}^{(d)}$ by swapping values in the columns $\overline{j}$ and $\overline{j+2}$. This row is $f^{(d,j)} = [\overline{1}, \overline{3}, \ldots, \overline{j-2}, \overline{j+2}, \overline{j+3}, \overline{j+5}, \ldots, \overline{2d}]$, if j is odd, and $f^{(d,j)} = [\overline{1}, \overline{3}, \ldots, \overline{j-1}, \overline{j}, \overline{j+4}, \overline{j+6}, \ldots, \overline{2d}]$, if j is even. Hence, $F_{\mathcal{N}_d \cup \mathcal{S}_{d,j}} = F_{\mathcal{N}_d} \cup F_{\mathcal{S}_{d,j}} = F_{\mathcal{N}_d} \cup \{f^{(d,j)}\}$. The constructed matrix $M_{d,j}$ contains all rows in $R_d \setminus F_{\mathcal{N}_d \cup \mathcal{S}_{d,j}}$. In what follows we will refer to rows $F_{\mathcal{N}_d \cup \mathcal{S}_{d,j}}$ simply as the *forbidden* rows. Table 1 shows the forbidden rows for $d = 3$ and $j = 2$.

Let π be any $(d, d-1, \infty)$-consecutive order of matrix $M_{d,j}$. Let s_i be the i-th selected column in π, for every $i = 1, \ldots, 2d$. We will prove two claims about π.

Table 1. Forbidden rows $F_{\mathcal{N}_3 \cup \mathcal{S}_{3,2}}$ which are not used to construct matrix $M_{3,2}$. The first four rows are the $\mathcal{N}_3$-forbidden rows and $f_0^{(3)}, f_2^{(3)}, f_3^{(3)}, f^{(3,2)}$ are the $\mathcal{S}_{3,2}$-forbidden rows. Note that the last row can be obtained from row $f_1^{(3)}$ by swapping values in selected columns $\overline{2}$ and $\overline{4}$.

	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	non-selected columns
$f_0^{(3)}$	0	1	0	1	0	1	0
$f_1^{(3)}$	1	0	0	1	0	1	0
$f_2^{(3)}$	1	0	1	0	0	1	0
$f_3^{(3)}$	1	0	1	0	1	0	0
$f^{(3,2)}$	1	1	0	0	0	1	0

Claim 1. *Columns* $\overline{1}$ *and* $\overline{2d}$ *are the first and the last selected columns in* π*, i.e.,* $\{s_1, s_{2d}\} = \{\overline{1}, \overline{2d}\}$.

Proof. Assume that column $\overline{1} \notin \{s_1, s_{2d}\}$. Consider the rows obtained from the $\mathcal{N}_d$-forbidden rows by permuting values according to π, i.e., rows $f^{(d)}_{i,\pi} = [s_1, s_3, \ldots, s_{2i-1}, s_{2i+2}, s_{2i+4}, \ldots, s_{2d}]$, where $i = 0, \ldots, d$. Since these rows are not $(d, d-1, \infty)$-consecutive w.r.t. π, as adding (non-selected) columns with 0's between selected columns cannot create a pair of adjacent 1's, they are $\{\pi\}$-forbidden. At least two of these rows have a 0 in column $\overline{1}$: if column $\overline{1}$ is in an even position (when considering only the order of selected columns in π), i.e., if $s_{2k}{=}\overline{1}$ for some integer k, take rows $f^{(d)}_{d-1,\pi}$ and $f^{(d)}_{d,\pi}$, otherwise take rows $f^{(d)}_{0,\pi}$ and $f^{(d)}_{1,\pi}$. At least one of them is not forbidden, since column $\overline{1}$ contains a 0 only in one forbidden row ($f^{(d)}_0$). It follows that the order π is not $(d, d-1, \infty)$-consecutive, a contradiction. Hence, $\overline{1} \in \{s_1, s_{2d}\}$ (and by symmetry, also $\overline{2d} \in \{s_1, s_{2d}\}$). □

Claim 2. *If* $j < 2d-3$*, then* π *is* $\cdots, \overline{1}, \text{“}\ldots, \overline{2d-1}, \cdots, \overline{2d}, \cdots$ *(or the reverse order), where “*$\cdots$*” represents any sequence of non-selected columns, and “*$\ldots$*” any sequence of columns.*

Proof. First, assume that $\overline{2d-1} \notin \{s_2, s_{2d-1}\}$. Consider the $\{\pi\}$-forbidden rows $f^{(d)}_{i,\pi}$ defined in the proof of Claim 1. At least two of them contain a 1 in column $\overline{2d-1}$: if k is even, take rows $f^{(d)}_{0,\pi}$ and $f^{(d)}_{1,\pi}$, otherwise take rows $f^{(d)}_{d-1,\pi}$ and $f^{(d)}_{d,\pi}$. Since $j < 2d-3$, there is only one forbidden row ($f^{(d)}_d$) with a 1 in column $\overline{2d-1}$. Hence, one of these two rows is not forbidden, which leads to a contradiction, since this row is $\{\pi\}$-forbidden. Hence, $\overline{2d-1} \in \{s_2, s_{2d-1}\}$.

It follows, by Claim 1, that the only other possible order (up to reversal) is $\cdots, \overline{1}, \cdots, \overline{2d-1}, \ldots, \overline{2d}, \cdots$. Consider row $r = [s_1, s_3, \ldots, s_{2d-1}]$. Obviously, this row is $\{\pi\}$-forbidden. Since r contains 0's in columns $s_{2d} = \overline{2d}$ and $s_2 = \overline{2d-1}$, it is not forbidden, a contradiction. The claim follows. □

In what follows, we will show by induction on d that for any $j \in \{2, \ldots, 2d-3\}$, any $(d, d-1, \infty)$-consecutive order of $M_{d,j}$ is either normal or $(\overline{j}, \overline{j+2})$-swapped. The base case can be easily checked by enumeration of all $6! \cdot 2^5$ types of orders of $M_{3,j}$ (6! orders of selected columns and 2^5 possibilities depending on whether pairs of adjacent selected columns are separated by at least one non-selected column or not).

Claim 3. *For any* $j = 2, 3$*, any* $(3, 2, \infty)$*-consecutive order of* $M_{3,j}$ *is either normal or* $(\overline{j}, \overline{j+2})$*-swapped.*

Now, as the induction hypothesis: assume that the property holds for d' ($d' \geq 3$) and any $j' \in \{2, \ldots, 2d'-3\}$. We will show that it also holds for $d = d'+1$ and any $j \in \{2, \ldots, 2d-3\}$. Since $d \geq 4$, by symmetry, we can assume that $j < 2d-4 = 2d'-2$. Consider a submatrix M' of $M_{d,j}$ constructed as follows. First, keep only rows with 0 in column $\overline{2d-1}$ and 1 in column $\overline{2d}$. Then, remove columns $\overline{2d-1}$ and $\overline{2d}$, cf. Figure 1. We will show that $M' = M_{d',j}$. It is easy to

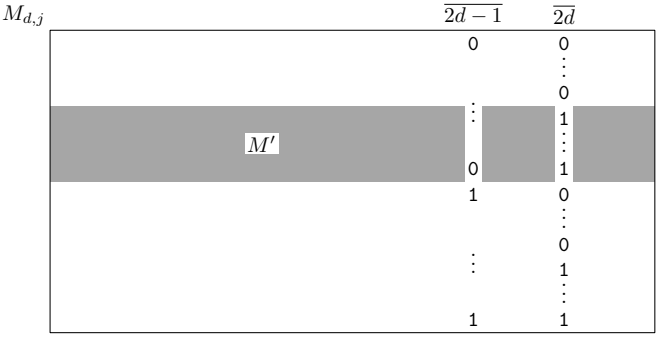


Fig. 1. An illustration of the construction of matrix M' (shaded area) from $M_{d,j}$

see that every row in M' is in $R_{d'}$: we removed exactly one of $d = d' + 1$ 1's by removal of columns $\overline{2d-1}$ and $\overline{2d}$, cf. Figure 1. The rows chosen from $M_{d,j}$ must have a pair of adjacent 1's in any normal (respectively, any $(\overline{j}, \overline{j+2})$-swapped) order. Neither of the two 1's in this pair can be in columns $\overline{2d-1}$ or $\overline{2d}$, since these two columns are the last two selected columns in all such orders. Hence, the adjacent pair of 1's is not removed by removal of columns $\overline{2d-1}$ and $\overline{2d}$. It follows that each row in M' is $(d', d'-1, \infty)$-consecutive w.r.t. all normal and $(\overline{j}, \overline{j+2})$-swapped orders of selected columns $\{\overline{1}, \ldots, \overline{2d'}\}$. Hence, every row in M' is in $M_{d',j}$. On the other hand, for every row in $M_{d',j}$, adding column $\overline{2d-1}$ containing 0 and column $\overline{2d}$ containing 1 will produce a row which is in R_d, and is $(d, d-1, \infty)$-consecutive w.r.t. all normal and $(\overline{j}, \overline{j+2})$-swapped orders. Hence, $M' = M_{d',j}$.

