No. d'ordre: 017-2009 Année 2009

Thèse

présentée devant l'Université Claude Bernard - LYON I

pour l'obtention du
Diplôme de Doctorat
(*arrêté du 7 août 2006)*

Soutenue le 30 janvier 2009

par

Marília D. V. Braga

L'espace de solutions du tri par inversions et son utilisation dans l'analyse de réarrangements de génomes

Exploring the solution space of sorting by reversals when analyzing genome rearrangements

Composition du jury

Christian Gautier, *Président*

Eduardo Rocha, *Rapporteur*

Marie-France Sagot, *Directrice*

David Sankoff, *Rapporteur*

Eric Tannier, *Co-encadrant*

UMR CNRS 5558
Laboratoire de Biométrie et Biologie Evolutive
Université Claude Bernard LYON I - Bât G. Mendel
43, boulevard du 11 novembre 1918
69622 Villeurbanne

Mis en page avec la classe thloria.

Acknowledgments

I would like to express all my greateful thanks to Marie-France Sagot, Eric Tannier, Christian Gautier, Gabriel Marais, Fabrice Vavre, Guillaume Fertin, Eduardo Rocha, David Sankoff, Jens Stoye, João Meidanis, João Carlos Setubal, Nathalie Arbasetti, Isabelle Ravis, Gaelle Tworkowski, Lionel Humblot, Stéphane Delmotte, Bruno Spataro, my family and all my friends in Lyon and in Brazil.

This work was funded by the European Union Programme Alβan (scholarship no. E05D053131BR), the French projects ANR (REGLIS NT05-3_45205 and MIRI BLAN08-1_335497), INRIA ArcoIris (associated with the University of São Paulo, Brazil) and Rhône-Alpes Bioinformatics Center (PRABI).

To my parents, Antônio Fernando and Maria Inês.

Résumé

Le calcul de la distance d'inversion et celui d'une séquence optimale d'inversions pour transformer un génome dans un autre sont des outils algorithmiques très utiles pour l'analyse de scénarios d'évolution réels. Quand les duplications de gènes ne sont pas acceptées, il existe des algorithmes polynomiaux pour résoudre ces deux problèmes. Néanmoins, le nombre de séquences optimales différentes est très grand, et il faut alors considérer d'autres critères pour pouvoir réaliser une analyse plus précise.

Une stratégie possible est celle de chercher les séquences qui respectent certaines contraintes biologiques, comme par exemple les intervalles communs, qui sont les ensembles de gènes co-localisés dans les génomes analysés - une séquence d'inversions qui ne sépare pas les intervalles communs doit être plus réaliste qu'une séquence qui sépare. Une autre approche est celle de générer l'univers de toutes les séquences optimales, mais, comme cet ensemble peut être trop grand pour être interpreté, un modèle pour regrouper des sous-ensembles de séquences optimales dans des classes d'équivalence a été proposé, ce qui permet de réduire la taille de l'ensemble à traiter. Néanmoins, le problème de trouver les classes sans énumérer toutes les solutions optimales restait ouvert. Un de nos résultats les plus importants est, donc, l'algorithme qui donne une réponse à ce problème, c'est-à-dire, un algorithme qui génère une séquence optimale par classe d'équivalence et qui donne aussi le nombre de séquences par classe, sans énumérer toutes les séquences. Mais, bien que le nombre de classes soit beaucoup plus petit que le nombre de séquences, il peut être encore trop grand.

On propose alors l'utilisation de différentes contraintes biologiques, comme les intervalles communs (détectés initialement et progressivement), pour réduire le nombre de classes, et on montre comment utiliser ces méthodes pour analyser des cas réels d'évolution. En particulier, on analyse le scénario évolutif de la bactérie *Rickettsia* et des chromosomes sexuels X et Y chez l'être humain. Par rapport aux résultats des études précédentes, qui se sont basées sur une seule séquence optimale, on obtient une meilleure caractérisation de ces scénarios évolutifs.

Tous les algorithmes développés sont implémentés en java, integrés à baobabLuna, un logiciel qui contient des outils pour manipuler des génomes et des inversions. Le téléchargement et le tutoriel de baobabLuna sont disponibles en ligne.

Un autre résultat de notre travail est un modèle pour calculer une mesure de similarité entre deux génomes quand les duplications de gènes sont acceptées. On a montré que notre approche, qu'on appelle *repetition-free longest common subsequence* (rflcs), est un problème NP-difficile, comme les autres approches qui considèrent aussi des gènes dupliqués pour calculer une distance génomique.

Mots-clés: Évolution ; réarrangements de génomes ; algorithmes ; tri par inversions

Abstract

Calculating the reversal distance and finding one optimal sequence of reversals to transform a genome into another are useful algorithmic tools to analyse real evolutionary scenarios. When gene duplications are not allowed, there are polynomial algorithms to solve both problems. However, the number of different optimal sorting sequences is usually huge and some additional criteria should be taken in consideration in order to obtain a more accurate analysis.

One strategy is searching for sequences that respect some biological constraints, such as the common intervals, which are the list of clusters of co-localised genes between the considered genomes - an optimal sequence of reversals that does not break the common intervals may be more realistic than one that does break. Another approach is to explore the whole universe of sorting sequences, but, since this set may be too big to be directly interpreted, a model has been proposed to group the sorting sequences into classes of equivalence, reducing thus the size of the set to be handled. Nevertheless, the problem of finding an algorithm to direct generate the classes without enumerating all sequences was stated to be open. One of the most important results of our work is such an algorithm, and besides one representative, we are also able to give the number of sequences in each equivalence class. Although the number of classes is much smaller than the number of sorting sequences, it can also be too big.

We then propose the use of different biological constraints, such as the common intervals (initially and progressively detected), to reduce the universe of sequences and classes, and show how to apply these methods to analyze real cases in evolution. In particular, we analyze the evolution of the *Rickettsia* bacterium, and of the sexual chromosomes X and Y in human. We obtain a better characterization of the evolutionary scenarios of these genomes, with respect to the results of previous studies, that were based on a single sorting sequence.

All the algorithms developed in this work are implemented, integrated to BAOBABLUNA, a java framework to deal with genomes and reversals. Download and tutorial for BAOBABLUNA are available on-line.

Another result of our work is a model to compute a measure of similarity between genomes when duplications are allowed. Our approach, that is called repetition-free longest common subsequence (RFLCS), was proven to lead to an NP-hard problem, as well as other approaches to compute a genomic distance measure considering gene duplications.

Keywords: Evolution ; genome rearrangements ; algorithms ; sorting by reversals

Contents

Résumé **V**

Abstract **VI**

List of Figures **XI**

List of Tables **XIII**

List of Algorithms **XV**

Chapter 1 Introduction **1**

Chapter 2 Biological concepts **5**

2.1 Genome 5
2.2 Genes and protein code 6
2.3 Gene duplication 7
2.4 Genome organization 8
2.5 Recombination 8
2.6 DNA replication 10
2.7 Genome rearrangements 11

Chapter 3 Sorting by reversals **13**

3.1 Permutations, intervals and reversals 16
3.2 The breakpoint graph and the reversal distance 18
3.3 Safe and unsafe reversals 25
3.4 Sorting a signed permutation 26
3.5 The symmetry of sorting by reversals 27
3.6 Component-specific reversals 27

Chapter 4 Traces of sorting sequences of reversals **29**

4.1 The space of all optimal sorting sequences 30

4.1.1 The symmetry of the space of sorting sequences 30
4.1.2 An algorithm to enumerate all sorting sequences 31
4.2 Traces 33
4.2.1 The symmetry of traces 35
4.2.2 Normal form of a trace 35
4.2.3 Computing traces by enumerating all sorting sequences 36
4.3 An algorithm to directly enumerate the traces 38
4.3.1 The algorithm 38
4.3.2 Theoretical complexity 40
4.4 Component-specific reversals and trace composition 43
4.5 Implementation and performance 49
4.6 Final remarks 50

Chapter 5 Biological constraints and applications **53**
5.1 Modeling traces with biological constraints 54
5.2 Common intervals 55
5.2.1 Initial detection of common intervals 57
5.2.2 Progressive detection of common intervals 58
5.2.3 Theoretical complexity and experiments 61
5.2.4 Accepting interval breaks 64
5.3 Replication origin and terminus in prokaryotic circular chromosomes 65
5.3.1 Analysis of the *Rickettsia* bacterium 70
5.3.2 Evaluating the execution time 75
5.4 Stratification on evolution of sexual chromosomes 75
5.4.1 Model of evolution by strata 77
5.4.2 Algorithm for exploring the sequences that stratify a permutation 78
5.4.3 Analysis of the results 79
5.4.4 On the execution time of the strata variant 82
5.4.5 A deeper study of the human X and Y sexual chromosomes 82
5.4.5.1 Analysis of strata in extended permutations 83
5.4.5.2 Simulations to estimate the stratification likelihood 84
5.5 Symmetry *versus* asymmetry when applying constraints 88
5.6 Compatibility between constraints 89
5.7 Final remarks 90

Chapter 6 BAOBABLUNA **93**
6.1 Optimization of memory use 94

6.1.1 The compressible sorted set 94
6.1.2 Freezing operations 96
6.1.3 Performance 96
6.2 Architecture 97
6.3 Test 100
6.4 Download and setup 101
6.5 Final remarks 102

Chapter 7 Conclusions, limitations and perspectives 103
7.1 Main results 103
7.1.1 An algorithm to enumerate all the traces 103
7.1.2 Biological constraints 104
7.1.3 Applications 106
7.1.3.1 Analysis of the human sexual chromosomes X and Y 106
7.1.3.2 Analysis of the *Rickettsia* bacterium 107
7.1.4 BAOBABLUNA 107
7.2 Limitations 108
7.3 Future perspectives 108

Appendix A BAOBABLUNA interface 111
A.1 The text representation for the breakpoint graph 111
A.2 Download and setup 112
A.3 Running executable programs 112
A.3.1 `baobab.exec.permutation.analyzeSignedPermutation` 113
A.3.2 `baobab.exec.permutation.sort` 114
A.3.3 `baobab.exec.permutation.performReversals` 115
A.3.4 `baobab.exec.permutation.decomposeSignedPermutation` 115
A.3.5 `baobab.exec.trace.analyzeTraces` 115

Appendix B Dealing with duplications 125
B.1 Repetition-free longest common subsequence 126
B.2 Hardness analysis 126
B.2.1 MAX 2,3-SAT 126
B.2.2 $L-$reducing MAX 2,3-SAT to RFLCS 127
B.3 Experiments 130
B.4 Final remarks 130

Appendix C Extended abstract in French **131**

C.1 Introduction 131

C.2 Permutations, intervalles et inversions 132

C.3 Le tri par inversions et son espace de solutions 133

C.3.1 Traces 133

C.3.2 Un algorithme pour énumérer directement les traces 135

C.4 Contraintes biologiques 136

C.5 Applications 139

C.5.1 Analyse des chromosomes sexuels humains X et Y 139

C.5.2 Analyse des bactéries *Rickettsia* 140

C.6 baobabLuna 140

C.7 Conclusions et limitations 140

C.8 Perspectives 141

Appendix D Article submitted to ***Genome Biology and Evolution*** **143**

List of Figures

1 Nucleotides and the organisation of a DNA molecule. 6
2 The protein synthesis. 7
3 Examples of chromosomes 9
4 The replication of a prokaryotic circular chromosome. 10
5 Examples of large scale rearrangements 11

6 Sorting a genome into another by reversals only 14
7 A transposition or a sequence of three reversals may produce the same rearrangement in a genome 16
8 The breakpoint graph construction 19
9 The effects of a reversal over the breakpoint graph 21
10 Unoriented components in a breakpoint graph 22
11 Eliminating unoriented components 23
12 Merging super-hurdles 24
13 Example of a fortress 24
14 Safe and unsafe reversals 25

15 Decomposing a 4-trace in its prefixes. 39
16 Constructing all the traces of a permutation 41
17 Processing a prefix of a trace when computing traces. 41
18 Distribution of the width of posets for random permutations 44
19 The breakpoint graph of a permutation with two oriented components. 46

20 Sorting sequences of reversals with respect to common intervals 57
21 Progressive perfect sorting sequence of reversals 59
22 Dealing with common interval breaks 65
23 A reversal with respect to the replication terminus of a circular chromosome . . . 67
24 A threshold on the terminus-symmetry rate determines whether a reversal is terminus-symmetric or terminus-asymmetric. 68
25 The inferred position of the replication terminus in a circular chromosome. 68
26 Two equivalent optimal sequences of reversals sorting a circular permutation do not have the same composition with respect to terminus-symmetric and terminus-asymmetric reversals. 69
27 Example of external, terminus-symmetric, and terminus-asymmetric reversals . . 69
28 Phylogenetic tree of *Rickettsia* 70
29 An optimal sequence of reversals to transform the ancestor $R2$ into *Ricketsia felis* 71
30 An alternative optimal sequence of reversals to transform the ancestor $R2$ into *Ricketsia felis* 75

31 Sequence of reversals transforming human X into human Y chromosome 77

32 The structure of `CompressibleSortedSet`. 95
33 Effects of an insertion in `CompressibleSortedSet` 96
34 Comparison of memory use of standard and test versions of the implementation of the algorithm to compute traces in BAOBABLUNA 98
35 The pattern of the architecture of executable programs in BAOBABLUNA. 100

36 Limiting the displacement of the replication terminus in a circular chromosome. . 109

37 The graphical and the text representations of a breakpoint graph 112
38 Get the help of a program in BAOBABLUNA. 113
39 Analyze a linear permutation 113
40 Analyze a circular permutation 114
41 Sort a linear permutation 114
42 Sort a circular permutation 115
43 Perform two subsequent reversals on a linear permutation 116
44 Decompose the breakpoint graph of a linear permutation 117
45 Analyze traces of a linear permutation 118
46 Analyze traces of a linear permutation 118
47 Analyze traces of a linear permutation by trace composition. 119
48 Analyze perfect traces of a linear permutation 120
49 Analyze progressive perfect subtraces of a linear permutation 120
50 Apply the terminus-symmetry constraint for searching subtraces of a circular permutation 121
51 Analyze strata-induced subtraces of a linear permutation 122
52 Apply the terminus-symmetry constraint together with the progressive detection of common intervals for searching subtraces of a circular permutation 122
53 Incompatibility between strata and progressive detection of common intervals when analyzing subtraces of a linear permutation 123

54 An instance of MAX 2,3-SAT with one optimal solution. 127
55 L−reduction of MAX 2,3-SAT(V, C) to RFLCS(x, y) 128

56 La decomposition d'une 4-trace dans ses préfixes. 136
57 La construction de toutes les traces d'une permutation 137

List of Tables

1 The effects of a reversal that is part of an optimal sorting sequence in a breakpoint graph. 25

2 The 28 optimal sequences of reversals sorting $(-3, 2, 1, -4)$. 32
3 Examples of permutations, their sizes, their reversal distances and their number of optimal sorting sequences. 33
4 The two equivalence classes of optimal sequences of reversals sorting $(-3, 2, 1, -4)$. 33
5 Non-equivalent sequences composed by the same reversals. 34
6 The traces of optimal sequences of reversals sorting a permutation. 46
7 The traces of optimal sequences of reversals sorting the components of a permutation. 47
8 Composing the traces that sort a permutation by multiplying the traces that sort its components. 49
9 Execution time of searching all sorting sequences *versus* searching traces 50

10 A 2-tuple representing a progressive perfect subtrace. 61
11 Comparison of the results of computing traces, perfect traces and progressive perfect subtraces 63
12 Traces of optimal sequences sorting *Ricketsia felis* into the ancestor $R2$ 72
13 Progressive perfect subtraces of optimal sequences sorting *Ricketsia felis* into the ancestor $R2$ 73
14 Subtraces of optimal sequences sorting *Ricketsia felis* into the ancestor $R2$ with a bounded number of external and terminus-asymmetric reversals 74
15 Progressive perfect subtraces of optimal sequences sorting *Ricketsia felis* into the ancestor $R2$ with a bounded number of external and terminus-asymmetric reversals 74
16 Comparison of the results of computing traces, progressive perfect subtraces and subtraces with a bounded number of external and terminus-asymmetric reversals 76
17 Traces of sequences of reversals transforming human X into human Y chromosome 81
18 A subtrace of sorting sequences that produce a stratification on the X chromosome. 81
19 A subtrace of sorting sequences that produce an alternative stratification on the X chromosome. 82
20 Comparison of the results of computing traces and strata-induced subtraces . . . 82
21 Homologous markers in human X and Y chromosomes 85
22 Analysis of all optimal sequences of reversals that sort the human X chromosome into Y, with respect to all possible stratifications 87

23 Comparison of the execution time of standard and test versions of the implementation of the algorithm to compute traces in BAOBABLUNA 99
24 Test cases used to assure the quality of BAOBABLUNA. 101

25 Les 28 séquences d'inversion optimales qui trient la permutation $(-3, 2, 1, -4)$. . 134
26 Exemples de permutations, leurs tailles, leurs distances d'inversion et leurs nombre de séquences optimales. 134
27 Les deux classes d'équivalence qui contiennent les séquences optimales pour trier $\pi_A = (-3, 2, 1, -4)$. 134
28 Le temps d'exécution de chercher toutes les séquences d'inversions *versus* chercher toutes les traces 137

List of Algorithms

1 Sorting a signed permutation 26

2 Enumerating all optimal sorting sequences for a permutation 32
3 Adding an element ρ_i to a normal form f of an $(i-1)-$trace: $f+\rho_i$ 36
4 Enumerating all optimal sorting sequences and computing traces 37
5 Enumerating all the traces of a signed permutation 40

6 Enumerating all the progressive perfect subtraces of two signed permutations . . 62
7 Enumerating all the strata-induced subtraces of two signed permutations and a given $k-$strata in the origin permutation 80

Chapter 1

Introduction

This work concerns the algorithmics of genome rearrangements, and is focused on the comparison of two different genomes, mostly considering that the differences observed between them are due to reversals.

Genomes are subject to constant mutations during evolution. Those mutations can be of small scale, such as single nucleotide polymorphisms (SNPs), or of large scale, such as reversals, insertions, deletions, transpositions, fusions and fissions of chromosomes. Reversals are among the events more frequently observed, specially in the evolution of prokaryotes. In eukaryotes, reversals have also been observed. As an example, current theories claim that reversals have a major role to explain the evolution of sexual chromosomes in mammals and in other organisms [40, 55, 60].

One of the most studied problems in a computational approach to pairwise comparative genomics is to determine the rearrangements that have transformed one genome into the other. When the accepted events are restricted to reversals, the genomes are assumed to be free of gene duplications and the orientation of the genes is taken into account, there are polynomial algorithms to calculate the reversal distance between two genomes (that is, the minimum number of reversals required to transform a genome into another) and to determine an optimal sequence of reversals that transforms one genome into the other [32, 33]. Several studies propose algorithms that give one optimal sequence of reversals [4, 11, 26, 31, 63], but there may be a huge number of such sequences. As we will see later in this manuscript, when comparing two short genomes that share only twelve homologous markers, for example, the number of optimal sorting sequences can be greater than $30,000$, and it can be insufficient when attempting to give a biological interpretation to know only one among them.

In order to obtain a more accurate analysis, some additional criteria should be taken in consideration. One strategy is searching for sequences that respect some biological constraints, such as common intervals, which are the list of clusters of co-localised genes between the considered genomes - an optimal sequence of reversals that does not break the common intervals may be more realistic than one that does break them [26]. Another approach, proposed by Siepel [59], is a method to enumerate all optimal sorting sequences. This is however almost as useless as providing only one sequence, because often the sequences are so many that the whole set cannot be presented (when it can be computed). Bergeron *et al.* [9] then provided a way to group the parsimonious sequences into equivalence classes. However, no algorithmic study was performed, and in particular the problem of giving one sequence in each class without enumerating all the sequences was mentioned open.

One of the most important results of our work is a solution to this problem. We propose an algorithm that gives one optimal sequence of reversals that sorts a genome into another per class of equivalence, and counts the number of sequences in each class [18]. The number of classes is much smaller than the number of optimal sequences, but it may still be too big to be interpreted. Thus, to reduce the number of classes even more, we started to put both strategies together, that is, generating only the classes in which there are sequences that respect some biological constraints, such as the common intervals (initially and progressively detected) [17, 18], the composition of the sequence of reversals with respect to the origin and the terminus of the replication (specific to circular prokaryotic genomes), and the stratification of a genome (specific to the X and Y chromosomes evolution). We analyzed qualitatively how the constraints may affect the chronology of the reversals, showing that some of these constraints lead to symmetric (when the results of sorting a genome A into a genome B can be obtained from the results of sorting B into A) and others lead to asymmetric approaches. In addition, a set of constraints can be applied together, under the condition that they are compatible.

We then applied our methods to the analysis of real cases in evolution. In the study of evolution of the human chromosomes X and Y, reversals are believed to have a major role and to have happened in a special order, creating the so called strata on the X chromosome [40, 55, 60]. Using strata as constraints, we were able to generate the classes in which the sequences were in agreement with the given strata. With this method we could test different hypotheses of stratification [18, 41]. In the study of prokaryotic chromosomes, in which it was observed that reversals are more likely to happen around the replication terminus, we used a constraint on the minimum number of such reversals, combining this method with the progressive detection of common intervals in the analysis of the *Rickettsia* bacterium [47, 48], obtaining a better characterization of its evolutionary scenario than previous studies [13].

All the algorithms developed in this work were implemented in an object oriented paradigm, integrated to the software BAOBABLUNA [16], a java framework to deal with genomes and reversals, that is available on-line. In order to be able to deal with the huge amount of data when constructing the classes of equivalent sorting sequences, we developed a java structure that is able to efficiently compress and store the equivalence classes in a sorted set during the construction. We compared the performance of this structure with a java standard implementation of a sorted set, showing that we are able to save memory without losing in the execution time. With BAOBABLUNA, we run experiments of all the variants of our algorithm, showing the gain in the execution time when the biological constraints are applied.

This manuscript is organized as follows. In Chapter 2, we introduce the biological concepts that are required to understand our work and in Chapter 3, we talk about sorting by reversals without gene duplications. Chapters 4 and 5 contain our main results, that are an algorithm that gives a representation of the space of all solutions to the sorting by reversals problem, how to take biological constraints in consideration and the application of this method to study real cases in evolution. In Chapter 6 we talk about BAOBABLUNA [16], a Java framework with the implementations of all algorithms presented in Chapters 4 and 5. Finally, Chapter 7 contains a summary of the results and future perspectives to our work.

The interface of the executable programs in BAOBABLUNA is described in Appendix A. In Appendix B, we describe another result of our work, which is a model to compute a measure of similarity between genomes when duplications are allowed. Our approach, that is called repetition-free longest common subsequence (RFLCS) [1], was proven to lead to an NP-hard problem, as well as other approaches to compare two genomes considering gene duplications [14, 15, 20, 57]. In Appendix C we give an extended abstract of our work in French. And Appendix D contains the study *Footprints of inversions at present and past pseudoautosomal boundaries in* *human sex chromosomes* that we developed in collaboration with Lemaitre *et al.* [41]. This work concerns the evolution of the human sexual chromosomes X and Y and was recently submitted to the journal *Genome Biology and Evolution*.

Chapter 2

Biological concepts

Summary

2.1 Genome 5
2.2 Genes and protein code 6
2.3 Gene duplication 7
2.4 Genome organization 8
2.5 Recombination 8
2.6 DNA replication 10
2.7 Genome rearrangements 11

In this chapter, we present some basic concepts in molecular biology [65] that are important to understand our work.

2.1 Genome

A genome is the hereditary information of an organism and is encoded in a nucleic acid molecule called DeoxyriboNucleic Acid (DNA)[1].

Many physical and chemical properties of the DNA are known. Simply put, we can describe it as a linear macromolecule composed by smaller molecules, the nucleotides. The nucleotides are chemically very similar, except by one of its components, which is a nitrogenous base (often simply called *base*). There are four different types of bases, which are adenine (A), guanine

[1]In some Viruses, the genome is encoded as a simpler kind of molecule, the RiboNucleic Acid (RNA), which is a single strand composed by the nucleotides that carry the bases adenine (A), cytosine (C), guanine (G) and uracil (U).

(G), chemically classified as purines, and cytosine (C), thymine (T), chemically classified as pyrimidines. Thus, there are also four different types of nucleotides, according to the base they carry.

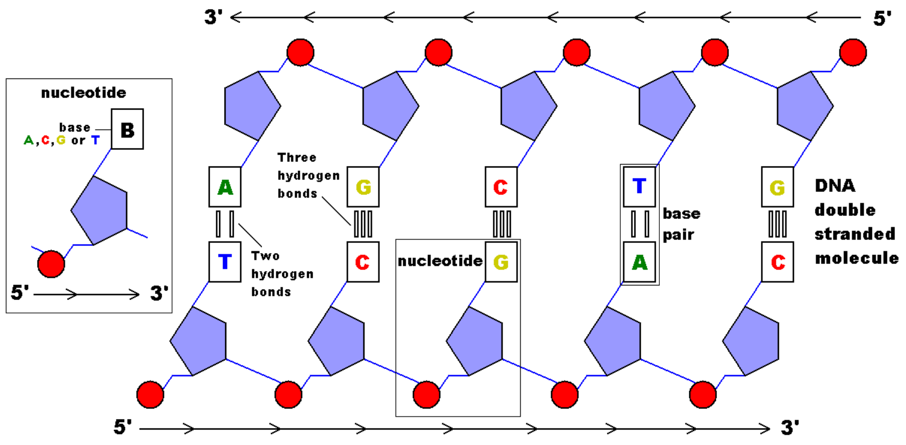


Figure 1: Nucleotides and the organisation of a DNA molecule.

In a DNA molecule, the nucleotides are organized in two complementary helicoidal strands, respecting the base-pairing A-T and G-C. The two extremities of each DNA strand are identified by the symbols $5'$ and $3'$. Then, to each strand is given a direction that goes from the $5'$ extremity to the $3'$ extremity. Moreover, in a DNA molecule, the arrangement of its strands is antiparallel, that is, the direction of the nucleotides in one strand is opposite to the direction of the nucleotides in the other strand (figure 1). Due to this property, one strand is said to be the reverse complement of the other one. The length of a DNA molecule is measured in base pairs (bps, respectively Kbps, Mbps, Gbps).

2.2 Genes and protein code

A genome is a code with all the instructions to synthesize the proteins that participate in the development and functioning of an organism. We call *gene* a portion of the genome that codes for a protein. During protein synthesis, one strand of the DNA molecule is transcripted as a RiboNucleic Acid (RNA), that is, as a single stranded molecule composed by nucleotides that carry the same purines (adenine (A) and guanine (G)), the pyrimidine cytosine (C), and, instead of thymine, the pyrimidine uracil (U). Then, each sequence of three nucleotides (codon) of this

RNA molecule codes for[2] an amino-acid, which is the basic molecule of a protein.

There are 64 different codons (because each one of the three positions of a codon can be occupied by one of the four types of bases, A, C, G or U) that can code for 20 different types of amino-acids. This means that there is a redundancy in the amino-acid coding, that is, some amino-acids are coded by more than one codon. As an example, the aminoacid Phenylalanine (Phe or F) is coded by the codons UUU and UUC, while the aminoacid Tryptophan (Trp or W) is coded only by the codon UGG. Figure 2 illustrates the process of protein synthesis.

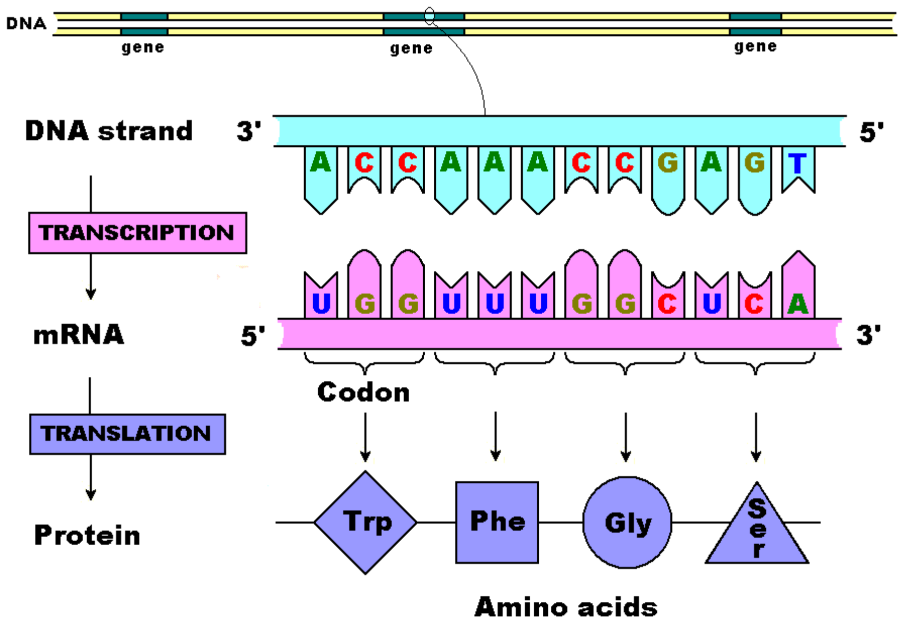


Figure 2: The protein synthesis.

2.3 Gene duplication

Gene duplication is any duplication of a region of DNA that contains a gene. The two genes that exist after a duplication event are called *paralogs* and usually code for proteins with different functions. By contrast, genes that code for proteins with similar functions but exist in different species are called *orthologs*, and are inherited from the common ancestral of these species. Paralogs and orthologs are *homologous* genes. In biological research it is important, but often

[2]In fact, the synthesis process is more complex, and some parts of the RNA are spliced out and not translated. A complete description of protein synthesis can be obtained in [65].

difficult, to differentiate paralogs and orthologs.

2.4 Genome organization

DNA is present in the cells of an organism, distributed in one or more chromosomes. In prokaryotes, which are organisms that lack a cell nucleus (such as bacteria or archeae), the genome is usually composed by a single circular chromosome (figure 3 (A)), whose size varies at least from 160 Kbps (the bacterium *Candidatus Carsonella ruddii* [46]) to 13 Mbps (the bacterium *Sorangium cellulosum* [54, 58]).

Organisms that have a cell nucleus, the eukaryotes (animals, plants, fungi, and protists), usually have a genome more complex, composed of a group of linear chromosomes. In eukaryotes, the chromosomes are located inside the cell nucleus and a genome can be as small as 20 Mbps (*Saccharomyces cerevisiae* [29]) or as large as 670 Gbps (*Amoeba dubia* [52]). The relationship between genome size and organism complexity is not clear. The human genome (*Homo sapiens*), for instance, has only 3.25 Gbps, which means that *Amoeba dubia*, an unicellular organism, has a genome over 200 times longer than the human genome.

Moreover, eukaryotes generally have a genome less dense in genes than prokaryotes, but the relationship between genome size and the number of genes is also not clear. The bacterium *Escherichia coli* has 4,243 genes in its genome of 4.64 Mbps, while the plant *Arabidopsis thaliana* has a genome of 119 Mbps and 28,159 genes, and it is estimated that the human genome may have about 26,500 genes in its 3.25 Gbps.

2.5 Recombination

Eukaryotes usually have a fixed number of copies of each chromosome, and the number of homologous sets of chromosomes in the cell of an organism is said to be its **ploidy** (all the copies of the same chromosome are said to be homologous chromosomes). When all the cells of an organism have only one copy of each chromosome, the organism is said to be haploid.