Now, consider a $(d, d-1, \infty)$-order π of $M_{d,j}$. By Claim 2, we can assume that the order of columns in π is

$$\cdots, \overline{1}, \ldots, \overline{2d-1}, \cdots, \overline{2d}, \cdots .$$

Let π' be the restriction of π obtained by removing columns $\overline{2d-1}$ and $\overline{2d}$. Since $\overline{2d-1}$ and $\overline{2d}$ are the last selected columns in π and they contained 0 and 1, respectively, in every row chosen to construct M', M' is $(d', d'-1, \infty)$-consecutive w.r.t. π'. By the induction hypothesis, π' is normal or $(\overline{j}, \overline{j+2})$-swapped, and hence, in π, selected columns in $\{\overline{1}, \ldots, \overline{2d-2}\}$ appear together and in the order $\overline{1}, \ldots, \overline{2d-2}$ or $\overline{1}, \ldots, \overline{j-1}, \overline{j+2}, \overline{j+1}, \overline{j}, \overline{j+3}, \ldots, \overline{2d-2}$ (up to reversal). Now, it is enough to show that there are no non-selected columns between columns $\overline{2d-2}$ and $\overline{2d-1}$, and between $\overline{2d-1}$ and $\overline{2d}$.

Consider the row $[\overline{2}, \overline{4}, \ldots, \overline{2d-4}, \overline{2d-2}, \overline{2d-1}]$. This row is in $M_{d,j}$, since it is in R_d and it is $(d, d-1, \infty)$-consecutive w.r.t. all normal and $(\overline{j}, \overline{j+2})$-swapped orders (recall that we assume that $j < 2d-4$). However, if there is a non-selected column between columns $\overline{2d-2}$ and $\overline{2d-1}$ in π, this row is $\{\pi\}$-forbidden. Similarly, the row $[\overline{1}, \overline{3}, \ldots, \overline{2d-5}, \overline{2d-1}, \overline{2d}]$ shows that there is no non-selected column between $\overline{2d-1}$ and $\overline{2d}$ in π. Hence, in π is either normal or $(\overline{j}, \overline{j+2})$-swapped, which proves the induction step. □

3 Complexity of the $(d, d-1, \infty)$-C1P Problem

Here, we prove that for every $d \geq 4$, the $(d, d-1, \infty)$-C1P Problem is NP-complete by reducing from 3SAT.

Theorem 2. *For every $d \geq 4$, the $(d, d-1, \infty)$-C1P Problem is NP-complete.*

Proof. Consider $d \geq 4$. Since the $(d, d-1, \infty)$-C1P Problem is clearly in NP, the remainder of this proof is showing that the problem is NP-hard. Given a 3SAT formula ϕ on n variables and m clauses, we construct a matrix M_ϕ with $2dn+3m$ columns that has the $(d, d-1, \infty)$-C1P if and only if ϕ is satisfiable. For every variable x_i of ϕ, add the $2d$ columns $x_i^1, \ldots, x_i^{2d}$ to M_ϕ and the $\binom{2d}{d} - d - 2$ rows from Theorem 1 so that these $2d$ columns appear appear together and in fixed order, except that columns x_i^3 or x_i^5 may swap positions, in any $(d, d-1, \infty)$-consecutive ordering of M_ϕ. The normal order will correspond to this variable being *true*, and the order where x_i^3 and x_i^5 are swapped will correspond to this variable being *false*. For every clause c_j of ϕ, add the 3 new columns from the set $C_j = \{c_j^1, c_j^2, c_j^3\}$ to M_ϕ. For each variable x_i in the clause c_j we add row $[x_i^2, x_i^k, x_i^9, x_i^{11}, \ldots, x_i^{2d-1}, C_j \setminus \{c_j^\ell\}]$ to M_ϕ, where $k = 3$ if x_i has a positive occurrence in c_j and $k = 5$ otherwise, and ℓ is the position of x_i in c_j. The set of rows added to M_ϕ for clause $c_2 = \{x_1 \vee \neg x_4 \vee x_5\}$ when $d = 4$ is illustrated in Figure 2.

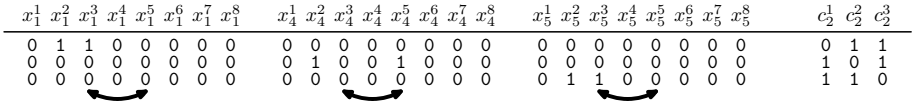


Fig. 2. The 3 rows of M_ϕ encoding clause $c_2 = \{x_1 \vee \neg x_4 \vee x_5\}$ when $d = 4$

If, in a $(d, d-1, \infty)$-consecutive ordering of M_ϕ, the ℓ-th literal of c_j (x_i or $\neg x_i$) is *false*, the columns x_i^2 and x_i^k are separated by columns with 0's, and hence, columns in $C_j \setminus \{c_j^\ell\}$ must be adjacent, otherwise there are $d-1$ gaps in the row for this literal, i.e., it is not $(d, d-1, \infty)$-consecutive. Since, in any ordering of the 3 columns of C_j, at least one pair of columns in C_j is not adjacent, at least one literal of c_j must be set to *true*. Note in Figure 2 that if more than one literal of c_2 is true, then columns c_2^1, c_2^2 and c_2^3 do not have to appear together in a $(4, 3, \infty)$-consecutive ordering of M_ϕ. It is easy to see that ϕ is satisfiable if and only if there is $(d, d-1, \infty)$-consecutive ordering of M_ϕ.

Since the number of columns and rows added to M_ϕ is linear in the number of variables and clauses of ϕ, this construction can be done in polynomial-time, and thus the $(d, d-1, \infty)$-C1P Problem is NP-hard. □

The above construction requires at least 4 ones per row: two in the variable block and two in the clause block. However, the problem is NP-complete also in the case when $d = 3$, but requires a different construction. We present this in the next section.

4 The $(3, 2, \infty)$-C1P Problem

We will first show that the following hypergraph covering problem is NP-complete. Note that a hypergraph $H = (V, E)$ is *3-uniform* when all of its hyperedges are *3-edges*, that is, hyperedges that contain exactly three vertices.

Definition 1. *A* graph covering *of a 3-uniform hypergraph $H = (V, E)$ is a graph $G = (V, E')$ such that there exists a surjective (onto) map $m : E \to E'$, such that for every $h \in E$, $m(h) \subset h$. Here, we say that edge $m(h)$* covers *the hyperedge h.*

Informally, a graph covering of a 3-uniform hypergraph is a graph constructed by picking one edge from each hyperedge.

Problem 1 (3-Uniform Hypergraph Covering By Paths Problem (3-UHC-P Problem)). *Given a 3-uniform hypergraph $H = (V, E)$, is there a graph covering of H which consists only of disjoint paths?*

Variations of this problem were defined in [7,8,9]. The first variation allowed the hypergraph to have also 2-edges and 4-edges (4-edges were covered by two parallel edges) and required that the graph covering contains only disjoint edges and vertices. This variation was shown to be polynomial-time solvable and provided an algorithm for a special version of haplotyping problem via galled-tree networks [7]. The second variation required all connected components of the graph covering to be paths of length at most 3. This variation was shown to be NP-complete [9]. A slightly more complex version of this was then used to show that in general the haplotyping problem via galled-tree networks is NP-complete [8].

We will show that our version of hypergraph covering problem is NP-complete. Later in this section we will show that this problem is equivalent to the $(3, 2, \infty)$-C1P Problem, thus showing that the $(3, 2, \infty)$-C1P Problem is also NP-complete.