In general, the cells of organisms that have a sexual reproduction are diploid, containing one set of chromosomes from each parent. In other words, most of the cells in a diploid organism

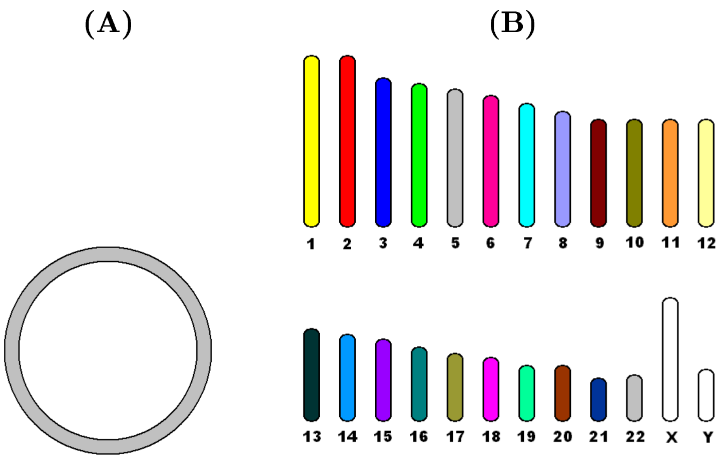


Figure 3: A. A prokaryotic circular chromosome. B. The 24 linear chromosomes of the human species (Homo sapiens) are separated in two groups, the autosomes, numbered from 1 to 22, and the sex-determining, which are the chromosomes X and Y.

contain pairs of homologous chromosomes. Usually, one of these pairs is composed by the *sex-determining chromosomes* (identified by X and Y in humans and other mammals) and the others are pairs of *autosomes*, which are homologous non sex-determining chromosomes. Figure 3 (B) illustrates the chromosomes of the human species.

The composition of the sex-determining pair of chromosomes is different in males and females. In human, females have a pair with two copies of the X chromosome, and males have a pair with two different chromosomes X and Y, but there are also other possible patterns. In birds, for instance, males have a pair with two copies of the same chromosome, and females have a pair with two different chromosomes. The gametes (male or female germ cells) of a diploid organism are haploid, that is, they have only one set of autosomes and one sex-determining chromosome.

The production of gametes in a diploid organism occurs when a specific kind of its diploid cells divides by a process called **meiosis**. The result of the meiosis of a diploid cell are four haploid cells, and one or more among these haploid cells originate the gametes. During meiosis, a phenomenon may happen that is called chromosomal **crossing over**, by which a pair of homologous autosomes **genetically recombine**, exchanging sections of their DNA.

Crossing over and genetic recombination also happen between homologous sex-determining chromosomes (such as the pair XX in mammalian females) and even between homologous portions of non-homologous sex-determining chromosomes. It has been observed, for instance, that although human chromosomes X and Y are very different, they have two homologous regions at their extremities in which genetic recombination occurs. These regions are called pseudo-autosomal regions.

2.6 DNA replication

DNA replication is the process of copying a double-stranded DNA molecule to form two double-stranded molecules. It occurs when a cell divides itself to create two copies with the same DNA content as the original cell (as in asexual reproduction of unicellular organisms, or in the **mitosis** of an eukaryotic cell). As each DNA strand is the reverse complement of the other, both strands can serve as templates for copying the opposite strand. Thus, during DNA replication, each one of the strands of the original molecule is preserved in its entirety and is used as a template for the assembly of a new strand.

The process of replication happens in different ways, depending on the organism and the molecule type. In particular, the chromosome of a prokaryotic organism, which is generally circular, has a bidirectional replication process. In a specific position of the chromosome, identified as the replication origin, the replication starts in both directions. The simultaneous replication processes will then finish in the opposite side of the molecule, in a position identified as the replication terminus (see Figure 4).

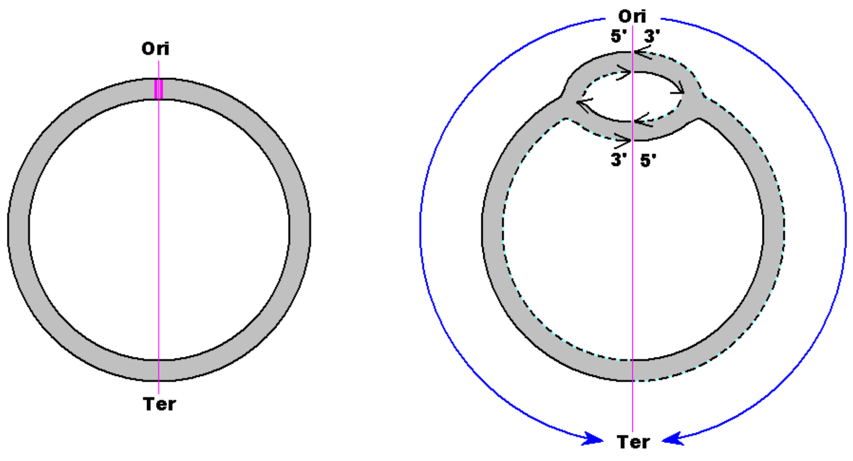


Figure 4: The replication of a prokaryotic circular chromosome.

In eukaryotes the replication process is in general much more complicated, with several replication origins and terminuses through the chromosomes.

2.7 Genome rearrangements

Genomes are not static, but are subject to continuous mutations during the cellular processes. These mutations can be of different types and scales. Events such as single nucleotide polymorphisms (SNPs), that affect only one nucleotide at a time, are said to be small scale events, and are more frequent than large scale events [53]. The main known rearrangements or large scale events are reversals of large portions of chromosomes, insertions of new genes (usually due to duplications or horizontal transfer between species), deletions or loss of genes, transpositions of DNA fragments within a chromosome, fusions and fissions of chromosomes, translocations of DNA fragments between chromosomes. Some examples of observed rearrangements between current species are represented in Figure 5 [13, 35].

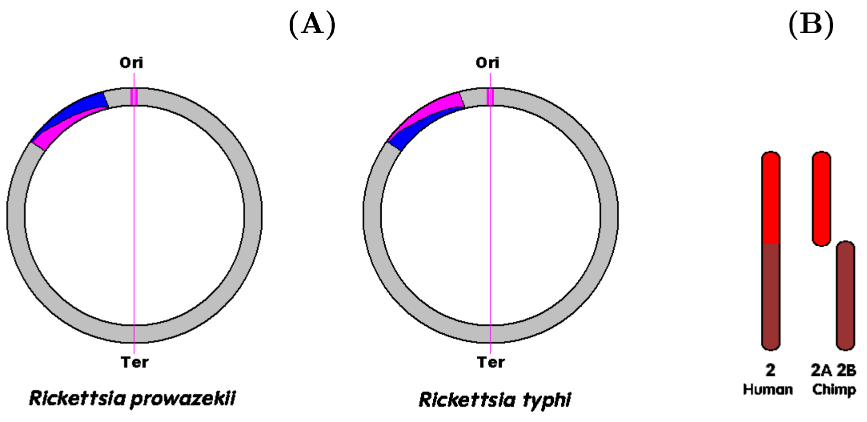


Figure 5: A. The most important difference between the circular chromosomes of the bacteria Rickettsia prowazekii and Rickettsia typhi [13] is very likely to be due to a reversal. B. The human chromosome 2 seems to be a fusion of chimpanzee's chromosomes 2A and 2B [35].

Reversals are among the rearrangement events more frequently observed in the evolution of organisms, specially prokaryotes. Most of the existing differences between six species of the *Rickettsia* bacteria can be explained by reversals [13, 17]. In eukaryotes, reversals have also been observed [44] and current theories claim that reversals may also have an important role in the evolution of sex-determining chromosomes [40, 41, 55, 60]. The rates of different rearrangement events in whole genomes of several eukaryotic species were studied in [43], showing that reversals are considerably frequent. Even if these and other examples do not prove that reversals happen in practice, they are well accepted as indices of the occurrence of reversals.

In comparative genome studies, genomes are compared in order to determine the rearrangements that may have happened between them. Our work is in the algorithmics of comparative genomics, and is mostly focused on the problem of sorting one genome into another when gene duplications are not allowed and the rearrangement events are restricted to reversals. Observe that, in this case, only unichromosomal genomes can be analyzed, thus this approach is more adequate to analyze the evolutionary scenario of prokaryotes and of some special cases in eukaryotes.

In the next chapter we will describe in detail the problem we are interested in, that is called *sorting by reversals.*

Chapter 3

Sorting by reversals

Summary

3.1 **Permutations, intervals and reversals** 16
3.2 **The breakpoint graph and the reversal distance** 18
3.3 **Safe and unsafe reversals** 25
3.4 **Sorting a signed permutation** 26
3.5 **The symmetry of sorting by reversals** 27
3.6 **Component-specific reversals** 27

The classical algorithmic problems in pairwise comparative genomics are to compute the rearrangement distance between two genomes [33], that correspond to the minimum number of rearrangement events that are required to transform one genome into the other, and to determine an optimal sequence of events to transform one genome into the other. These problems have several variations, according to the events that may be considered [63].

Our research is mostly focused on rearrangement problems restricted to reversal events and, in this chapter, we talk about sorting one unichromosomal genome into another by reversals when gene duplications and insertions are not allowed. Observe that we also assume that the order of the genes is known in both genomes, which often is not true in practice [66]. One of the first studies that proposed algorithms to compute the reversal distance between two genomes was developed by Kececioglu and Sankoff [38], with an approach that does not take into account the orientation of the genes. Later this approach, called *unsigned sorting by reversals*, was proven to lead to an NP-hard problem [22]. We worked on a different approach, called *signed sorting* *by reversals*, or simply *sorting by reversals*, in which the orientation of the genes is taken into account. Kececioglu and Sankoff [38] had already observed that some aspects of signed sorting by reversals were easier to analyze, and, indeed, this approach can be solved in polynomial time [32, 33], as we will describe in this chapter.

Despite the simplifications (not considering duplications or insertions and assuming that the order of the genes is known in both genomes) mentioned above, the sorting by reversals problem is very interesting. From the biological point of view, as we said before, reversals are frequently observed, specially in prokaryotes. And reversals are also interesting from the algorithmic point of view. First we note that it is always possible to sort a genome into another by reversals. In the worst case, we need two reversals to put each marker of the first genome in the position that it occupies in the second genome (one reversal to put the marker in the proper position and eventually a second reversal to inverse its orientation). Thus, if the two considered genomes has n homologous markers, in the worst case we need $2n$ reversals to sort one genome into the other. We will see later in this chapter that in general at most n reversals are sufficient to sort a genome into another and a fictitious example is given in Figure 6.

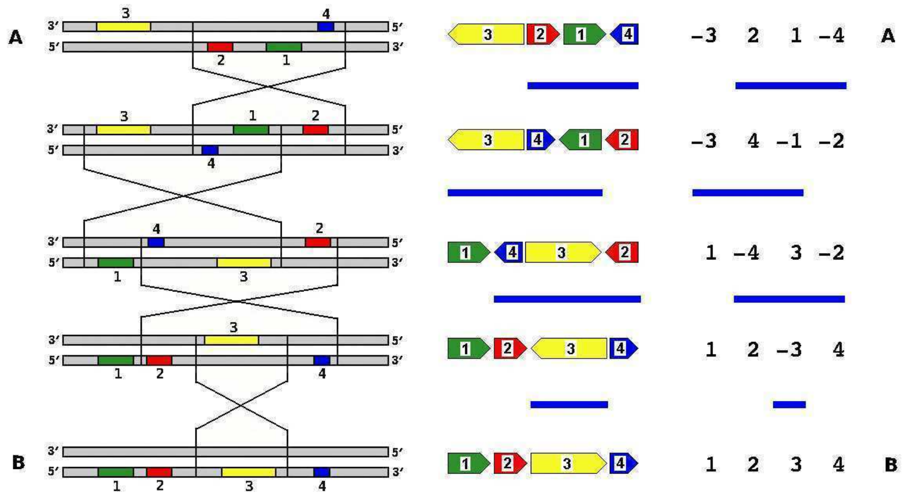


Figure 6: Sorting genome A into genome B by reversals only. Homologous markers (usually genes) are identified by the same numbers and colours. Signs indicate the DNA strand the markers lie on.

Computing the reversal distance, that is, the minimum number of reversals that are required to transform one genome into the other, and finding an optimal sorting sequence can be solved in polynomial time [32, 33]. These two problems have been the topic of several works. The fastest algorithm to compute the distance takes $O(n)$ time [4] and the fastest way to find an optimal sorting sequence is subquadratic [11, 31, 63]. It is possible that this mathematical notion of reversal distance and the method of searching optimal sequences can underestimate the actual number of steps that occurred biologically. However, the solutions of these two problems are still valuable tools that help to analyze and to understand evolutionary scenarios. Currently, there are at least two available softwares to solve these problems. One is the package `GRAPPA`[3], that is discussed in more detail in [45] and contains the fastest algorithm to compute the reversal distance (mentioned above). The other is the software `GRIMM`[4], that is described in [64] and contains one of the most used programs to sort a genome into another by reversals. These programs were used in particular by Ross *et al.* [55] in the analysis of the human sexual chromosomes X and Y and by Blanc *et al.* [13] in the analysis of the *Rickettsia* bacteria.

Observe that with reversals we can simulate a transposition, that is another possible rearrangement event in unichromosomal genomes. A transposition is said to happen when two consecutive markers of a genome exchange their positions. It is always possible to produce the same result as a transposition with a sequence of three reversals (see Figure 7). Thus a sequence of m transpositions can always be transformed in a sequence of $3m$ reversals. However, this does not mean that there is a clear relation between the reversal distance and the transposition distance. Eventually a sequence of m transpositions can be replaced by a sequence with less than $3m$ reversals. Moreover, although the reversal distance can be obtained in polynomial time, the complexity of computing the transposition distance is still an open problem in the algorithmics of genome rearrangements [5].

In the rest of this chapter we will introduce our notation and explain the classical approach of Hannenhali and Pevzner [32, 33, 53] for the sorting by reversals problem.

[3]The package `GRAPPA` (Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms) contains several programs to deal with genome rearrangements and can be downloaded at `http://www.cs.unm.edu/~moret/GRAPPA/`.

[4]The software `GRIMM` contains also algorithms for multichromosomal genome rearrangements and is available online at `http://grimm.ucsd.edu/GRIMM/`.

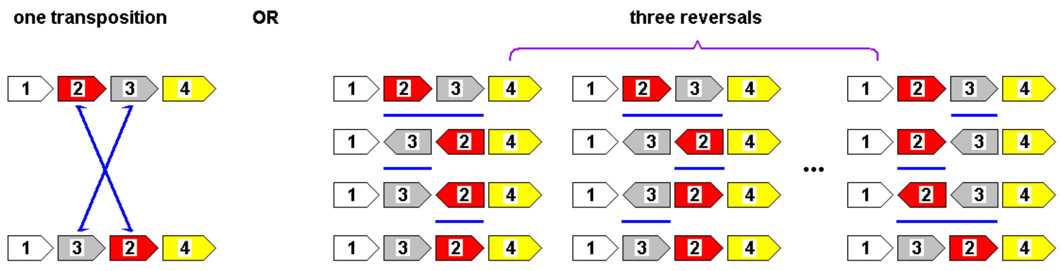


Figure 7: A transposition or a sequence of three reversals may produce the same rearrangement in a genome. Observe that the three reversals can be applied in different orders.

3.1 Permutations, intervals and reversals

We represent the studied genomes by the list of homologous markers (usually genes or blocks of contiguous genes) between them. These homologous genomic markers are represented by the integers $1, 2, \ldots, n$, with a plus or minus sign to indicate the strand they lie on. The order and orientation of the markers of one genome in relation to the other is represented by a *signed permutation* $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ of size n over $\{-n, \ldots, -1, 1, \ldots, n\}$, such that, for each value i from 1 to n, either i or $-i$ is mandatorily represented, but not both. The *identity permutation* $(1, 2, 3, \ldots, n)$ is denoted by $\mathcal{I}_n$.