Theorem 3. *The 3-UHC-P Problem is NP-complete.*

Proof. Clearly, the problem is in NP. We will show it is also NP-hard by reduction from 3SAT(3), a restricted version of 3SAT, proved NP-complete by Papadimitriou [12], in which every variable has exactly two positive and one negative occurrence in the clauses.[1] We will call a graph covering of a hypergraph a *valid covering* if it consists only of disjoint paths. Note that a valid covering does not contain vertices of degree 3 or more and does not contain cycles. Given 3SAT(3) formula ϕ with variables $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$, we now construct a 3-uniform hypergraph H_ϕ on at most $12n + 15m$ hyperedges which contains, among other vertices, a vertex for each literal of ϕ (there are $3n$ such vertices) that has a valid covering if and only if ϕ is satisfiable.

[1] We remark that the exact formulation of 3SAT(3) in [12] allows also variables with one positive and two negated occurences, but these can easily be converted to the other type of variables by replacing them with their negations in all clauses. Clearly, this does not affect the complexity of the problem.

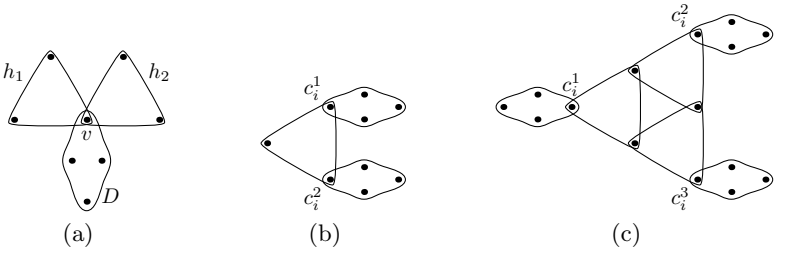


Fig. 3. (a) A simple dependency on coverings of two touching hyperedges enforced by a copy of D (depicted as a diamond). (b) The 2-clause and (c) 3-clause gadgets for clause c_i.

First we give an important building block that is used throughout this construction: the complete 3-uniform hypergraph D on 4 vertices. In any valid covering of D, there is no isolated vertex. Indeed, assume for contradiction that v is the isolated vertex in a valid covering G. Let u_1, u_2, u_3 be the remaining three vertices. Then there is a pair u_i, u_j such that $\{u_i, u_j\}$ is not an edge in G. However, no edge is covering hyperedge $\{v, u_i, u_j\}$, a contradiction. We will use several copies of D in the construction to introduce a dependency on coverings of touching hyperedges and depict them as diamonds in the figures. For instance, consider the hypergraph in Figure 3(a). Since in any valid covering G of this hypergraph, v is a member of an edge in D, at most one of the hyperedges h_1 and h_2 can "pick" an edge involving v, otherwise vertex v would have degree 3 or more.

Now to the main construction. Consider the instance ϕ of 3SAT(3) with variables $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$. We say that a clause *selects* one of its literals in a truth assignment of ϕ if this literal has value *true* in this assignment. Obviously, a truth assignment of ϕ is a satisfying truth assignment if and only if every clause selects at least one literal and for every $x \in X$, at most one of x and $\neg x$ is selected. We design a hypergraph H_ϕ composed of clause gadgets which will guarantee the first condition and variable gadgets which will ensure the second condition.

Figure 3(b)–(c) depicts the 2-clause and 3-clause gadgets, respectively. Given a valid covering of the clause gadget for clause c_i with literals c_i^1, c_i^2 (and c_i^3 for a 3-clause), we say that it this gadget *selects* a literal vertex c_i^j, if c_i^j is a member of two edges in the clause gadget. Note that in both clause gadgets at least one of the literal vertices is selected. This is obvious for the 2-clause gadget. For the 3-clause gadget, if none of the literal vertices is selected in a valid covering of this gadget, then in the three hyperedges in Figure 3(c), no picked edge involves c_i^1, c_i^2 or c_i^3. But this creates a cycle, a contradiction. Now, all $3n$ literal vertices c_i^j from the set of clause gadgets for C will appear in the variable gadgets, and if a literal vertex c_i^j is selected by a clause gadget then it cannot be a member of

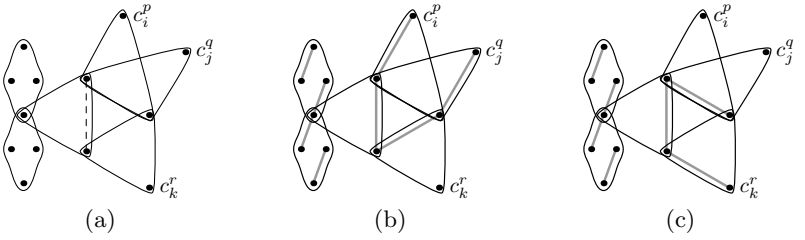


Fig. 4. (a) The variable gadget for variable with positive occurrences c_i^p and c_j^q and negated occurrence c_k^r in the clauses. The dashed edge is always picked in any valid covering. (b) Grey edges are picked when this variable is set to *false* in a satisfying assignment of ϕ. (c) Grey edges are picked when the variable is set to *true*.

any edge in the variable gadget, otherwise c_i^j has degree 3 or more. The variable gadget for each $x \in X$ will use this property to ensure that literal vertices x and $\neg x$ are not selected at the same time.

Figure 4(a) depicts the variable gadget for variable $x \in X$ with the two positive occurrences c_i^p and c_j^q, and one negated occurrence c_k^r of this variable x in the clauses. Note that if both a positive and the negated literal vertices of x are selected by a clause gadget in a valid covering of H_ϕ, then it forces a cycle in the variable gadget of x, a contradiction. It follows that if H_ϕ has a valid covering then ϕ is satisfiable.

Conversely, if ϕ has a satisfying assignment τ, let us pick one literal for each clause which makes it satisfied in τ and build the covering of H_ϕ as follows. In each clause gadget, (i) in each hyperedge of this clause gadget that contains a literal vertex, pick an edge containing the literal vertex if this literal was selected for this corresponding clause, and (ii) for each diamond, choose any of the 3 valid coverings of this diamond that consist of 2 parallel edges. In the variable gadgets, pick the edges as depicted in Figure 4(b) if the variable has value *false* in τ and otherwise, pick the edges as depicted in Figure 4(c). By selecting edges in this fashion, every hyperedge of H_ϕ is covered by an edge, and each literal vertex is adjacent to at most two edges in the covering, one of them lying in the diamond. Hence, there is no vertex of degree 3 and no cycles in this covering, i.e., this covering is valid.

Since the number of hyperedges used in the construction is at most $15m+12n$, i.e., linear in the size of ϕ, this construction can be done in polynomial-time, and hence, the 3-UHC-P Problem is NP-hard. □

The following lemma shows the correspondence between the $(3, 2, \infty)$-C1P Problem and the 3-UHC-P Problem. A 3-uniform hypergraph $H = (V, E)$ can be represented by a binary matrix B_H with $|V|$ columns and $|E|$ rows, where for each hyperedge $h \in E$, we add a row with 1's in the columns corresponding to vertices in h and 0's everywhere else. Obviously, there is one-to-one correspondence between 3-uniform hypergraphs and such matrices.

Lemma 1. *A 3-uniform hypergraph $H = (V, E)$ can be covered by disjoint paths if and only if matrix B_H has the $(3, 2, \infty)$-C1P.*

Proof. Assume first that H has a covering G that consists of disjoint paths. Then there is a Hamiltonian path P on V containing all edges of G. This path defines an order on the vertices in V. Consider the ordering of the columns of matrix B_H based on this order (V is the set of columns of B_H). It is easy to see that this ordering is $(3, 2, \infty)$-consecutive.

Conversely, assume that matrix B_H is $(3, 2, \infty)$-consecutive w.r.t. order $\pi = v_{i_1}, \ldots, v_{i_n}$. Consider the following covering G of H: for every hyperedge pick the edge between two adjacent columns/vertices in π. Note that every picked edge is $\{v_{i_j}, v_{i_{j+1}}\}$ for some j. Hence, G consists of disjoint paths. □

By Theorem 3 and Lemma 1 it follows that the $(3, 2, \infty)$-C1P Problem is NP-complete.

Theorem 4. *The $(3, 2, \infty)$-C1P Problem is NP-complete.*

5 Conclusion

In this work, we have studied the weakest formulation of the C1P Problem with gaps: in the $(d, d-1, \infty)$-C1P Problem it is required that only two of the d `1`'s in each row are adjacent, while the other `1`'s can end up arbitrarily far away from this pair. It is thus surprising that this problem is still NP-complete for any $d \geq 3$ as we have shown in Sections 3 and 4. This closes the case of the complexity of the (d, k, δ)-C1P Problem, with the exception of the $(\infty, 2, 1)$-C1P case, or just the $(2, 1)$-C1P case [10], which remains open.

There are three directions we would like to follow in the future work: (i) Is it possible to find a nice characterization of non-(d, k, δ)-C1P matrices in terms of forbidden structures, such as Tucker submatrices [14], especially, for small values of d? It has recently been shown that such a characterization could be used in the design of algorithms related to the C1P [5,2]. (ii) Can we generalize the hypergraph covering problem to d-uniform hypergraphs and selection of p edges from each hyperedge? If finding such coverings under the same condition (cover is a collection of paths) remains NP-complete, it could provide a stronger and more elegant proof of the results in this paper (in particular, it could show that the (d, k, ∞)-C1P Problem is NP-complete for d-uniform matrices). We have some preliminary results in this direction. Considering other conditions on the covering could also give rise to a set of new interesting problems. (iii) Assuming that k is close to d, for each row there are many orders of columns which make this row (d, k, ∞)-consecutive. Hence, for a small number of rows, random instances of matrices have the (d, k, ∞)-C1P almost always. We would like to investigate the ratios between the number of rows and columns for which this is the case.

Acknowledgements

We would like to thank Cedric Chauve for helpful remarks on this paper.

References

1. Alizadeh, F., Karp, R., Weisser, D., Zweig, G.: Physical mapping of chromosomes using unique probes. J. Comput. Biol. 2(2), 159–184 (1995)
2. Chauve, C., Haus, U.W., Stephen, T., You, V.P.: Minimal conflicting sets for the consecutive-ones property in ancestral genome reconstruction. In: Ciccarelli, F.D., Miklós, I. (eds.) RECOMB-CG 2009. LNCS (LNBI), vol. 5817, pp. 48–58. Springer, Heidelberg (2009)
3. Chauve, C.: Maňuch, J., Patterson, M.: On the gapped consecutive-ones property. In: Proc. of European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB). ENDM, vol. 34, pp. 121–125 (2009)
4. Chauve, C., Tannier, E.: A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. PLoS Comput. Biol. 4, e1000234 (2008)
5. Dom, M.: Recognition, generation, and application of binary matrices with the consecutive-ones property. Ph.D. thesis, Institut für Informatik, Friedrich-Schiller-Universität, Jena (2008)
6. Goldberg, P., Golumbic, M., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. J. Comput. Biol. 2(1), 139–152 (1995)
7. Gupta, A., Maňuch, J., Stacho, L., Zhao, X.: Algorithm for haplotype inferring via galled-tree networks with simple galls. In: Măndoiu, I.I., Zelikovsky, A. (eds.) ISBRA 2007. LNCS (LNBI), vol. 4463, pp. 121–132. Springer, Heidelberg (2007)
8. Gupta, A., Maňuch, J., Stacho, L., Zhao, X.: Haplotype inferring via galled-tree networks is NP-complete. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 287–298. Springer, Heidelberg (2008)
9. Gupta, A., Maňuch, J., Stacho, L., Zhao, X.: Haplotype inferring via galled-tree networks using a hypergraph covering problem for special genotype matrices. Discr. Appl. Math. 157(10), 2310–2324 (2009)
10. Maňuch, J., Patterson, M., Chauve, C.: Hardness results for the gapped C1P problem (unpublished manuscript)
11. McConnell, R.M.: A certifying algorithm for the consecutive-ones property. In: Proc. of Symposium on Discrete Algorithms (SODA), pp. 761–770. SIAM, Philadelphia (2004)
12. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
13. Saxe, J.B.: Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. SIAM J. on Alg. and Discr. Meth. 1(4), 363–369 (1980)
14. Tucker, A.C.: A structure theorem for the consecutive 1's property. J. of Comb. Theory, Series B 12, 153–162 (1972)

An Approximation Algorithm for Computing a Parsimonious First Speciation in the Gene Duplication Model

Aïda Ouangraoua[1,2,3], Krister M. Swenson[2,4], and Cedric Chauve[1,2]

[1] Department of Mathematics, SFU, Burnaby (BC), Canada
cchauve@sfu.ca
[2] Lacim, Université du Québec à Montréal, Montréal (QC), Canada
[3] INRIA LNE, LIFL, Université Lille 1, France
aida.ouangraoua@inria.fr
[4] Department of Mathematics and Statistics, University of Ottawa, Ottawa (ON), Canada
thekswenson@gmail.com

Abstract. We consider the following problem: given a forest of gene family trees on a set of genomes, find a first speciation which splits these genomes into two subsets and minimizes the number of gene duplications that happened before this speciation. We call this problem the Minimum Duplication Bipartition Problem. Using a generalization of the Minimum Edge-Cut Problem, known as Submodular Function Minimization, we propose a polynomial time and space 2-approximation algorithm for the Minimum Duplication Bipartition Problem. We illustrate the potential of this algorithm on both synthetic and real data.

1 Introduction

Gene duplication is a fundamental evolutionary mechanism for important groups of eukaryotes such as vertebrates [3], insects [15], plants [19] or fungi [23]. Gene duplications, together with gene losses, result in *gene families*, which can contain several copies of a same gene in a given genome. Recent advances in methods for reconstructing phylogenetic trees for individual gene families, have resulted in large sets of accurate *gene trees* for eukaryote species, such as TreeFam [24]. Phylogenomics aims at reconstructing the evolution of *species (genomes)* by inferring a species tree for the set of genomes from a set of gene trees. The *Minimum Duplication Problem* (MD Problem), also known as the Gene Duplication Problem, is to infer, from a set of gene trees, a species tree that induces an evolutionary history with a minimum number of gene duplications. This problem is NP-hard, and is neither fixed-parameter tractable (FPT) nor approximable with a constant ratio [2,12]. Recent advances in local search heuristics proved to be useful [1] and have been applied on several eukaryotic datasets with interesting results (see [19,25]), but with no optimality guarantee.

Recently, Chauve and El-Mabrouk [7,20] described a formal link between the Minimum Duplication Problem and a problem of supertrees, where, given a set of uniquely leaf-labeled gene trees (there is at most one copy of each gene in

E. Tannier (Ed.): RECOMB-CG 2010, LNBI 6398, pp. 290–301, 2010.

each genome), the goal is to reconstruct a species tree that disagrees with the minimum number of gene trees [6]. This problem — a version of the MD Problem restricted to uniquely leaf-labeled trees — is NP-hard too, even in the simple case where each gene tree is a rooted triplet [4]. For supertree problems, greedy heuristics based on the principle of computing successive optimal speciations, modeled as edge-cuts in a graph whose vertices are the considered species, are now standard [18,21]. In such heuristics, each Minimum Edge-Cut splits the set of considered species into two subsets that correspond to a speciation that results in two distinct lineages. Computing an optimal first speciation (a first speciation that disagrees with the least number of rooted triplets) is tractable, as the Minimum Edge-Cut problem is tractable. A complete species tree can then be obtained from a series of locally optimal speciations.

In the present work we consider the Minimum Duplication Bipartition (MDB) Problem: given a set of gene trees (where a gene occurs any number of times), find a bipartition of the considered genes (corresponding to a speciation) that minimizes the number of duplications that happened before this speciation. A first motivation for this problem is that it leads, as for supertree problems, to a natural greedy heuristics to reconstruct a species tree from a set of gene trees. Also, solving the Minimum Duplication Bipartition Problem can provide valuable information on the combinatorial nature of early speciations for large eukaryotic datasets with respect to gene duplications. Our main result is a polynomial time 2-approximation algorithm for the Minimum Duplication Bipartition Problem that generalizes the Minimum Edge-Cut approach used in supertrees and relies on Submodular Function Minimization [10]. Although our focus here is mostly theoretical, and explores the combinatorial structure of parsimonious first speciations, we also provide experimental results, on small datasets, which illustrate the potential of our approach.

We first define, in Section 2, gene trees, species trees, duplications, and the optimization problems considered in this paper as well as our motivation to introduce the MDB Problem. In Section 3 we describe our 2-approximation algorithm[1]. Our experimental results are described in Section 4.

2 Preliminaries: Objects, Problems, Background

In this section and the sequel, $\mathcal{G} = \{1, 2, \ldots, k\}$ is a set of integers representing k different species (genomes).