A subset of numbers $\rho \subseteq \{1, 2, \ldots, n-1, n\}$ is said to be an *interval* of a permutation π if there exist $i, j \in \{1, \ldots, n\}$, $1 \leq i \leq j \leq n$, such that $\rho = \{|\pi_i|, |\pi_{i+1}|, \ldots, |\pi_{j-1}|, |\pi_j|\}$. Given a permutation π and an interval ρ of π, we can apply a *reversal* on the interval ρ of π, that is, the operation which reverses the order and flips the signs of the elements of ρ, denoted by $\pi \circ \rho$. If $\pi = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_{n-1}, \pi_n)$ and $\rho = \{|\pi_i|, |\pi_{i+1}|, \ldots, |\pi_{j-1}|, |\pi_j|\}$,

$$\pi \circ \rho = (\pi_1, \pi_2, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_{i+1}, -\pi_i, \pi_{j+1}, \ldots, \pi_{n-1}, \pi_n).$$

For example, with the permutation $\pi = (-3, 2, 1, -4)$ and the interval $\rho = \{1, 2, 4\}$ we have $\pi \circ \rho = (-3, 4, -1, -2)$. Due to this, an interval ρ can also be used to denote a reversal.

We say that a permutation is *linear* when it represents a linear chromosome, or *circular* when it represents a circular chromosome. When a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ is circular, the circular permutation $\overline{\pi} = (-\pi_n, -\pi_{n-1}, \ldots, -\pi_2, -\pi_1)$ (generated by a reversal over all values of π) and all circular permutations obtained by a *shift* in π or $\overline{\pi}$ are equivalent to π. A shift of i

elements in a circular permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-i}, \pi_{n-i+1}, \pi_{n-i+2}, \ldots, \pi_{n-1}, \pi_n)$ transfers the last i elements of π to the beginning of π. This operation generates the circular permutation $(\pi_{n-i+1}, \pi_{n-i+2}, \ldots, \pi_{n-1}, \pi_n, \pi_1, \pi_2, \ldots, \pi_{n-i})$. Observe, for example, that the circular permutations $\pi = (-3, 2, 1, -4)$ and $\pi' = (-1, -2, 3, 4)$ are equivalent (we can obtain π' by applying a shift of 3 on $\overline{\pi}$).

For a given permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$, we say that there is a ***point*** between each pair of consecutive values π_i and π_{i+1} in π. In addition, if π is circular, there is one additional point between π_n and π_1. If π is linear, there are two additional points, one before π_1 and the other after π_n. We denote by $pts(\pi)$ the number of points in a permutation π. Thus, if π is circular, then $pts(\pi) = n$. Otherwise π is linear and $pts(\pi) = n + 1$.

When we analyze a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ with respect to another permutation π_T, each point in π can be an ***adjacency*** or a ***breakpoint***. We say that a pair of consecutive values (π_i, π_{i+1}) in π is an adjacency between π and π_T when either the values in the pair (π_i, π_{i+1}) or the values in the pair $(-\pi_{i+1}, -\pi_i)$ are consecutive in π_T. Moreover, if the permutations are circular, we assume that π_n is the last value of π_T[5], and the pair (π_n, π_1) is an adjacency when π_1 is the first value in π_T. If the permutations are linear, we have an adjacency before π_1 if π_1 is also the first value in π_T and an adjacency after π_n if π_n is also the last value of π_T. All points that are not adjacencies between π and π_T are called breakpoints. We denote by $adj(\pi)$ the number of adjacencies and by $brp(\pi)$ the number of breakpoints in a permutation π. It is easy to see that $brp(\pi) = pts(\pi) - adj(\pi)$. Observe that, if π is sorted, that is, $\pi = \pi_T$, then π has only adjacencies and no breakpoints, and, if $\pi \neq \pi_T$, then π has at least one breakpoint.

A ***sequence*** or ***$i-$sequence*** of reversals $\rho_1\rho_2\ldots\rho_i$ is valid for a permutation π if ρ_1 is an interval of π, ρ_2 is an interval of $\pi \circ \rho_1$, ρ_3 is an interval of $(\pi \circ \rho_1) \circ \rho_2$, and so on. If $\rho_1\rho_2\ldots\rho_i$ is a valid $i-$sequence of reversals for a permutation π, then $\pi \circ \rho_1\rho_2\ldots\rho_i$ denotes the consecutive application of the reversals ρ_1, ρ_2, $\ldots\rho_i$ in the order in which they appear. We say that an $i-$sequence of reversals $\rho_1\ldots\rho_i$ ***sorts*** a permutation π into a permutation π_T if $\pi \circ \rho_1 \ldots \rho_i = \pi_T$.

The length of a shortest sequence of reversals sorting a permutation π into π_T is called the ***reversal distance*** of π and π_T, and is denoted by $d(\pi, \pi_T)$. Let $s = \rho_1\rho_2\ldots\rho_i$ be a valid

[5]If the permutations are circular, without loss of generality, we can assume that the last value in π and π_T are the same; if it is not the case, we take as π an equivalent circular permutation with this characteristic.

$i-$sequence of reversals for a permutation π. If $d(\pi \circ s, \pi_T) = d(\pi, \pi_T) - i$, then s is said to be an *optimal $i-$sequence*. Moreover, if s is an optimal $i-$sequence and $i = d(\pi, \pi_T)$, then s is simply called an *optimal sorting sequence* for π and π_T. We also define the $k-$prefix of an optimal sorting sequence s as the sequence composed by the first k reversals of s. Observe that if s' is a $k-$prefix of an optimal sequence s sorting π into π_T, then $d(\pi \circ s', \pi_T) = d(\pi, \pi_T) - k$, that is, s' is an optimal $k-$sequence for π and π_T. For example, if we consider two linear permutations $\pi = (-3, 2, 1, -4)$ and $\pi_T = \mathcal{I}_4$, we have $d(\pi, \pi_T) = 4$ and one optimal sorting sequence is $\{1,2,4\}\{1,3,4\}\{2,3,4\}\{3\}$, whose $1-$, $2-$ and $3-$prefixes are $\{1,2,4\}$, $\{1,2,4\}\{1,3,4\}$ and $\{1,2,4\}\{1,3,4\}\{2,3,4\}$.

Henceforth we will generally use simply the term sequence or $i-$sequence to refer to an optimal sequence or optimal $i-$sequence of reversals. Moreover, for the purposes of our work, the initial and the target permutations π and π_T are either both linear, or both circular. Without loss of generality, we often omit the target permutation π_T. In this case, π_T corresponds to the identity permutation $\mathcal{I}_n = (1, 2, 3, \ldots, n)$, where n is the size of the initial permutation π, and the notation $d(\pi)$ is equivalent to $d(\pi, \mathcal{I}_n)$.

3.2 The breakpoint graph and the reversal distance

As mentioned, given a permutation π, calculating $d(\pi)$ and finding one optimal sequence of reversals sorting π can be computed in polynomial time. The classical approach for analyzing these two problems was developed by Hannenhalli and Pevzner [8, 32, 33, 53] and is based on a special structure called the *breakpoint graph*, whose edges can be *black* or *gray*.

For a given permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$, we construct the breakpoint graph of π as follows. If π is linear, we may add the values 0 and $n+1$, that represent the extremities of the chromosome, obtaining the permutation $\pi' = (0, \pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n, n+1)$. If π is circular (without loss of generality we assume $\pi_n = n$), we may add only the value 0, obtaining the permutation $\pi' = (0, \pi_1, \pi_2, \ldots, \pi_{n-1}, n)$. Then we may link each pair of consecutive values by a horizontal black edge (each black edge represents a point in the permutation). Lastly, we may link with gray edges the first extremity of the black edge that appears after zero or a positive value i (analogously the last extremity of the black edge that appears before a negative value $-i$)

with the last extremity of the black edge that appears before a positive value $i + 1$ (analogously the first extremity of the black edge that appears after a negative value $-(i + 1)$). Thus, each gray edge links extremities of black edges. At the end, we have a graph with a collection of cycles, and in each cycle black and gray edges alternate. When a cycle contains only one black and one gray edge, it covers an adjacency and is called *trivial cycle*. The cycles that contain four or more edges cover at least two breakpoints and are called *long cycles*. The construction of the breakpoint graph of a linear permutation is illustrated in Figure 8 (A).

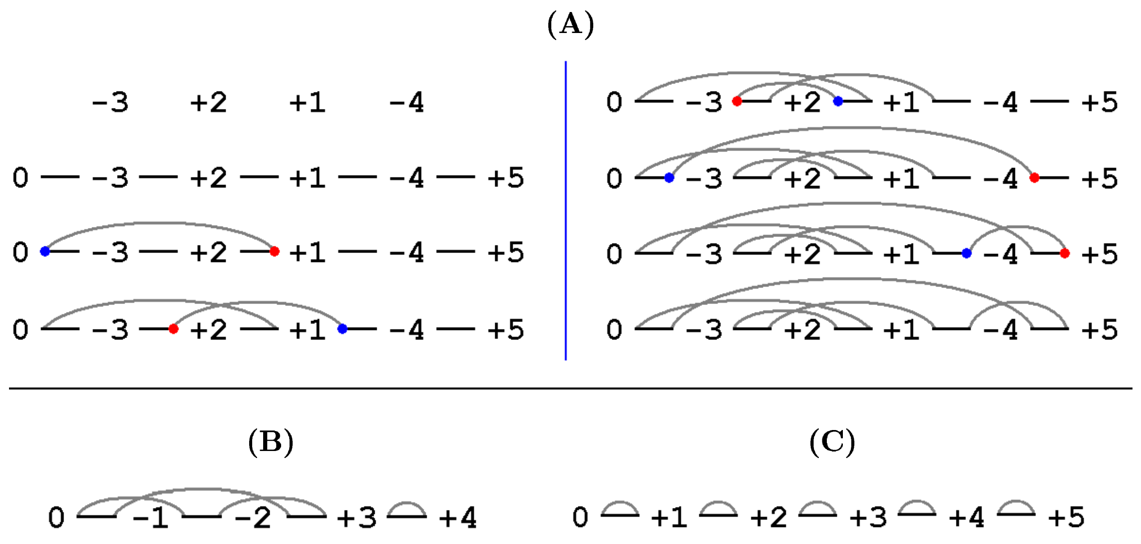


*Figure 8: (A) The construction of the breakpoint graph for the linear permutation $\pi = (-3, 2, 1, -4)$ is done by the following steps: 1- add the values **0** and **+5**, that represent the extremities of the chromosome; 2- link each pair of consecutive values by a black edge. 3- link with gray edges the first extremity of the black edge that appears after zero or a positive value i (analogously the last extremity of the black edge that appears before a negative value $-i$) with the last extremity of the black edge that appears before a positive value $i + 1$ (analogously the first extremity of the black edge that appears after a negative value $-(i + 1)$). The obtained breakpoint graph has one long cycle with five breakpoints and no adjacencies. (B) The breakpoint graph for the circular permutation $(-3, 2, 1, -4)$, which is equivalent to the circular permutation $(-1, -2, 3, 4)$. In this case, in the first step we may add only the value **0** in the beginning of $(-1, -2, 3, 4)$, henceforth the procedure is identical. This graph has two cycles: one trivial cycle (which correspond to the adjacency between 3 and 4) and one long cycle with three breakpoints. (C) The breakpoint graph for the linear permutation $\mathcal{I}_4 = (1, 2, 3, 4)$. This graph has five trivial cycles (each trivial cycle is an adjacency) and no breakpoints.*

Observe that, for a given permutation π, the breakpoint graph is different depending on whether π is linear or circular, as we can see comparing the graph for the linear permutation $(-3, 2, 1, -4)$ and the circular permutation $(-3, 2, 1, -4)$ (Figure 8 (A) and (B)). However, they can be analyzed exactly in the same way, that is, the only difference between circular and linear permutation analyses is the breakpoint graph construction. Thus, without loss of generality, henceforth we will often talk about breakpoint graphs, without specifying whether the corresponding permutations are linear or circular. To denote the breakpoint graph of a permutation π, we will use the same symbol π.

If a permutation π is sorted, it has only adjacencies, and the resulting breakpoint graph is a collection of $pts(\pi)$ trivial cycles (see Figure 8 (C)). A breakpoint graph that has only trivial cycles is said to be sorted. Since a long cycle contains at least two breakpoints, if π is unsorted, then π has at most $pts(\pi) - 1$ cycles. This indicates that, in order to sort a permutation, we may induce an increase of the number of cycles in its corresponding breakpoint graph. The number of cycles in the breakpoint graph of a permutation π is denoted by $cyc(\pi)$.

Hannenhalli and Pevzner [32, 33, 53] described the effects of a reversal ρ over a breakpoint graph π. The authors demonstrated that a reversal ρ is either a *split reversal*, that increases the number of cycles by one, (in this case we have $cyc(\pi \circ \rho) = cyc(\pi) + 1$), or a *joint reversal*, that decreases the number of cycles by one (in this case we have $cyc(\pi \circ \rho) = cyc(\pi) - 1$), or a *neutral reversal*, that maintains the number of cycles unchanged (in this case we have $cyc(\pi \circ \rho) = cyc(\pi)$). In order to characterize these three types of reversals, we assign a direction to each black edge, according to an arbitrary tour in each cycle of the graph. Then, if the extremities of the reversal are in black edges in the same cycle and have opposite directions, we have a split reversal. If the extremities of the reversal are in black edges in different cycles, we have a joint reversal (independently of the directions of the black edges). Finally, if the extremities of the reversal are in black edges in the same cycle and have the same direction, we have a neutral reversal that does not change the number of cycles in the graph. To understand the reasons of these effects, we should investigate how the reversals affect the topology of the graph. In fact, only the two black edges that correspond to the extremities of the reversal are modified. Although some vertices may also have their corresponding values inversed, all the other edges in paths that alternate gray and black edges remain unchanged (consequently, their relative directions remain also unchanged). Figure 9 illustrates the three types of reversals.

In order to sort a permutation, we must maximize the number of split reversals in the sorting sequence. With this information, we can start to conceive the formula for the reversal distance. If we can find a sequence s that has only split reversals for sorting a breakpoint graph π, the

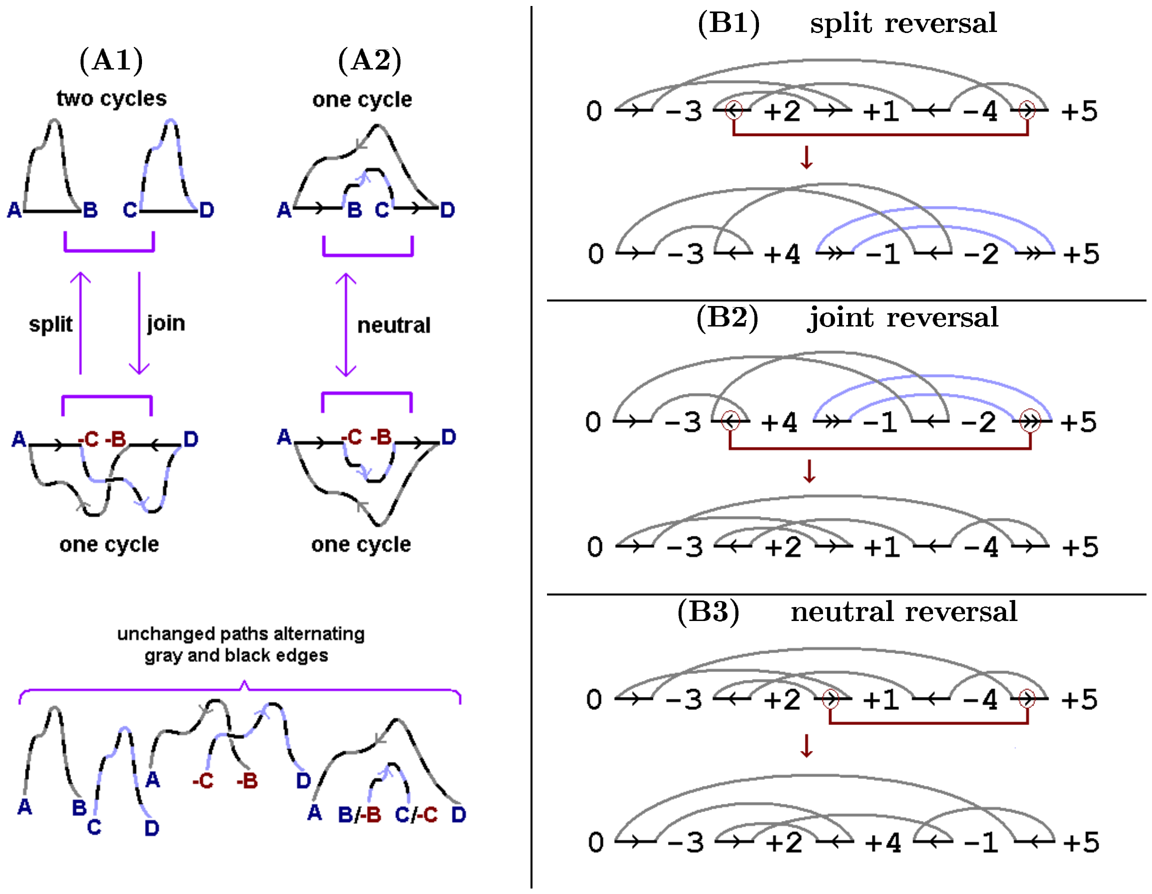


Figure 9: The effects of a reversal over the breakpoint graph. We may assign a direction to each black edge, by an arbitrary tour in each cycle of the graph. The images A1 and A2 illustrate how a reversal affects the topology of the graph. The point A,B (respectively A,-C) appears before the point C,D (respectively -B,D) in the considered permutations. Observe that, with respect to the topology, only the two black edges that correspond to the extremities of the reversal are modified. All the other edges in paths that alternate gray and black edges remain unchanged, although the vertices that are between B and C in the permutation must have their corresponding values inversed. (A1) The two cycles on the top are joined by a reversal whose extremities are in the represented black edges. Inversely, the unique cycle on the botton is split by a reversal whose extremities are in the represented black edges, that have opposite directions. (A2) The number of cycles in the graph is not changed by a reversal whose extremities are in black edges in the same cycle, with the same direction. The images B1, B2 and B3 show the effects over the breakpoint graphs represented in the standard form. (B1) Split reversal: a reversal whose extremities are in black edges in the same cycle and opposite directions may break the cycle in two. (B2) Joint reversal: A reversal whose extremities are in black edges in different cycles may join the two cycles in one (independently of the directions of the black edges). (B3) Neutral reversal: a reversal whose extremities are in black edges in the same cycle and same directions does not change the number of cycles in the graph.

length of s is $pts(\pi) - cyc(\pi)$. However, a split reversal does not always exist. For example, if all black edges of all cycles in the graph have the same direction, we cannot perform a split reversal (Figure 10 (A)). Thus, in some cases, we may need to add some joint and/or neutral reversals in a sorting sequence, and the reversal distance is $d(\pi) \geq pts(\pi) - cyc(\pi)$.

Fortunately, it is always possible to calculate the number of non-split reversals in a sorting sequence. We can define an exact formula to the reversal distance, but first we need to define other properties of the breakpoint graph. When a cycle in the graph has black edges with opposite directions, it is called an *oriented cycle*. Otherwise all black edges in the cycle have the same direction and we have an *unoriented cycle*. A component of the graph is a collection of cycles, such that each cycle of the component has at least one gray edge that overlaps with a gray edge of another cycle in the component. Adjacencies are trivial components, and a non-trivial component contains at least two breakpoints. When a non-trivial component has at least one oriented cycle, it is an *oriented component*. Otherwise it is an *unoriented component*. Figure 10 (B) shows a breakpoint graph with an oriented and an unoriented component.

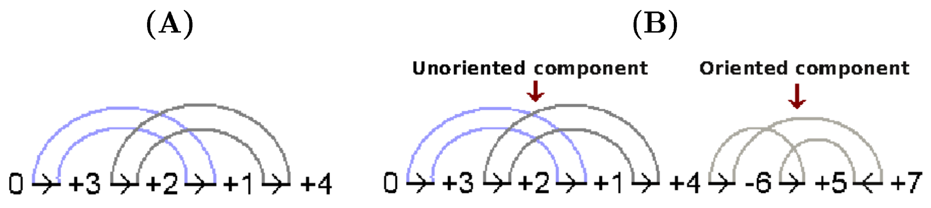


Figure 10: (A) A breakpoint graph in which we cannot perform a split reversal. (B) A breakpoint graph with an oriented and an unoriented component.

A reversal ρ is called *cut reversal* when its extremities are in the same cycle of an unoriented component. A cut reversal is always neutral and transforms an unoriented component into an oriented component (Figure 11 (A)), thus we say that a cut reversal eliminates an unoriented component (observe that a cut reversal does not change the number of cycles in the breakpoint graph). When the breakpoint graph has more than one unoriented component, it is not always necessary to use one cut reversal for each unoriented component. An unoriented component Y separates two other unoriented components X and Z when there is a black edge of Y between any black edge of X and any black edge of Z. In this case, a reversal that has one extremity in X and one extremity in Z will regroup the components X, Y and Z into one oriented component (Figure 11 (B)); this kind of reversal is called *merge reversal*. A merge reversal is always a joint reversal that regroups i unoriented components into one oriented component, for $i \geq 2$, thus we say that a merge reversal eliminates $i \geq 2$ unoriented components (observe that a merge reversal decreases the number of cycles in the breakpoint graph by one).

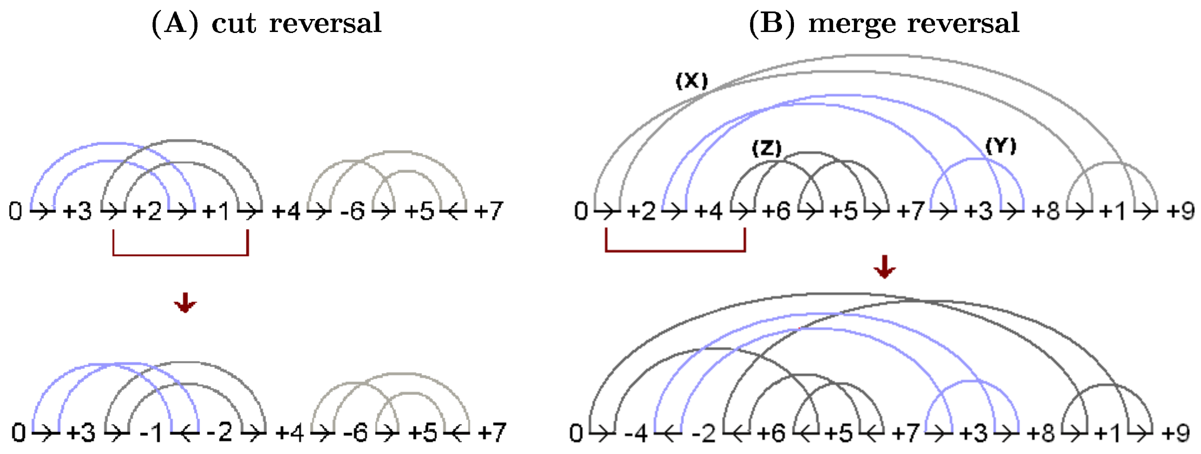


Figure 11: (A) A cut reversal transforms an unoriented into an oriented component and does not change the number of cycles in the breakpoint graph (it is a neutral reversal). (B) The unoriented component Y separates the unoriented components X and Z. A merge reversal regroups the unoriented components X, Y and Z into one oriented component and decreases the number of cycles in the breakpoint graph of one (it is a joint reversal).

An unoriented component that does not separate two other unoriented components is called a *hurdle*. We represent by $hrd(\pi)$ the number of hurdles in a breakpoint graph π. Since a hurdle does not separate unoriented components, each hurdle X can be eliminated either by a cut reversal whose extremities are in points of the same cycle of X (Figure 11 (A)), or together with another hurdle Z by a merge reversal whose extremities are in a point of X and a point of Z (Figure 11 (B)). A cut reversal eliminates one hurdle and does not change the number of cycles in the graph, while a merge reversal eliminates two hurdles at once, and decreases the number of cycles in the graph by one. Thus, each hurdle requires one additional reversal and we can improve the distance formula to $d(\pi) \geq pts(\pi) - cyc(\pi) + hrd(\pi)$. We say that a hurdle Z protects an unoriented component Y that is not a hurdle, if Y becomes a hurdle after the elimination of Z by a cut reversal. In this case, the hurdle Z is called *super-hurdle*. Eliminating a super-hurdle by a cut-reversal does not decrease the number of hurdles in the graph (Figure 12), consequently a super-hurdle may always be eliminated together with another super-hurdle by a merge reversal, that will regroup the two super-hurdles and their corresponding protected unoriented components into one oriented component (Figure 11 (B)).

It remains only one particular case to complete the reversal distance formula. When all the i hurdles of a breakpoint graph are super-hurdles and i is an odd number, the permutation requires an additional effort to be sorted. A breakpoint graph with this characteristic is called a

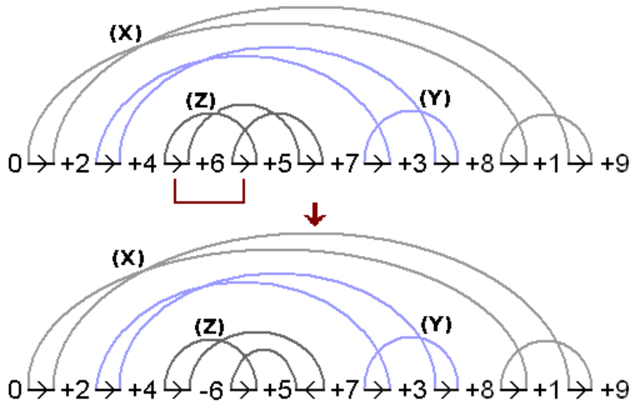


Figure 12: The unoriented component Y separates the super-hurdles X and Z. After eliminating the super-hurdle Z by a cut reversal, the component Y becomes a hurdle, thus the number of hurdles in this graph is not reduced after applying this cut reversal.

fortress. One additional reversal is sufficient to eliminate the fortress (this reversal may be chosen among several possibilities, for example a cut reversal to eliminate a hurdle, or a merge reversal regrouping two hurdles). We denote by $frt(\pi)$ a value that indicates whether the breakpoint graph π is a fortress or not. Thus, if π is a fortress, then $frt(\pi) = 1$, otherwise $frt(\pi) = 0$.

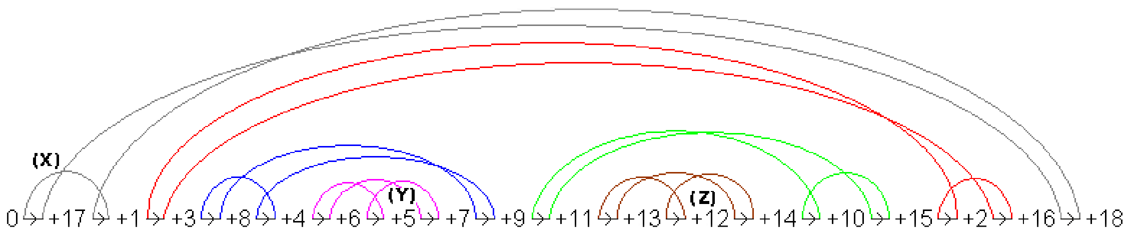


Figure 13: A fortress with 3 super-hurdles (X, Y and Z).

Table 1 summarizes the effects of a reversal that is part of an optimal sorting sequence in a breakpoint graph. The final formula for the reversal distance is:

$$d(\pi) = pts(\pi) - cyc(\pi) + hrd(\pi) + frt(\pi)$$

Remember that if $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$ is a linear permutation, then $pts(\pi) = n + 1$. Otherwise π is a circular permutation and $pts(\pi) = n$.

Reversal	**Type**	$\Delta_{cyc(\pi)}$	$\Delta_{hrd(\pi)}$	$\Delta_{frt(\pi)}$
`split`	`split`	$+1$	0	n/A
`hurdle cut`	`neutral`	0	-1	n/A
`hurdle merge`	`joint`	-1	-2	n/A
`fortress elim by unor. comp. cut`	`neutral`	0	0	-1
`fortress elim by unor. comp. & hurdle merge`	`joint`	-1	-1	-1

Table 1: The effects of a reversal that is part of an optimal sorting sequence in a breakpoint graph. The columns $\Delta_{cyc(\pi)}$, $\Delta_{hrd(\pi)}$ and $\Delta_{frt(\pi)}$ give, respectively, the variation in the number of cycles, hurdles and fortress of a permutation after applying each reversal.

3.3 Safe and unsafe reversals

If a breakpoint graph does not have unoriented components, it can be sorted with split reversals only. However, if we take no caution to select a split reversal, it may cause the production of new hurdles, which is an undesirable side effect (Figure 14 (A)). A split reversal that produces hurdles is called *unsafe reversal*, while a split reversal that does not produce hurdles is called *safe reversal* (Figure 14 (B)). Fortunately, it has been proven that, for any oriented component, there is always one safe reversal [53].

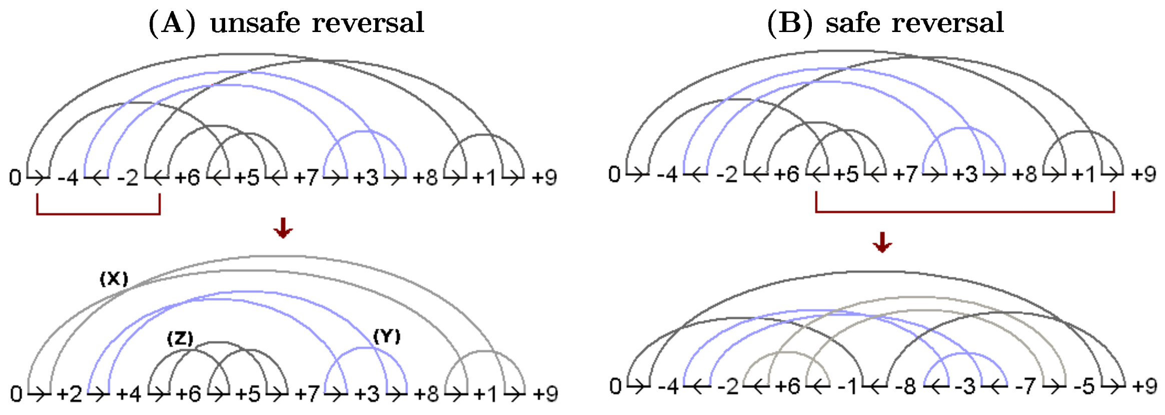


Figure 14: (A) An unsafe reversal breaks a cycle in two, but creates three unoriented component (X, Y and Z). (B) Alternatively, a safe reversal breaks a cycle in two without creating unoriented components.

Hurdles are very rare, and fortresses are even more rare in permutations that represent real genomes [9]. In practice, split reversals are sufficient to sort the majority of the permutations, and the main challenge is to find safe reversals. A simple way to do that is testing each split reversal to verify whether it is safe or not, until finding a safe reversal. However, there are faster ways to select a safe reversal, and one approach is based on another structure related to the breakpoint graph, that is called ***overlap graph*** (see more details in [37]).

3.4 Sorting a signed permutation

With the approach described in this chapter, we can obtain a procedure to sort a permutation π by reversals (Algorithm 1).