Gene and species trees, bipartitions. A *species tree* on $\mathcal{G}$ is a tree with exactly k leaves, where each $i \in \mathcal{G}$ is the label of a single leaf. A tree is binary if every internal vertex has exactly two children. A *gene tree* on $\mathcal{G}$ is a binary tree where each leaf is labeled with an integer from $\mathcal{G}$. A gene tree is a formal representation of a phylogenetic tree of a gene family, where each leaf labeled i represents a gene which is a member of the gene family located on genome i. A gene tree is *uniquely leaf-labeled* if no two leaves have the same label. A gene tree is a *rooted triplet* if it has exactly three leaves.

[1] Missing proofs are available at
`http://www.cecm.sfu.ca/~cchauve/SUPP/RECOMBCG10`.

Given a tree T and a vertex x of T whose leaves are labeled by integers from $\mathcal{G}$, we denote by $L(x)$ (resp. $L(T)$) the subset of $\mathcal{G}$ defined by the labels of the leaves of the subtree of T rooted at x (resp. the leaves of T). If x is not a leaf, we denote by x_ℓ and x_r the two children of x.

A *bipartition* B on a set S is a partition of S into two subsets. We represent a bipartition by a, possibly non-binary, species tree on S containing exactly three internal vertices — the root v and its two children v_ℓ and v_r — such that $L(v_\ell) \cap L(v_r) = \emptyset$.

Reconciliation between Gene Trees and Species Trees. The *Lowest Common Ancestor Mapping (LCA mapping)* is central in the problem of reconciling a gene tree and a species tree. Given a gene tree T and a species tree S on $\mathcal{G}$, the LCA mapping M maps vertices of T to vertices of S as follows: for a vertex x of T, $M(x) = v$ is the unique vertex of S such that $L(x) \subseteq L(v)$ and v is either a leaf of S or $L(x)$ is not included in the leaf set of any child of v. In other words, v is the deepest among all possible. A vertex x of T is then a *duplication with respect to* S if $M(x) = M(x_r)$ and/or $M(x) = M(x_\ell)$; otherwise, x is called a *speciation with respect to* S (see Figure 1). The same definitions apply to a forest F of gene trees on $\mathcal{G}$. The *duplication cost* of F given S denoted by $d(F, S)$ is the number of vertices of F that are duplications with respect to S. Note that the definitions of duplication and speciation apply to a species tree that is a bipartition on the set $\mathcal{G}$, as these definitions do not depend on the species tree being binary.

If $L(x_\ell) \cap L(x_r) \neq \emptyset$, then x is a duplication vertex with respect to any species tree S on $\mathcal{G}$. Such a vertex is called an *apparent duplication*. Vertices of F that are not apparent duplication are called *non-apparent duplication*.

Inferring parsimonious species trees and speciations. It is well known that $d(F, S)$ is the minimum number of gene duplication events required in any evolutionary scenario that resulted in F (see [7,11] and references there), which leads to the following optimization problem called the Minimum Duplication Problem (MD Problem): given a gene tree forest F find a species tree S such that $d(F, S)$ is minimum.

The MD Problem is NP-hard [16], even in the case where every gene tree is a uniquely leaf labeled and rooted triplet [4], in which case it is in fact equivalent to

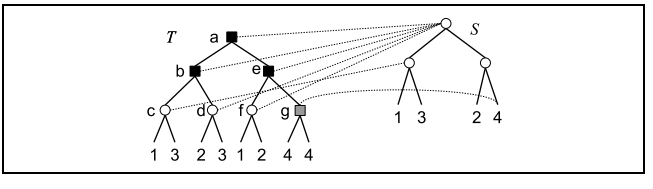


Fig. 1. A gene tree T and a species tree S on a set of genome $\mathcal{G} = \{1, 2, 3, 4\}$. The LCA mapping from vertices of T to vertices of S is indicated by dashed lines linking vertices. The vertices of T that are duplications with respect to S are represented by square vertices; the black colored square vertices correspond to pre-duplications while the grey colored square vertices are the duplications that are not pre-duplications. Here, the first speciation of S is the bipartition with root v such that $L(v_l) = \{1, 3\}$ and $L(v_r) = \{2, 4\}$. a, b and g are apparent duplications.

a supertree problem called the Minimum Rooted Triplet Inconsistency (MRTI) Problem. This link with supertrees is important as recent hardness results on the MRTI Problem imply that first, the MD Problem is W[2]-hard, and thus not FPT [2,12] — contrary to what was believed ten years ago [22] — but also that it cannot be approximated within a constant ratio unless P=NP [6]. Hence, for solving the MD Problem, one has to rely on exponential time algorithms such as [13] or local-search methods with no optimality guarantee [1,25].

In [7], it was shown that the MD Problem is in fact a slight variant of a supertree problem (see also [20] that explores the link between gene duplications and supertrees). Greedy heuristics for hard supertree problems based on computing successive speciations events have proved to be effective [21], and in [7], an application of such a heuristic showed promising results on synthetic data. This motivates the introduction of the main problem we study in this paper.

Given a gene tree forest F on $\mathcal{G}$ and a bipartition B on $\mathcal{G}$ with root v, a duplication x of F with respect to B is said to *precede the first speciation with respect to* B if $M(x) = v$. Such vertices are called *pre-duplications* (e.g. a, b and e in Figure 1). We denote by $d_1(F, B)$, the number of pre-duplications of F with respect to B.

Minimum Duplication Bipartition Problem (MDB Problem):
Input: A gene tree forest F on $\mathcal{G}$;
Output: A bipartition B on $\mathcal{G}$ such that $d_1(F, B)$ is minimum.

Before discussing previous works, we state an obvious, but very useful, property related to duplication vertices of a forest of gene trees F on $\mathcal{G}$.

Property 1. Let x be a vertex of F. Given a bipartition B on $\mathcal{G}$ with root v, x is a pre-duplication with respect to B if and only if there exists a pair $\{s, t\} \subseteq L(v_\ell) \times L(v_r)$ such that $\{s, t\} \subseteq L(x_\ell)$ or $\{s, t\} \subseteq L(x_r)$.

As far as we know, the MDB problem was introduced in [22], where an exponential time algorithm was proposed. It was also shown in [7], although not formally stated, that if there exists a bipartition such that all pre-duplications are apparent duplications, then such a bipartition can be computed in polynomial time and space. The hardness of the MDB Problem is still open, but preliminary results showing the hardness of a slight variant using quadruplets as input gene trees (G. Blin and S. Vialette, personal communication), suggest it may be NP-complete. In [8], it was shown that if F contains a single gene tree, the MDB Problem is 3-approximable. However, in the more general case of a forest F with t gene trees, the approximation ratio is not constant: if a parsimonious first speciation implies d duplications, then the algorithm described in [8] computes a first speciation that can imply up to $2d+t$ duplications. In the present work, we show that the MDB Problem can be approximated with a constant ratio of 2 in polynomial time and space.

Related optimization problems. Given a connected graph $G = (V, E)$, an *edge-cut* of G is an edge set $E' \subseteq E$ whose removal disconnects the graph G. A bipartition B with root v on the set of vertices of G induces a unique edge-cut of G, denoted by $E_G(B)$, composed of the edges $(s, t) \in E$ such that $s \in L(v_\ell)$ and $t \in L(v_r)$. So $E_G(B) = \{(s, t) \in E \mid s \in L(v_\ell),\ t \in L(v_r)\}$. Hence, a subset X of V induces a bipartition on V with root v such that $L(v_\ell) = X$ and $L(v_r) = V - X$. We

denote this bipartition by $B_V(X)$ and the edge-cut of G induced by $B_V(X)$ is denoted by $E_G(X)$ ($E_G(X) = E_G(B_V(X))$).

The *Minimum Edge-Cut (MEC) Problem* asks for a bipartition on the vertices of G inducing an edge-cut of G of minimum cardinality. If the edges of G are labeled on a given set Σ of labels, given an edge-cut E' of G, the *label-set* of E' denoted by label(E') is the subset of Σ composed of the labels of the edges in E'. The following cut problem is a natural generalization of the MEC Problem and is essential in our algorithm:

Minimum Labeled-Edge-Cut (MLEC) Problem:
Input: A connected edge-labeled graph $G = (V, E)$;
Output: A bipartition B on V such that the cardinality of label($E_G(B)$) is minimum.