Algorithm 1 Sorting a signed permutation

```
Input: A signed permutation $\pi$
Output: An optimal sequence of reversals sorting $\pi$

  construct the breakpoint graph of $\pi$
  $s \leftarrow \epsilon$   [$s$ is an empty sequence in the beginning]
  if $frt(\pi) = 1$ then
     choose a reversal $\rho$ to eliminate the fortress
     $\pi \leftarrow \pi \circ \rho$
     $s \leftarrow s \cdot \rho$  [concatenates the reversal $\rho$ to $s$]
  end if
  while there is a pair of super-hurdles $X$ and $Y$ in $\pi$ do
     choose a merge reversal $\rho$ to eliminate $X$ and $Y$
     $\pi \leftarrow \pi \circ \rho$
     $s \leftarrow s \cdot \rho$  [concatenates the reversal $\rho$ to $s$]
  end while
  while there is a hurdle $Z$ in $\pi$ do
     choose a cut reversal $\rho$ to eliminate $Z$
     $\pi \leftarrow \pi \circ \rho$
     $s \leftarrow s \cdot \rho$  [concatenates the reversal $\rho$ to $s$]
  end while
  while $\pi$ is not sorted do
     choose a safe split reversal $\rho$ to $\pi$
     $\pi \leftarrow \pi \circ \rho$
     $s \leftarrow s \cdot \rho$  [concatenates the reversal $\rho$ to $s$]
  end while
  return $s$   [$s$ is an optimal sorting sequence for $\pi$]
```

The theoretical complexity of Algorithm 1 is $O(n^5)$, where n is the size of the input permutation [53]. Further studies improved this theoretical complexity and currently the fastest algorithm to find an optimal sorting sequence is subquadratic [11, 31, 63], while the reversal distance can be computed in $O(n)$ time [4].

3.5 The symmetry of sorting by reversals

For any sequence of reversals $s = \rho_1\rho_2 \ldots \rho_{d-1}\rho_d$ sorting a permutation π into a permutation π_T, we define the inverse of s as $inv(s) = \rho_d\rho_{d-1} \ldots \rho_2\rho_1$. Observe that the sequence $inv(s)$ sorts π_T into π, and, consequently, each optimal sequence sorting π into π_T has an equivalent optimal sequence sorting π_T into π. If we go back to Figure 6, in which the reversal distance between the two genomes is 4, for instance, we see that, while the optimal sequence $s = \{1, 2, 4\}\{1, 3, 4\}\{2, 3, 4\}\{3\}$ sorts genome A into genome B, the optimal sequence $inv(s) = \{3\}\{2, 3, 4\}\{1, 3, 4\}\{1, 2, 4\}$ sorts genome B into genome A. Due to this, the approach of sorting one genome into another by reversals is said to be symmetric.

3.6 Component-specific reversals

A merge reversal joins cycles, and, since it only appears in optimal sequences for eliminating hurdles or the fortress, it merges components. In contrast, a split or a neutral reversal is always internal to a component, and does not change the properties of the other components in the breakpoint graph.

Proposition 1 *Applying a split or neutral reversal ρ in a permutation π does not change the components of π that do not contain the extremities of ρ.*

Proof. The extremities of a split or neutral reversal ρ are in the same cycle, and consequently, in the same component of π. The other components of π are either completely within ρ, or completely external to ρ. If a component C is within ρ, all points in C are inversed, but the number of cycles, and the relative directions of the black edges in the cycles of C remain unchanged. If a component C is external to ρ, then C remains obviously unchanged. □

Due to this, if a permutation π can be sorted with split and neutral reversals only, then the components of π can be sorted independently. A breakpoint graph that does not contain super-hurdles can be sorted with split and neutral reversals only, thus the components of this kind of breakpoint graph can be sorted independently.

Sorting oriented components independently is a topic that has been studied in several works on the sorting by reversals problem (see for instance [11]).

Chapter 4

Traces of sorting sequences of reversals

Summary

4.1 The space of all optimal sorting sequences 30

4.1.1 The symmetry of the space of sorting sequences 30

4.1.2 An algorithm to enumerate all sorting sequences 31

4.2 Traces 33

4.2.1 The symmetry of traces 35

4.2.2 Normal form of a trace 35

4.2.3 Computing traces by enumerating all sorting sequences 36

4.3 An algorithm to directly enumerate the traces 38

4.3.1 The algorithm 38

4.3.2 Theoretical complexity 40

4.4 Component-specific reversals and trace composition 43

4.5 Implementation and performance 49

4.6 Final remarks 50

As we saw in the previous chapter, when duplications are not allowed, computing the reversal distance between two genomes and finding an optimal sequence of reversals that sort a genome into another can be solved in polynomial time [32, 33]. However, the number of optimal sorting sequences is usually huge. Considering the permutation $(-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$, for example, the number of sorting sequences is $8,278,540$, and it can be insufficient when attempting a biological interpretation to know only one among them. In order to select a potentially more meaningful solution, some studies find a sequence in which the reversals do not break the common intervals [26], which are the clusters of co-localised genes between the two considered genomes.

Alternatively to all the studies that give only one sequence among possibly many, Siepel [59] proposed an algorithm that allows the enumeration of all sorting sequences. However, as the list of all sorting sequences is usually huge, enumerating all is almost as useless as giving only one of them. Bergeron *et al.* [9] then observed that many sorting sequences are equivalent and might be grouped in equivalence classes. This approach can reduce considerably the universe of sequences to be handled. However, the authors did not provide an algorithm to enumerate the classes without enumerating all the sequences.

One important result of our work is an answer to this question. We provide an algorithm to directly enumerate all the classes of equivalent sorting sequences and to count the number of sequences in each class. Unfortunately, although the number of classes is much smaller than the number of sequences, it can still be too big to be interpreted. We implemented all the algorithms that we have developed integrated to BAOBABLUNA, a java framework to deal with permutations and reversals, that will be described in Chapter 6. Unfortunately, we verified that currently we are unable to compute traces for permutations with a reversal distance of about 20 or higher. Nevertheless, with BAOBABLUNA we run several experiments, showing the gain in the execution time when the traces are computed directly, with respect to the enumeration of all sorting sequences (all experiments were made on a 64 bit personal computer with two 3GHz CPUs and 2GB of RAM).

4.1 THE SPACE OF ALL OPTIMAL SORTING SEQUENCES

The space of all optimal sequences sorting a permutation π into a permutation π_T is defined as the set $S = \{ s \mid s \text{ is an optimal sequence sorting } \pi \text{ into } \pi_T \}$.

4.1.1 The symmetry of the space of sorting sequences

Since the sorting by reversals is a symmetric approach, the same is valid for the space of sorting sequences. Recall that, for any sequence of reversals $s = \rho_1\rho_2\ldots\rho_{d-1}\rho_d$ sorting a permutation π into a permutation π_T, we defined the inverse of s as $inv(s) = \rho_d\rho_{d-1}\ldots\rho_2\rho_1$. Observe that the sequence $inv(s)$ sorts π_T into π, and, consequently, each sequence sorting π into π_T has an equivalent sequence sorting π_T into π.

Let S be the set of all optimal sequences sorting π into π_T. We define the inverse of S as $inv(S) = \{ \; inv(s) \mid s \in S \; \}$.

Proposition 2 *The set S contains all optimal sequences sorting π into π_T, if, and only if, $inv(S)$ is the set of all optimal sequences sorting π_T into π.*

Proof. We saw that, for each sequence $s \in S$ sorting π into π_T, there is a sequence $inv(s) \in inv(S)$ that sorts π_T into π. By contradiction, we prove that $inv(S)$ contains all sequences that sort π_T into π. Otherwise, suppose a sequence $u \notin inv(S)$, such that u sorts π_T int π. Then we would have a sequence $inv(u)$ that sorts π into π_T and, since $u \notin inv(S)$, $inv(u) \notin S$. This is a contradiction, because S is assumed to contain all sequences sorting π into π_T.

The proof in the opposite direction is analogous. □

Due to Proposition 2, we can say that enumerating the optimal sequences that sort π into π_T is equivalent to enumerating the optimal sequences that sort π_T into π.

4.1.2 An algorithm to enumerate all sorting sequences

The space of all sorting sequences can be generated thanks to an algorithm proposed by Siepel [59]. Given a permutation π, the algorithm of Siepel [59] computes all optimal $1-$sequences for π. Considering the permutation $\pi = (-3, 2, 1, -4)$, for example, the possible $1-$sequences are $\{1\}$, $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{2\}$, $\{3\}$ and $\{4\}$. This algorithm has complexity $O(n^3)$ and results in $O(n^2)$ reversals, where n is the size of the input permutation.

With this algorithm, we can obtain the set of all optimal sorting sequences for a given permutation π. The sorting sequences can be constructed iteratively, so that, at a step i, for each optimal $(i-1)-$sequence s previously computed we run the algorithm of Siepel [59] to find all optimal $1-$sequences for $\pi \circ s$ and concatenate each returned $1-$sequence ρ to the sequence s, constructing an $i-$sequence $s\rho$. In other words, the set of $i-$sequences is computed from the set of optimal $(i-1)-$sequences by iterating the algorithm for finding all $1-$sequences. This iterative procedure is described in Algorithm 2.

Theoretical complexity of Algorithm 2. The algorithm of Siepel [59], that searchs all optimal $1-$sequences for a given permutation, results in $O(n^2)$ optimal $1-$sequences. For each step i,

Algorithm 2 Enumerating all optimal sorting sequences for a permutation

Input: A signed permutation π

Output: The set of all sequences of reversals sorting π

```
$d \leftarrow$ reversal distance of $\pi$
$\mathcal{R} \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $\pi\}$   [Siepel [59]]
$\mathcal{S} \leftarrow \mathcal{R}$
for each integer $i$ from 2 to $d$ do
  $\mathcal{S}' \leftarrow \emptyset$   [contains the $i-$sequences]
  for each $s$ in $\mathcal{S}$   [$s$ is an $(i-1)-$sequence]   do
    $\pi' \leftarrow \pi \circ s$   [apply the $(i-1)-$sequence $s$ to $\pi$]
    $\mathcal{R} \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $\pi'\}$   [Siepel [59]]
    for each reversal $\rho \in \mathcal{R}$ do
      $s' \leftarrow s \cdot \rho$   [concatenate $\rho$ at the end of sequence $s$]
      insert $s'$ in $\mathcal{S}'$   [$s'$ is an $i-$sequence]
    end for
  end for
  $\mathcal{S} \leftarrow \mathcal{S}'$
end for
return $\mathcal{S}$   [$\mathcal{S}$ is the final set of $d-$sequences]
```

the size of the set of optimal $i-$sequences is therefore at most $n^2 * n^2 * \ldots * n^2$ (i times), that is, $O(n^{2i})$. Since the algorithm of Siepel has complexity $O(n^3)$, we can enumerate the set of all optimal sorting sequences in time at most $n^3 * \sum_{k=1}^{d} n^{2k}$, where n is the size and d is the reversal distance of the input permutation (d is $O(n)$). In this way, Algorithm 2 has time complexity $O(n^{2n+3})$. □

If $\pi = (-3, 2, 1, -4)$, running Algorithm 2 for the permutation π will result in 28 optimal sorting sequences (Table 2).

01.	$\{1\}\{1,2,3\}\{2\}\{4\}$	**11.**	$\{1,2,3\}\{4\}\{1\}\{2\}$	**21.**	$\{4\}\{1,2,3\}\{1\}\{2\}$
02.	$\{1\}\{1,2,3\}\{4\}\{2\}$	**12.**	$\{1,2,3\}\{4\}\{2\}\{1\}$	**22.**	$\{4\}\{1,2,3\}\{2\}\{1\}$
03.	$\{1\}\{2\}\{1,2,3\}\{4\}$	**13.**	$\{2\}\{1\}\{1,2,3\}\{4\}$	**23.**	$\{4\}\{2\}\{1\}\{1,2,3\}$
04.	$\{1\}\{2\}\{4\}\{1,2,3\}$	**14.**	$\{2\}\{1\}\{4\}\{1,2,3\}$	**24.**	$\{4\}\{2\}\{1,2,3\}\{1\}$
05.	$\{1\}\{4\}\{1,2,3\}\{2\}$	**15.**	$\{2\}\{1,2,3\}\{1\}\{4\}$	**25.**	$\{1,2,4\}\{1,3,4\}\{2,3,4\}\{3\}$
06.	$\{1\}\{4\}\{2\}\{1,2,3\}$	**16.**	$\{2\}\{1,2,3\}\{4\}\{1\}$	**26.**	$\{1,2,4\}\{1,3,4\}\{3\}\{2,3,4\}$
07.	$\{1,2,3\}\{1\}\{2\}\{4\}$	**17.**	$\{2\}\{4\}\{1\}\{1,2,3\}$	**27.**	$\{1,2,4\}\{3\}\{1,3,4\}\{2,3,4\}$
08.	$\{1,2,3\}\{1\}\{4\}\{2\}$	**18.**	$\{2\}\{4\}\{1,2,3\}\{1\}$	**28.**	$\{3\}\{1,2,4\}\{1,3,4\}\{2,3,4\}$
09.	$\{1,2,3\}\{2\}\{1\}\{4\}$	**19.**	$\{4\}\{1\}\{1,2,3\}\{2\}$		
10.	$\{1,2,3\}\{2\}\{4\}\{1\}$	**20.**	$\{4\}\{1\}\{2\}\{1,2,3\}$		

Table 2: The 28 optimal sequences of reversals sorting $(-3, 2, 1, -4)$.

With this method we can explore the solution space of sorting a genome into another by reversals. Nevertheless the list of all sorting sequences is usually huge, thus enumerating all is almost as useless as giving only one of them. Table 3 shows, by several examples, how the number of sorting sequences may increase as the reversal distance between the considered genome increases.

Permutation (π)	$\mathbf{N}_M$	$d(\pi)$	$\mathbf{N}_S$
$\pi_A = (-3, 2, 1, -4)$	4	4	28
$\pi_B = (-4, 1, -3, 6, -7, -5, 2)$	7	6	204
$\pi_C = (-6, 5, 7, -1, -4, 3, 2)$	7	6	496
$\pi_D = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$	12	8	31,752
$\pi_E = (-4, 3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$	12	9	407,232
$\pi_F = (-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$	13	10	8,278,540
$\pi_G = (-12, 11, -10, -1, 16, -4, -3, 15, -14, 9, -8, -7, -2, -13, 5, -6)$	16	12	505,634,256
$\pi_H = (-12, 11, -10, 6, -5, 13, 2, 7, 8, -9, 14, -15, 3, 4, -16, 1)$	16	13	40,313,272,766

Table 3: Examples of permutations, their sizes, their reversal distances and their number of optimal sorting sequences.

4.2 Traces

Bergeron *et al.* [9] observed that many sorting sequences are equivalent and may be grouped in equivalence classes. Intuitively, all optimal sorting sequences in the same equivalence class are composed by the same reversals, but applied in different orders. The 28 sequences that sort the permutation $\pi_A = (-3, 2, 1, -4)$, for example, can be grouped in two classes of equivalence, one with 24 and the other with 4 sequences (Table 4).

Class 1:

01.	$\{1\}\{1,2,3\}\{2\}\{4\}$	**09.**	$\{1,2,3\}\{2\}\{1\}\{4\}$	**17.**	$\{2\}\{4\}\{1\}\{1,2,3\}$
02.	$\{1\}\{1,2,3\}\{4\}\{2\}$	**10.**	$\{1,2,3\}\{2\}\{4\}\{1\}$	**18.**	$\{2\}\{4\}\{1,2,3\}\{1\}$
03.	$\{1\}\{2\}\{1,2,3\}\{4\}$	**11.**	$\{1,2,3\}\{4\}\{1\}\{2\}$	**19.**	$\{4\}\{1\}\{1,2,3\}\{2\}$
04.	$\{1\}\{2\}\{4\}\{1,2,3\}$	**12.**	$\{1,2,3\}\{4\}\{2\}\{1\}$	**20.**	$\{4\}\{1\}\{2\}\{1,2,3\}$
05.	$\{1\}\{4\}\{1,2,3\}\{2\}$	**13.**	$\{2\}\{1\}\{1,2,3\}\{4\}$	**21.**	$\{4\}\{1,2,3\}\{1\}\{2\}$
06.	$\{1\}\{4\}\{2\}\{1,2,3\}$	**14.**	$\{2\}\{1\}\{4\}\{1,2,3\}$	**22.**	$\{4\}\{1,2,3\}\{2\}\{1\}$
07.	$\{1,2,3\}\{1\}\{2\}\{4\}$	**15.**	$\{2\}\{1,2,3\}\{1\}\{4\}$	**23.**	$\{4\}\{2\}\{1\}\{1,2,3\}$
08.	$\{1,2,3\}\{1\}\{4\}\{2\}$	**16.**	$\{2\}\{1,2,3\}\{4\}\{1\}$	**24.**	$\{4\}\{2\}\{1,2,3\}\{1\}$

Class 2:

01.	$\{1,2,4\}\{1,3,4\}\{2,3,4\}\{3\}$	**03.**	$\{1,2,4\}\{3\}\{1,3,4\}\{2,3,4\}$
02.	$\{1,2,4\}\{1,3,4\}\{3\}\{2,3,4\}$	**04.**	$\{3\}\{1,2,4\}\{1,3,4\}\{2,3,4\}$

Table 4: The two equivalence classes of optimal sequences of reversals sorting $(-3, 2, 1, -4)$.

To formalize the equivalence relation of these classes, we need to introduce the concept of *commutation.* Two intervals (or reversals) are said to *overlap* if they intersect but none is contained in the other. For example, in the permutation $(-3, 2, 1, -4)$, the intervals $\{2, 3\}$ and $\{1, 2, 4\}$ overlap, while $\{2,3\}$ and $\{1,2,3\}$ do not. Let $s = \rho_1\rho_2 \dots \rho_{i-1}\rho_i\rho_{i+1}\rho_{i+2} \dots \rho_d$ be a valid sequence of reversals for a permutation π, and ρ_i and ρ_{i+1} be two non-overlapping reversals that appear consecutively in s. As ρ_i and ρ_{i+1} do not overlap, then ρ_{i+1} is an interval of $\pi \circ \rho_1\rho_2 \dots \rho_{i-1}$ and ρ_i is an interval of $\pi \circ \rho_1\rho_2 \dots \rho_{i-1}\rho_{i+1}$, that is, the sequence $s' = \rho_1\rho_2 \dots \rho_{i-1}\rho_{i+1}\rho_i\rho_{i+2} \dots \rho_d$, which is obtained replacing $\rho_i\rho_{i+1}$ by $\rho_{i+1}\rho_i$ in s, is also a valid sequence of reversals for π. The operation of inverting the order of two consecutive non-overlapping reversals ρ_i and ρ_{i+1} in a sequence of reversals s is called *commutation* of ρ_i and ρ_{i+1}.

Two sequences are said to be *equivalent* if one can be obtained from another by a sequence of commutations of non-overlapping reversals. An equivalence class of optimal sequences of reversals under this equivalence relation is called *trace*. It is easy to see that all the sequences in a trace have the same number of reversals. We denote then by $i-$*trace* a trace of $i-$sequences.

The concept of traces is well studied in combinatorics, see for example [25]. It is particularly relevant in our study because of a result proven in [9], that states that the set of all optimal sequences of reversals sorting a signed permutation is a union of traces. As a consequence, if the set of sorting sequences is too big to be enumerated, the set of traces may be a more relevant result for the problem of sorting by reversals.

Observe that, for a given permutation, all sequences in the same trace are composed by the same reversals, but not every pair of sorting sequences composed by the same reversals are in the same trace. If we take the permutation $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$, for instance, we find two sequences (see Table 5) that sort π and are composed by the same reversals. However, one cannot be obtained by a sequence of commutations of non-overlapping reversals over the other, thus they are not in the same trace.

$\{1,2\}\{7\}\{8,10\}\{1,5,\dots,11\}\{8,9\}\{1,3,4,12\}\{2,\dots,12\}\{3,\dots,11\}$
$\{2,\dots,12\}\{7\}\{8,10\}\{1,3,4,12\}\{8,9\}\{1,5,\dots,11\}\{1,2\}\{3,\dots,11\}$

Table 5: Non-equivalent sequences composed by the same reversals.

4.2.1 The symmetry of traces

For a given trace T of optimal sequences of reversals sorting π into π_T, we define the inverse of T as $inv(T) = \{\ inv(s) \mid s \in T\ \}$.

Proposition 3 *The set T is a trace of optimal sequences of reversals sorting π into π_T, if, and only if, $inv(T)$ is a trace of optimal sequences of reversals sorting π_T into π.*

Proof. We saw that each s is a sequence sorting π into π_T, thus, $inv(s)$ is a sequence sorting π_T into π. For each pair of sequences $s, s' \in T$, we always obtain a pair $inv(s), inv(s')$ of equivalent sequences under the commutation of non-overlapping reversals (each pair of non-overlapping reversals ρ and θ in s and s' has a corresponding pair of non-overlapping reversals θ and ρ in $inv(s)$ and $inv(s')$), thus all sequences in $inv(T)$ are equivalent. By contradiction, we prove also that $inv(T)$ is a trace, that is, for a given sequence $u \in inv(T)$, all sequences that sort π_T into π and are equivalent to u are in $inv(T)$. Otherwise, suppose a sequence v that is equivalent to u, such that v is not in $inv(T)$. Then we would have a sequence $inv(v)$ that is equivalent to $inv(u)$, and, while $inv(u)$ is in T, $inv(v)$ is not. This is a contradiction: since T is assumed to be a trace, it contains all sequences that are equivalent under the commutation of non-overlapping reversals.

The proof in the opposite direction is analogous. $\square$

Let $\mathcal{T} = \{\ T \mid T$ is a trace of optimal sequences sorting π into $\pi_T\ \}$ be the set of all traces sorting π into π_T. We also define the inverse of $\mathcal{T}$ as $inv(\mathcal{T}) = \{\ inv(T) \mid T \in \mathcal{T}\ \}$.

As a consequence of Proposition 3, we can affirm that $\mathcal{T}$ is the set of all traces sorting π into π_T if, and only if, $inv(\mathcal{T})$ is the set of all traces sorting π_T into π. Thus, computing the traces from π to π_T is equivalent to computing the traces from π_T to π.

4.2.2 Normal form of a trace

A sequence s of a trace T is said to be in *normal form* if it can be decomposed into substrings[6] $s = u_1 < \cdots < u_m$[7] such that:

[6] The "substrings" are all *contiguous* subsets of the sequence of reversals.

[7] In the original notation the normal form is $s = u_1|\ldots|u_m$, but we prefer to use the symbol '<' instead of '|'.

- every pair of reversals of a substring u_i is non-overlapping;
- for every reversal ρ of a substring u_i $(i > 1)$, there is at least one reversal θ of the substring u_{i-1} such that ρ and θ overlap;
- every substring u_i is increasing according to the lexicographic order.

A theorem by Cartier and Foata [23] (cited in [9]) states that, for any trace, there is a unique element that is in normal form. We may therefore represent a trace by its element in normal form. The two traces of sequences sorting the permutation $\pi = (-3, 2, 1, -4)$, described in Table 4, for example, can be represented by the corresponding normal forms $\{1\}\{1, 2, 3\}\{2\}\{4\}$ and $\{1, 2, 4\}\{3\} < \{1, 3, 4\} < \{2, 3, 4\}$.

Given an optimal sorting sequence $s = \rho_1\rho_2 \ldots \rho_d$ for a permutation π with reversal distance d, the normal form of the trace T that contains s is constructed by iteratively adding the elements ρ_i, $1 \leq i \leq n$, to the normal form f of the $(i-1)-$trace containing the sequence $\rho_1 \ldots \rho_{i-1}$. This adding procedure is represented by $f + \rho_i$ and described by Algorithm 3.

Algorithm 3 Adding an element ρ_i to a normal form f of an $(i-1)-$trace: $f + \rho_i$

```
Input: The normal form $f = u_1 < u_2 < \cdots < u_k$ of an $(i-1)-$trace and the next element $\rho_i$
Output: The normal form of the $i-$trace containing the sequence $u_1u_2 \ldots u_k\rho_i$

Let $j$ be the maximum index such that $u_j$ contains an element that overlaps with $\rho_i$, or $0$ if such a $u_j$
does not exist
if $j = k$ then
   Add a new substring $u_{k+1} \leftarrow \rho_i$
else
   Add $\rho_i$ to the substring $u_{j+1}$, according to the lexicographic order
end if
```

Theoretical complexity of Algorithm 3. Since the reversal distance and the interval sizes are bounded by n, the procedure has complexity $O(n^2 \log n)$, considering that the elements of each reversal have to be sorted, which takes $O(n \log n)$, and that comparing reversals may be done in $O(n)$. □

4.2.3 Computing traces by enumerating all sorting sequences

An algorithm to enumerate all the traces can be derived from the algorithm that enumerates all the optimal sorting sequences. For each sequence, we may simply compute the associated trace and add it to the list of found traces if it is not already in it (Algorithm 4).

Algorithm 4 Enumerating all optimal sorting sequences and computing traces

Input: A signed permutation π

Output: The normal form and counter (f, c) of each trace of optimal sequences of reversals sorting π

```
$d \leftarrow$ reversal distance of $\pi$
$\mathcal{S} \leftarrow$ all optimal sorting $d-$sequences for $\pi$   [Algorithm 2]
$\mathcal{T} \leftarrow \emptyset$   [contains the $d-$traces]
for each $s$ in $\mathcal{S}$   [$s$ is a sorting $d-$sequence]   do
  $f_s \leftarrow \epsilon$   [to construct the normal form of $s$]
  for each $\rho_j$ in $s = \rho_1\rho_2\dots\rho_d$ do
    $f_s \leftarrow f_s + \rho_j$   [Algorithm 3]
  end for
  if there is $(f, c) \in \mathcal{T}$ such that $f_s = f$ then
    $c \leftarrow c + 1$   [update the counter of the $d-$trace repr. by $f$]
  else
    insert $(f_s, 1)$ in $\mathcal{T}$   [$(f_s, 1)$ represent a $d-$trace]
  end if
end for
return $\mathcal{T}$   [$\mathcal{T}$ is the final set of $d-$traces]
```

Theoretical complexity of Algorithm 4. The set of all optimal $d-$sequences can be computed with Algorithm 2 that has complexity $O(n^{2n+3})$, where n is the size of the permutation. There remains to construct the normal form of each trace and to group the sorting sequences by trace.

The normal form of each sorting sequence is compared to a list of previously constructed normal forms of traces, so that the trace counter is incremented if it exists already; otherwise the trace is inserted in the list with counter equal to 1. This may take $O(n^2 \log N)$ operations, where $O(n^2)$ is the size of a trace and N is the number of found traces. As N is bounded by the number of sorting sequences, we have $n^2 \log N \leq n^2 \log(n^{2n}) = 2n^3 \log n$.

Eventually, the total time complexity for enumerating all the normal forms of the traces is bounded by $O(n^{2n+3}) + O(n^{2n}(n^2 \log n + 2n^3 \log n)) = O(n^{2n+3} \log n)$. □

The upper bound on the theoretical complexity of Algorithm 4 does not give hope that this method can be applied to big permutations. We shall actually see in practice that it is intractable for permutations π above around $d(\pi) = 10$.

4.3 An algorithm to directly enumerate the traces

Bergeron *et al.* [9] provided no algorithmic insight for enumerating directly the traces, without enumerating all sorting sequences. The authors stated as an open problem the complexity of giving one element in each trace. One of the main contributions in our work is a solution to this problem, that is, an algorithm that enumerates the normal forms of all the traces of optimal sorting sequences given a signed permutation, and counts the number of sequences in each trace, without enumerating all the sequences.

The idea of the algorithm to enumerate the traces is almost naturally contained in the following notions. First we noticed that for any integer k from 1 to $d(\pi)$, if s and s' are two equivalent optimal $k-$sequences for π, that is, s can be obtained from s' by a sequence of commutations of non-overlapping reversals, then $\pi \circ s = \pi \circ s'$. The equivalence class that contains the sequences s and s' is called a *$k-$trace*. Observe that all sequences in an $k-$trace t are prefixes of at least one trace T of optimal sorting sequences for π. Thus, t is said to be a *$k-$prefix* of T. In other words, we can say that a $k-$trace t is a $k-$prefix of an $i-$trace T $(k \leq i)$ if and only if each $k-$sequence of t is a $k-$prefix of at least one $i-$sequence of T. In addition, we observed that the number of sequences in an $i-$trace is the sum of the number of sequences in its $(i-1)-$prefixes (see Figure 15). Instead of enumerating incrementally all the $i-$sequences and then computing and comparing the traces, it is therefore more valuable to enumerate incrementally and compare directly all the $i-$traces.

4.3.1 The algorithm

We may construct all $i-$traces simultaneously in an incremental way, without generating all the sorting sequences. With no additive cost, we also compute the number of sequences in each $i-$trace. The method is detailed in Algorithm 5.

Theorem 1 *At the end of Algorithm 5, $\mathcal{T}$ contains, for every trace T of optimal sequences for sorting π, one element of T (the normal form) and the number of sequences in T.*

Proof. The proof is by induction. We prove that at the end of step i of the main loop of Algorithm 5, the set $\mathcal{T}$ contains all the normal forms and the size of the $i-$traces of optimal

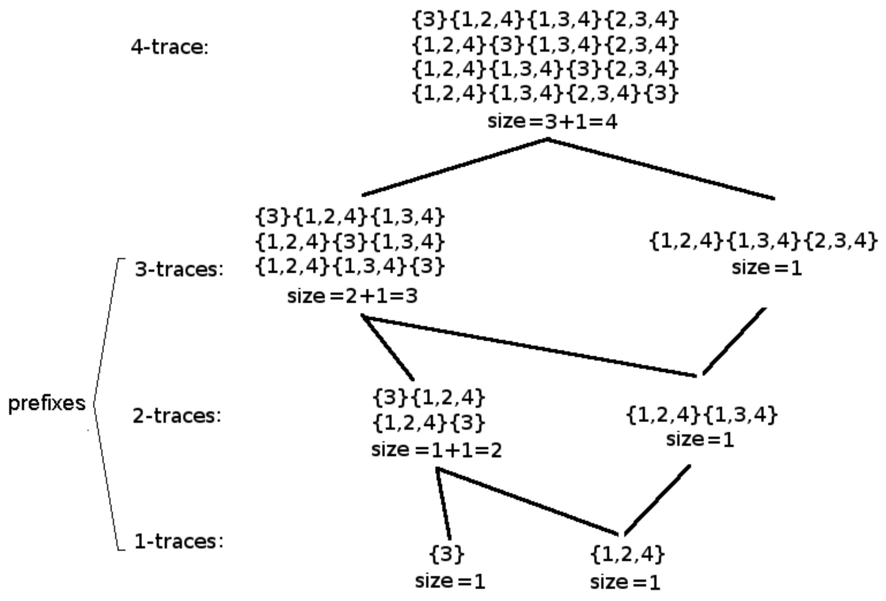


Figure 15: Decomposing a 4-trace in its prefixes.

sequences for π.

For $i = 1$, each $1-$trace is generated by running the algorithm of Siepel [59] over π and the size of a $1-$trace is 1.

For an arbitrary $2 \leq i \leq d(\pi)$, by hypothesis, $\mathcal{T}$ contains all the normal forms and the size of the optimal $(i-1)-$traces. Every $i-$trace has a prefix in this set, since a prefix of size $i-1$ of an optimal $i-$sequence is an optimal $(i-1)-$sequence. So every $i-$trace is found from an $(i-1)-$trace by adding a $1-$sequence.