A *set function* is a function $f : 2^V \rightarrow \mathbb{R}$ defined from the set of the $2^{|V|}$ subsets of a finite set V onto the real numbers $\mathbb{R}$. The set V is called the *ground set* of f. The Set Function Minimization Problem asks to find a non-empty subset X of V such that $f(X)$ is minimum. A *submodular function* is a set function f with ground set V such that for any subsets A and B of V, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. Several combinatorial optimization problems have been linked to submodular functions [10], in particular the MEC Problem. Given a submodular function f, the following optimization problem, which is tractable [14], is a special case of the Set Function Minimization Problem:

Submodular Function Minimization (SFM) Problem:
Input: A submodular function $f : 2^V \rightarrow \mathbb{R}$ with ground set V;
Output: A non-empty subset X of V such that $f(X)$ is minimum.

A *hypergraph* is a pair (V, E) where V is a set of vertices and E is a set of non-empty subsets of V called *hyperedges*. Given a hypergraph $G = (V, E)$, a bipartition B on V with root v induces a *hyperedge-cut* of G defined by the following subset of E:
$E_G(B) = \{e \in E \mid e \cap L(v_\ell) \neq \emptyset \text{ and } e \cap L(v_r) \neq \emptyset\}$.

Minimum Hypergraph Cut (MHC) Problem:
Input: A hypergraph $G = (V, E)$;
Output: A bipartition B on V such that the cardinality of $E_G(B)$ is minimum.

3 A 2-Approximation Algorithm for the MDB Problem

3.1 A Set Function Minimization Problem

In the following, given a gene tree forest, we label arbitrarily its internal vertices with a set Σ of labels in such a way that no two internal vertices have the same label.

Given a gene tree forest F, we define the *edge-labeled graph* $H(F) = (V, E)$ associated to F as follows (see Figure 2): $V = L(F)$ and there is an edge labeled with $a \in \Sigma$ between two vertices s and t of $H(F)$ if and only if there exists an internal vertex x of F labeled with a such that $\{s, t\} \subseteq L(x_\ell)$ or $\{s, t\} \subseteq L(x_r)$.

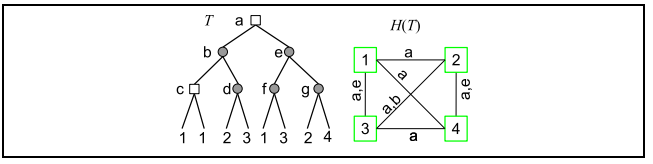


Fig. 2. A gene tree T on the set of genomes $\mathcal{G} = \{1, 2, 3, 4\}$ and the corresponding edge-labeled graph $H(T)$. Apparent duplication vertices of T appear as square vertices.

Lemma 1. *Let F be a gene tree forest on $\mathcal{G}$. If B is a bipartition on $L(F)$, then the set of labels of the pre-duplications of F with respect to B is exactly the set* $\text{label}(E_{H(F)}(B))$.

The MLEC Problem can be reduced to a Set Function Minimization Problem as follows: given an edge-labeled graph $G = (V, E)$, we define the *cut-set function* $f_G : 2^V \to \mathbb{R}$ as a function from the set of the subsets of V onto $\mathbb{R}$ such that, for any subset X of V, $f_G(X)$ is the cardinality of the set of labels $\text{label}(E_G(X))$. It is easy to see that solving the MLEC Problem on G can be achieved by minimizing f_G. In the following, given a gene tree forest F, we simply denote by f_G the cut-set function induced by an edge-labeled graph $G(F)$ associated to F.

Unfortunately, the cut-set function f_G associated to an edge-labeled graph G is not always a submodular function. For example (Figure 3), consider a single gene tree T with four leaves $\{1, 2, 3, 4\}$ and three internal vertices a, b and c whose sets of children are respectively $\{b, c\}$, $\{1, 3\}$ and $\{2, 4\}$. The edge-labeled graph $H(T)$ associated to T has four vertices $\{1, 2, 3, 4\}$ and only two edges $(1, 3)$ and $(2, 4)$ labeled with a. If we consider the subsets $A = \{1, 4\}$ and $B = \{2, 4\}$ of the set of $\{1, 2, 3, 4\}$, we see that $f_{H(T)}(A) = 1$, $f_{H(T)}(B) = 0$, $f_{H(T)}(A \cup B) = 1$ and $f_{H(T)}(A \cap B) = 1$. Then, $f_{H(T)}(A) + f_{H(T)}(B) = 1 < f_{H(T)}(A \cup B) + f_{H(T)}(A \cap B) = 2$, and $f_{H(T)}$ is not a submodular function which proves the following property:

Property 2. There exist gene trees forest F such that the cut-set function f_H, where $H = H(F)$, is not a submodular function.

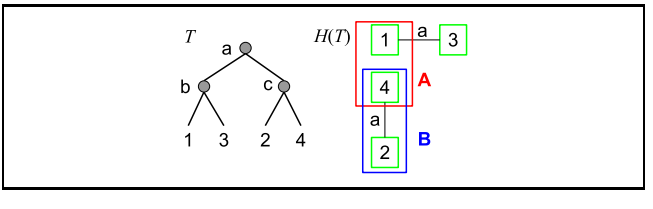


Fig. 3. Illustration of Property 2: a gene tree T (left) and the corresponding graph $H(T)$ (right), and two vertex sets A and B that contradict the submodularity of the cut-set function f_H

3.2 Submodular Function Minimization

In this section, we prove our main result.

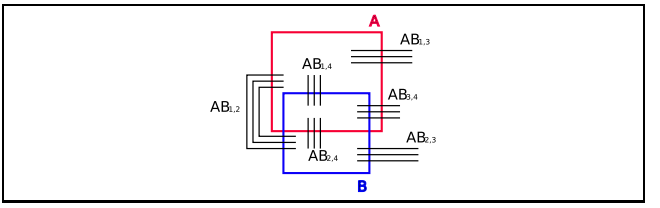


Fig. 4. Illustrations of the definition of the six sets of labels $\mathrm{AB}_{i,j}(G)$ $(1 \leq i < j \leq 4)$ associated to two subsets A and B of the set $L(F)$ of leaves of a gene tree forest F and an edge-labeled graph G whose set of vertices is $L(F)$: the A and B are represented by two squares and the solid black lines correspond to edges of G whose labels belong to one of the six sets according to membership of their extremities to the sets AB_i $(1 \leq i \leq 4)$

Theorem 1. *Let F be a gene tree forest with n vertices, on a set $\mathcal{G}$ of k genomes. If a most parsimonious bipartition B^* on $L(F)$ has cost $d_1(F, B^*) = d$, then it is possible to compute in time $O(kn)$ a bipartition B s.t. $d(F, B) \leq 2d$.*

The approximation results from transforming the graph $H(F)$ associated to a gene tree forest F in order to obtain a new edge-labeled graph $J(F)$ such that the set function f_J is a submodular function.

Characterization of non-submodularity. Given two subsets A and B of $L(F)$, we define the following four subsets of $L(F)$: $\mathrm{AB}_1 = A - B$, $\mathrm{AB}_2 = B - A$, $\mathrm{AB}_3 = L(F) - (A \cup B)$ and $\mathrm{AB}_4 = A \cap B$. Note that the intersection of any two of these subsets is empty, and the union of all of them is $L(F)$.

Given an edge-labeled graph G whose set of vertices is $L(F)$, we then define six sets of labels $\mathrm{AB}_{1,2}(G)$, $\mathrm{AB}_{1,3}(G)$ $\mathrm{AB}_{1,4}(G)$, $\mathrm{AB}_{2,3}(G)$, $\mathrm{AB}_{2,4}(G)$ and $\mathrm{AB}_{3,4}(G)$ as follows: given two integers i and j such that $1 \leq i < j \leq 4$, the set $\mathrm{AB}_{i,j}(G)$ is the set of labels of edges (s, t) in G such that $s \in \mathrm{AB}_i$ and $t \in \mathrm{AB}_j$ (see Figure 4).

In the following, given a gene tree forest F, two subsets A and B of $L(F)$, and the edge-labeled graph $H(F)$ associated to F, we simply denote any set $\mathrm{AB}_{i,j}(H(F))$ by $\mathrm{AB}_{i,j}$.

Lemma 2. *Let F be a gene tree forest and f_H be the cut-set function associated to $H(F)$. If f_H is not a submodular function then there exist two subsets A and B of $L(F)$ and an internal vertex x of F labeled with $a \in \Sigma$ that is a non-apparent duplication, such that at least one of the two following configurations holds:*

- *(1) $a \in \mathrm{AB}_{1,3} \cap \mathrm{AB}_{2,4}$ and $a \notin \mathrm{AB}_{1,2} \cup \mathrm{AB}_{1,4} \cup \mathrm{AB}_{2,3} \cup \mathrm{AB}_{3,4}$ or*
- *(2) $a \in \mathrm{AB}_{2,3} \cap \mathrm{AB}_{1,4}$ and $a \notin \mathrm{AB}_{1,2} \cup \mathrm{AB}_{1,3} \cup \mathrm{AB}_{2,4} \cup \mathrm{AB}_{3,4}$.*

Modification of the edge-labeled graph $H(F)$. We now present a modification of $H(F)$ that leads to our 2-approximation algorithm for the MDB Problem. The goal is to modify $H(F)$ into a new edge-labeled graph $J(F)$ such that the two configurations of Lemma 2 never hold, leading to a cut-set function that is submodular. The transformation is as follows: for each vertex x of F (say labeled with $a \in \Sigma$) that is not an apparent duplication, reassign to edges (s, t) of $H(F)$

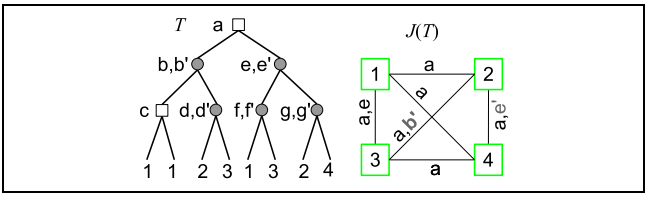


Fig. 5. A gene tree T on the set of genomes $\mathcal{G} = \{1, 2, 3, 4\}$ and the corresponding modified edge-labeled graph $J(F)$, where modified labels are displayed in grey color. Apparent duplication vertices of T appear as square vertices.

labeled with a such that $\{s, t\} \subseteq L(x_r)$, a new label that is different from a (see Figure 5).

More precisely, given a gene tree forest F we now label each non apparent duplication vertex of F with an ordered pair of labels in Σ, and define the *edge-labeled graph* $J(F) = (V, E)$ as follows (see Figure 5): the set V of vertices of $J(F)$ is $L(F)$, and there is an edge (s, t) labeled $a \in \Sigma$ in $J(F)$ if and only if there exists an internal vertex x of F such that:

- either $\{s, t\} \subseteq L(x_\ell)$ or $\{s, t\} \subseteq L(x_r)$ and x is an apparent duplication labeled with a,
- or $\{s, t\} \subseteq L(x_\ell)$ and x is a non-apparent duplication labeled with (a, a'),
- or $\{s, t\} \subseteq L(x_r)$ and x is a non-apparent duplication labeled with (a', a).

Lemma 3. *Given a gene tree forest F, the cut-set function f_J associated to $J(F)$ is a submodular function.*

Proof of Theorem 1. For any bipartition B on $L(F)$, the cardinality of label $(E_H(B))$ (resp. label$(E_J(B))$) is denoted by $d_H(B)$ (resp. $d_J(B)$).

We first prove that, for any bipartition B on $L(F)$ with root v, $d_H(B) \leq d_J(B) \leq 2 * d_H(B)$. It is relatively straightforward. First, note that $E_H(B) = E_J(B)$ since $J(F)$ differs from $H(F)$ only in the fact that the labels of some edges have been changed. Next, a label $a \in$ label$(E_H(B))$ corresponds to at least one but at most two labels in label$(E_J(B))$: if a is the label of an apparent duplication, then we have $a \in$ label$(E_H(B)) \Leftrightarrow a \in$ label$(E_J(B))$; if a is the label of a non-apparent duplication x, then the vertex x has two labels — a and a' — such that one or both belong to label$(E_J(B))$. This proves that $d_H(B) \leq d_J(B) \leq 2 * d_H(B)$.

Now, let B' be a bipartition on $L(F)$ inducing an optimal labeled edge-cut of $H(F)$ (i.e $d_H(B')$ is minimum). For any bipartition B on $L(F)$, if $d_H(B) > 2 * d_H(B')$ then B cannot induce an optimal labeled edge-cut of $J(F)$: indeed, if $d_H(B) > 2 * d_H(B')$, as $d_H(B) \leq d_J(B)$ and $d_J(B') \leq 2 * d_H(B')$, we have $d_J(B) > 2 * d_H(B') \geq d_J(B')$. Hence, for any bipartition B on $L(F)$ that is optimal for $J(F)$, we have $d_H(B) \leq 2 * d_H(B')$. This completes the proof that computing an optimal labeled edge-cut for $J(F)$ achieves a ratio 2 approximation for the MDB Problem.

The complexity stated in Theorem 1 follows. The $O(kn)$ time complexity is derived from the reduction of the minimization of the function f_J to the MHC

Problem as follows. Given a graph $G = (V, E)$ with edges labeled on a set Σ of labels, we define the hypergraph $G_h = (V_h, E_h)$ such that $V_h = V$ and, for each label $a \in \Sigma$, G_h contains an hyperedge composed of all vertices s in V that belong to an edge labeled by a. Given a gene tree forest F and a bipartition B on $L(F)$, if we consider the graph $J = J(F)$, it is obvious that the cardinality of label$(E_J(B))$ is equal to the cardinality of $E_{J_h}(B)$. Hence, the minimization of f_J can be reduced to the MHC Problem on J_h. The time complexity given in the theorem then follows from the algorithm described in [17] solving the MHC Problem on a hypergraph G in time and space $O(kn)$ where k (resp. n) is the number of vertices (resp. hyperedges) of G.

Additional remarks. If there exists at least one parsimonious first speciation B' such that all the corresponding pre-duplications are apparent duplications in F, then our algorithm computes a parsimonious bipartition, and not an approximation. Indeed, in such a case, B' induces an optimal cut for both $H(F)$ and $J(F)$. Note however that this does not imply that the cut-set function for $H(F)$ is submodular.

Conversely, the approximation ratio of 2 is tight, as illustrated in Figure 6. This example is easily expanded to any size by extending the top and bottom rows of the graph $J(F)$, adding pairs of labeled edges between the rows and the suitable number of unlabeled edges between the vertices of the top row and between the vertices of the bottom row.

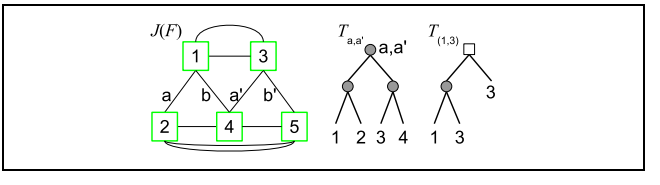


Fig. 6. The forest F is built from trees corresponding to the labeled and unlabeled edges in $J(F)$. For a pair of edges in $J(F)$ labeled with two labels a, a', say (p, q) and (u, v), we add to F the tree $((p, q), (u, v))$. For an unlabeled edge (u, v), we add to F the tree $((u, v), v)$. The tree $T_{a,a'}$ (resp. $T_{(1,3)}$) corresponds to the pair of edges labeled with a, a' (resp. the unlabeled edge $(1, 3)$).

4 Experimental Results

We performed three different experiments[2]. First, on several datasets of simulated gene families on 12 species (genomes) we studied the ability of the greedy approach — that infers a species tree by computing successive parsimonious speciations — to recover the exact species tree, using an exhaustive exploration of all possible speciations at each step (which was possible due to the fact we considered only 12 species). Next, on the same simulated datasets, we replaced the exhaustive exploration of all possible speciations at each step by our 2-approximation algorithm for computing a parsimonious speciation. Last, we performed the same experiment, using only the approximation algorithm on a large dataset of gene families on 23 fungal genomes.

[2] Data and results available at `http://www.cecm.sfu.ca/~cchauve/SUPP/RECOMBCG10`

Datasets. We uploaded from the Fungal Orthogroup Repository[3] the 6808 fungal gene trees containing genes belonging to at least three different species, among 23 fungal species.

We also analyzed the four synthetic datasets that were studied in [7]; each dataset contains 100 gene trees. Each gene tree was generated from a single ancestral gene with duplication (birth) and loss (death) rates computed using the software CAFE [9] from real Drosophilia gene families [15].

To balance the fact that each gene tree originates from a single ancestral gene (and then has no duplication before the first speciation), and to consider datasets including gene families generated with different duplication/loss rates, we created duplications that happened before the first speciation by clustering the 400 gene trees of the four datasets into 100 clusters of random size, and for every such cluster of a given size, say k, we generated a random binary tree with k leaves and replaced each leaf of this tree by a gene tree of the cluster, which amounts to creating around 4 duplications that precede the first speciation. We repeated this experiment ten times, generating ten different datasets.

Results. On each of the simulated datasets, we first observed that the greedy heuristic that computes successive parsimonious speciations using an exhaustive exploration lead to the exact species tree, i.e. the one that had been used to generate the synthetic gene trees. Despite the relatively modest size of our synthetic datasets (12 species), it illustrates the potential of this greedy heuristic in a phylogenomics context, especially as the heuristic described in [7] inferred a slightly incorrect species tree for the two datasets with the highest duplication/loss rates. Moreover, we observed that our approximation algorithm provided the exact species tree every time. We believe this result can be explained by the fact that most duplications that occurred during the generation of the gene trees are apparent duplications.

Due to the large number of species in the real data set — 6808 fungal gene trees from 23 species — we applied only our approximation algorithm. We observed that the inferred species tree is the one widely accepted in yeasts phylogenomics[4]. We also noticed that a significant number of speciations satisfied the property that all the associated pre-duplications were apparent duplications. Such speciations are then parsimonious (see the discussion at the end of the previous section). Moreover, aside from one branch (leading to node 28 of the species tree, where 103 pre-duplications are non-apparent) all other branches are associated with very few such non-apparent pre-duplications, which suggests that they might be parsimonious. Providing the gene trees we analyzed are correct, this clearly suggests that traces of most duplications that occurred during yeast evolution are still visible today.

5 Conclusion

We showed that computing a parsimonious first speciation in the gene duplication model can be approximated in polynomial time with a ratio of 2. As far as

[3] Version 1.1, `http://www.broadinstitute.org/regev/orthogroups/`

[4] The species tree can be seen at `http://www.cecm.sfu.ca/~cchauve/SUPP/RECOMBCG10`.

we know this is the first time a constant approximation algorithm is proposed in relation with the problem of inferring species trees using gene duplications. This result was obtained by describing the problem in terms of edge-cuts in particular graphs, which can be computed in polynomial time through submodular function minimization. This algorithm is also a natural generalization of the classical minimum edge-cut algorithm that is used in supertree consistency problems, which is highlighted by its link with the Submodular Function Minimization Problem. Our preliminary experiments showed that both the approach of inferring a species tree by computing successive parsimonious speciations and our approximation algorithm for computing such speciations are promising, and we plan to apply them on larger datasets, like those that will soon be available from [5,26].

From a theoretical point of view, the hardness of the Minimum Duplication Bipartition Problem is still an open problem, but we conjecture the problem is NP-complete. It is interesting to note that, as for to the Gene Duplication Problem, when there is a parsimonious first speciation whose pre-duplications are all apparent duplications, it can be detected in polynomial time. Also when F contains only uniquely leaf-labeled rooted triplets, the graph $H(F)$ does not need to be augmented as every label appears only once, and computing a parsimonious first speciation can be done by computing a minimum edge-cut in $H(F)$. However, as we showed in the proof of Property 1, this tractability property no longer holds when quadruplets whose root is a non-apparent duplication are considered instead of triplets, as the cut-set function is no longer submodular. The role of non-apparent duplications, especially with respect to the non-submodularity of the cut-set function of $H(F)$, seems to be fundamental to the hardness of the problem, in particular to the understanding of which families of gene tree forests are tractable or fixed-parameter tractable.

Acknowledgments. Work supported by an NSERC Discovery grant to C.C. and a fellowship from the ANR (project ANR-06-BLAN-0045) for A.O. We thank Tamon Stephen, Mukul Bansal and Sylvain Guillemot for useful discussions.

References

1. Bansal, M.S., et al.: Heuristics for the gene-duplication problem: A $\Theta(n)$ speed-up for the local search. In: Speed, T., Huang, H. (eds.) RECOMB 2007. LNCS (LNBI), vol. 4453, pp. 238–252. Springer, Heidelberg (2007)
2. Bansal, M.S., Shamir, R.: A Note on the Fixed Parameter Tractability of the Gene-Duplication Problem (submitted, 2010)
3. Blomme, T., van de Peer, Y., et al.: The gain and loss of genes during 600 millions years of vertebrate evolution. Genome Biol. 7, R43 (2006)
4. Bryant, D.: Hunting for trees, building trees and comparing trees: theory and methods in phylogenetic analysis. Ph.D. thesis, Dept. of Math., Univ. of Canterbury, New Zealand (1997)
5. Burleig, J.G., et al.: Genome-scale phylogenetics: Inferring the plant tree of life from 18,896 discordant gene trees Systematic Biology (in press, 2010)
6. Byrka, J., Guillemot, S., Jansson, J.: New Results on Optimizing Rooted Triplets Consistency. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 484–495. Springer, Heidelberg (2008)

7. Chauve, C., El-Mabrouk, N.: New perspectives on gene family evolution: losses in reconciliation and a link with supertrees. In: Batzoglou, S. (ed.) RECOMB 2009. LNCS, vol. 5541, pp. 46–58. Springer, Heidelberg (2009)
8. Chauve, C., Ouangraoua, A.: A 3-approximation algorithm for computing a parsimonious first speciation in the gene duplication model. arXiv:0904.1645v2 (2009)
9. De Bie, T., et al.: CAFE: a computational tool for the study of gene family evolution. Bioinformatics 22, 1269–1271 (2006)
10. Fujishige, S.: Submodular Functions and Optimization, 2nd edn. Annals of Discrete Math., vol. 58. Elsevier, Amsterdam (2005)
11. Górecki, P., Tiuryn, J.: DLS-trees: a model of evolutionary scenarios. Theoret. Comput. Sci. 359, 378–399 (2006)
12. Guillemot, S.: Approches combinatoires pour le consensus d'arbres et de séquences. Ph.D. thesis, Univ. Montpellier II, France (2008)
13. Hallett, M.T., Lagergren, J.: New algorithms for the duplication-loss model. In: RECOMB 2000, pp. 138–146. ACM Press, New York (2000)
14. Iwata, S., Orlin, J.B.: Simple combinatorial algorithm for submodular function minimization. In: SODA 2009, pp. 1230–1237. SIAM, Philadelphia (2009)
15. Hahn, M.W., Han, M.V., Han, S.-G.: Gene family evolution across 12 Drosophilia genomes. PLoS Genet. 3, e197 (2007)
16. Ma, B., Li, M., Zhang, L.: From gene trees to species trees. SIAM J. Comput. 30, 729–752 (2000)
17. Mak, W.-K.: Faster Min-Cut Computation in Unweighted Hypergraphs/Circuit Netlists. In: VLSI Design, Automation and Test, 2005 (VLSI-TSA-DAT), pp. 67–70. IEEE, Los Alamitos (2005)
18. Page, R.D.M.: Modified mincut supertrees. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 537–551. Springer, Heidelberg (2002)
19. Sanderson, M.J., McMahon, M.M.: Inferring angiosperm phylogeny from EST data with widespread gene duplication. BMC Evol. Biol. 7, S3 (2007)
20. Scornavacca, C., Berry, V., Ranwez, V.: From gene trees to species trees through a supertree approach. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 702–714. Springer, Heidelberg (2009)
21. Semple, C., Steel, M.: A supertree method for rooted trees. Discrete Appl. Math. 105, 147–158 (2000)
22. Stege, U.: Gene Trees and Species Trees: The Gene-Duplication Problem in Fixed-Parameter Tractable. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 288–293. Springer, Heidelberg (1999)
23. Wapinski, I., et al.: Natural history and evolutionary principles of gene duplication in fungi. Nature 449, 54–61 (2007)
24. Li, H., et al.: Treefam: a curated database of phylogenetic trees of animal gene families. Nucleic Acids Res. 34, 572–580 (2006)
25. Wehe, A., et al.: DupTree: a program for large-scale phylogenetic analyses using gene tree parsimony. Bioinformatics 24, 1540–1541 (2008)
26. Zhou, X., Lin, Z., Ma, H.: Phylogenetic detection of numerous gene duplications shared by animals, fungi and plants Genome Biol., 11, R38 (2010)

Author Index