Now it remains to prove that the cardinality of an $i-$trace T is the sum of the cardinalities of its $(i-1)-$prefixes, so that the right size of all traces is computed. Let $\rho_1, \ldots, \rho_k$ be the reversals that are in the last position of at least one sequence in T. Let x_j be the number of elements of T which have ρ_j as their last position. Then the number of sequences of T is $\sum_j x_j$. Now, for all j, since ρ_j is the last reversal of an optimal $i-$sequence $x_1 \ldots x_{i-1}\rho_j$ of T, $x_1 \ldots x_{i-1}$ is an optimal $(i-1)-$sequence of reversals, so it belongs to an $(i-1)-$trace T' of size x_j. By the induction hypothesis, the size of the trace T is therefore the sum of the sizes of all $(i-1)-$prefixes of T, and the algorithm provides this size, since it generates all prefixes. $\square$

Algorithm 5 Enumerating all the traces of a signed permutation

Input: A signed permutation π
Output: The normal form and counter (f, c) of each trace of optimal sequences of reversals sorting π

```
$d \leftarrow$ reversal distance of $\pi$
$\mathcal{T} \leftarrow \emptyset$
$S \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $\pi\}$   [Siepel [59]]
for each reversal $\rho \in S$ do
  insert $(\rho, 1)$ in $\mathcal{T}$   [each first reversal is a $1-$trace]
end for
for each integer $i$ from 2 to $d$ do
  $\mathcal{T}' \leftarrow \emptyset$   [contains the normal forms/sizes of all the $i-$traces]
  for each $(f, c)$ in $\mathcal{T}$   [$(f, c)$ represents an $(i-1)-$trace]   do
    $\pi_f \leftarrow \pi \circ f$   [apply the $(i-1)-$sequence $f$ to $\pi$]
    $S \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $\pi_f\}$   [Siepel [59]]
    for each reversal $\rho \in S$ do
      $f_\rho \leftarrow f + \rho$   [Algorithm 3]
      if there is $(f', c') \in \mathcal{T}'$ such that $f' = f_\rho$ then
        $c' \leftarrow c' + s$   [upd. the counter of the $i-$trace repr. by $f'$]
      else
        insert $(f_\rho, c)$ in $\mathcal{T}'$   [$(f_\rho, c)$ represent an $i-$trace]
      end if
    end for
  end for
  $\mathcal{T} \leftarrow \mathcal{T}'$
end for
return $\mathcal{T}$   [$\mathcal{T}$ is the final set of $d-$traces]
```

Figure 16 illustrates an execution of our algorithm for the permutation $\pi = (-3, 2, 1, -4)$.

4.3.2 Theoretical complexity

As we saw, the algorithm basically repeats the same procedure for each prefix of a final trace (see Figure 17). To each prefix we first apply the reversals of its normal form to the initial permutation (this takes $O(n^2)$). Then we compute the next $1-$sequences thanks to the algorithm of Siepel [59], that has complexity of $O(n^3)$ and may return $O(n^2)$ $1-$sequences. Each one of these returned $1-$sequences is added to the previous prefix in $O(n^2)$. Thus, each prefix is processed in $n^2 + n^3 + n^2 * n^2$, that is $O(n^4)$.

Consequently, the complexity of Algorithm 5 depends on the total number of prefixes, that is given by $\sum_{i=1}^{d(\pi)} trc(i)$, where $trc(i)$ is the number of $i-$traces of optimal $i-$sequences. To give an estimation of the number of prefixes of a trace, we need to adopt a representation of the traces as partially ordered sets (posets). It is possible to represent a trace T that contains an optimal sequence $\rho_1 \dots \rho_d$ by a partial ordering of the set $\mathcal{P}_T = \{(\rho_i, k_i)\}_i$, $1 \leq i \leq d$, where k_i

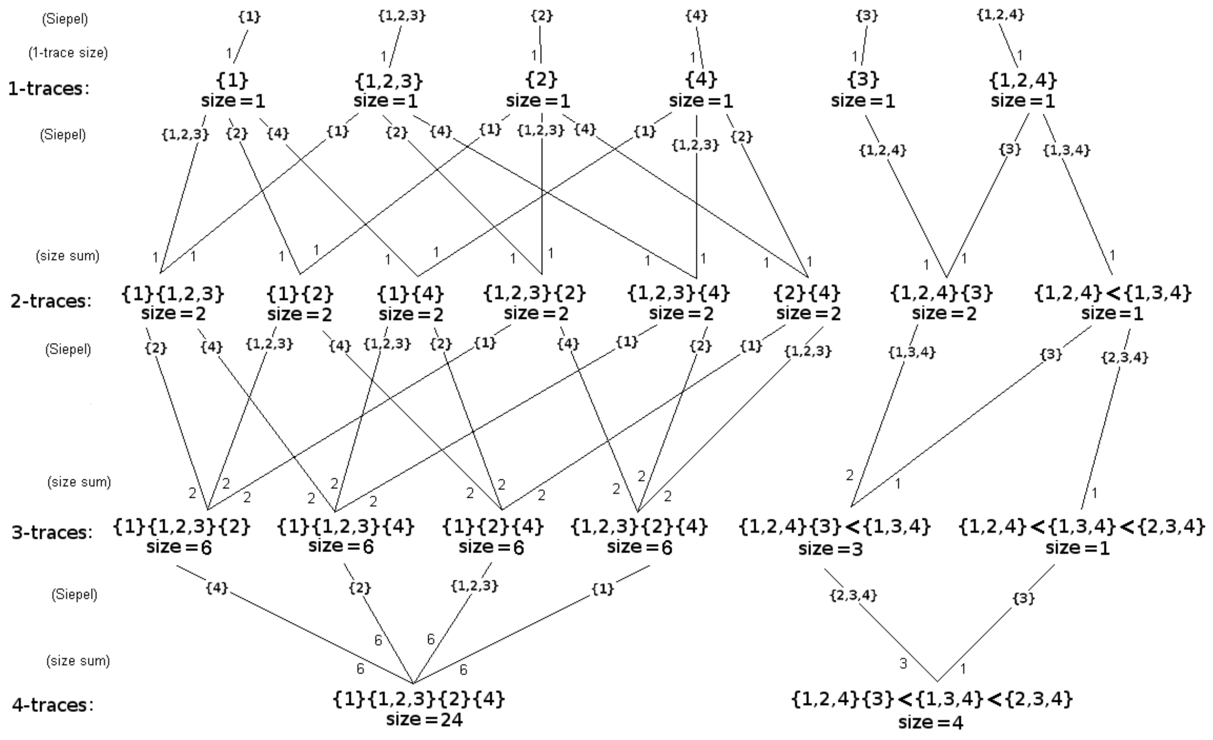


Figure 16: Constructing all the traces for the permutation $\pi = (-3, 2, 1, -4)$. In this example, the set of prefixes of the final trace $\{1\}\{1,2,3\}\{2\}\{4\}$ is disjoint of the set the prefixes of the final trace $\{1,2,4\}\{3\} < \{1,3,4\} < \{2,3,4\}$. This does not correspond to the general case (final traces usually share prefixes).

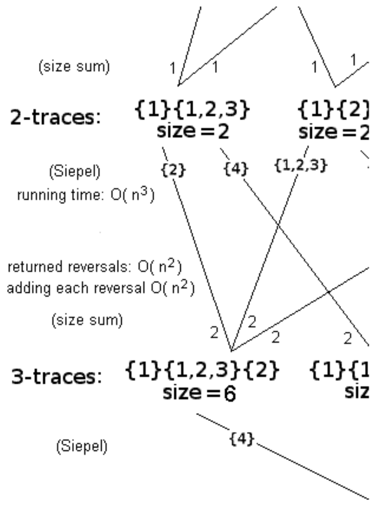


Figure 17: Processing a prefix of a trace when computing traces. To each prefix we first apply the reversals of its normal form to the initial permutation. Then we compute the next 1−sequences thanks to the algorithm of Siepel [59], that has complexity of $O(n^3)$ and may return $O(n^2)$ 1−sequences. Each one of these returned 1−sequences is added to the previous prefix in $O(n^2)$. Thus, each prefix is processed in $O(n^4)$.

is the number of occurrences of the reversal ρ_i in $\rho_1 \dots \rho_d$. The relation $<_T$ is defined as the transitive closure of the relation $\lhd$, itself defined by $(\rho_i, k_i) \lhd (\rho_j, k_j)$ if and only if $i < j$ and ρ_i and ρ_j overlap. In other words, $(\rho_i, k_i) <_T (\rho_j, k_j)$ if and only if the k_i^{th} ρ_i is always before the k_j^{th} ρ_j in the elements of T (see [25]). For example, $T = \{1,2,4\}\{3\} < \{1,3,4\} < \{2,3,4\}$ is a trace of optimal sorting sequences of reversals for the permutation $(-3,2,1,-4)$. The elements of $\mathcal{P}_T$ are $(\{1,2,4\},1)$, $(\{3\},1)$, $(\{1,3,4\},1)$ and $(\{2,3,4\},1)$, and the relations are $(\{1,2,4\},1) <_T (\{1,3,4\},1)$, $(\{1,3,4\},1) <_T (\{2,3,4\},1)$ and $(\{1,2,4\},1) <_T (\{2,3,4\},1)$.

The set $\mathcal{P}_T$ with the relation $<_T$ is a partially ordered set (poset). A *linear extension* of a poset is a total order $<_{tot}$ which satisfies $\rho <_T \theta \Rightarrow \rho <_{tot} \theta$. The set of all linear extensions of $(\mathcal{P}_T, <_T)$ is exactly the set of elements of the trace T (see [25]). We may therefore identify the trace T and the poset $(\mathcal{P}_T, <_T)$, and simply speak about the poset T.

The *height* of a trace T (or poset $\mathcal{P}_T$) is the cardinality of the maximum set of elements of $\mathcal{P}_T$ that is totally ordered by the relation $<_T$. It is also the number of sub-sequences u_i in the normal form of the trace T.

The *width* of a trace T (or poset $\mathcal{P}_T$) is a maximum cardinality set of elements of $\mathcal{P}_T$ that are not comparable by the relation $<_T$. It correponds to the largest subset of reversals of T in which every pair of reversals commute and is at least (but in general not equal to) the maximum size of a sub-sequence u_i in the normal form of the trace T. The width of a poset can be computed in polynomial time thanks to a reduction of Fulkerson [28] to a bipartite matching problem.

The representation of a trace as a poset allows us to use the parameters of the poset in the computations of the complexity of the algorithms, and it is also a nice way to present the solution of sorting by reversals. Indeed, a poset corresponds to a set of reversals that may have occurred during evolution and that could therefore help explain the difference between the organisation of two genomes. It indicates what we know and what we do not know about the order in which these potential reversals occurred. Instead of giving a list of sequences, or a unique sequence representing an equivalence class, the poset therefore gives *one* possible sorting sequence, with uncertainties as concerns the exact shape of the sequence.

An *ideal* of a poset $(\mathcal{P}_T, <_T)$ is a subset U of $\mathcal{P}_T$ such that if $\rho \in U$ and $\theta <_T \rho$, then $\theta \in U$. It is very easy to see that ideals of posets and prefixes of traces correspond to the same notions, and that in particular, the number of prefixes of a trace T is exactly the number of ideals of the poset $(\mathcal{P}_T, <_T)$. The advantage of this notation is that the number of ideals of a poset can be estimated. It is bounded by n^k, where n is the size of $\mathcal{P}_T$ and k is the width of the poset [61].

Theoretical complexity of Algorithm 5. The number of $i-$traces that we generate is therefore bounded by $Nn^{k_{max}}$, where N is the number of $d-$traces and k_{max} is the maximum width of a $d-$trace. Given this estimation, we may give a bound to the complexity of our algorithm. Indeed, for every $i-$trace, $1 \leq i \leq d-1$, we apply an $O(n^3)$ algorithm to find all the $1-$sequences. For all these $1-$sequences (there are at most n^2 of them), we then apply Algorithm 3 to construct the normal form of the $(i+1)-$trace, and compare the constructed normal form to the current list of normal forms of $(i+1)-$traces. This gives a final complexity of $O(Nn^{k_{max}}(n^3 + n^2(n^2 + n\log(Nn^{k_{max}})))) = O(Nn^{k_{max}+4})$. □

Observe that computing the number of linear extensions of a poset is $\#P-$complete [19], and the best known algorithms run in $O(n^k)$, where n is the size of the poset and k is its width [62]. Our algorithm counts the number of elements in each $d-$trace, that is the number of linear extensions of the associated posets. Our time complexity thus nearly reaches the best known complexity for the counting part.

The posets corresponding to the traces $\{1\}\{1,2,3\}\{2\}\{4\}$ and $\{1,2,4\}\{3\} < \{1,3,4\} < \{2,3,4\}$, that represent all optimal sequences sorting $\pi = (-3,2,1,-4)$, have widths 4 and 2 respectively. In figure 16 we can observe that the number of prefixes of trace $\{1\}\{1,2,3\}\{2\}\{4\}$, that has the higher width, is effectively higher than the number of prefixes of trace $\{1,2,4\}\{3\} < \{1,3,4\} < \{2,3,4\}$.

If in general the width of a poset may be as large as its number of elements, we have made some experiments on simulated permutations (see Figure 18) which show that, in practice, this parameter is often lower, which explains the speed-up of our algorithm compared to a total enumeration procedure.

4.4 Component-specific reversals and trace composition

If a permutation π has two or more unoriented components, the space of sorting sequences for π may contain sequences that have at least one merge reversal, to merge two unoriented

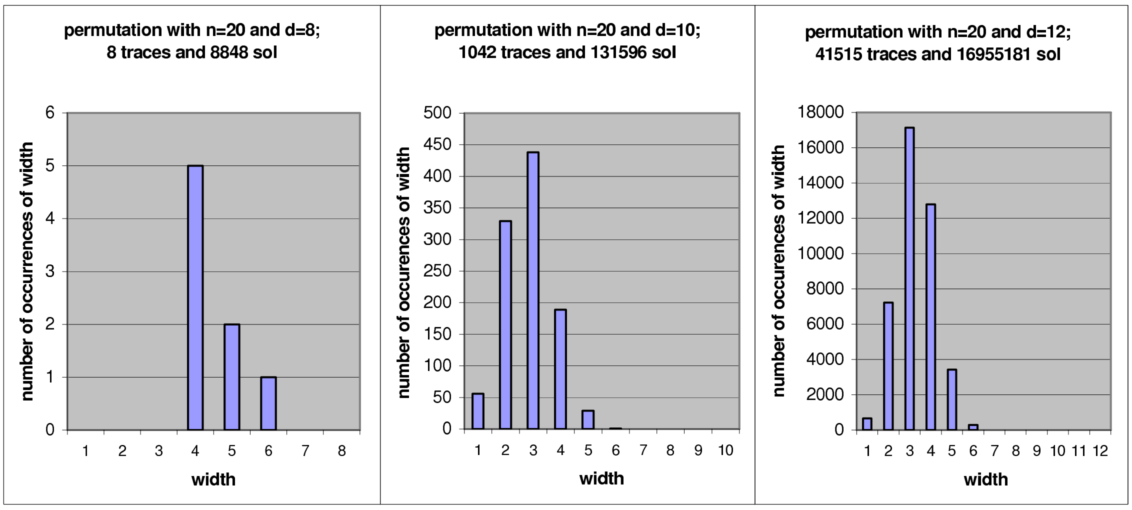


Figure 18: Distribution of the width of posets for 3 *random permutations of size* 20 *and different reversal distances* ($d = 8$, $d = 10$ *and* $d = 12$)*. For each permutation, we computed the traces of optimal sequences and we calculated the number of occurrences of each width in the traces.*

components of π. However, when a permutation π has at most one unoriented component, no optimal sorting sequence contains a merge reversal.

Proposition 4 *If a permutation π has at most one unoriented component, no optimal sequence sorting π contains a merge reversal.*

Proof. Recall that the formula for the reversal distance is $d(\pi) = pts(\pi) - cyc(\pi) + hrd(\pi) + frt(\pi)$. Since a merge reversal joins two cycles, it decreases $cyc(\pi)$ by one. However, when a merge reversal is used to eliminate a fortress, it decreases $hrd(\pi)$ by one and $frt(\pi)$ by one. A merge reversal can also be used to merge two hurdles, and in this case it decreases $hrd(\pi)$ by two. If the permutation π has at most one unoriented component, $frt(\pi) = 0$ and $hrd(\pi) \leq 1$. Thus a merge reversal applied to π can not decrease $frt(\pi)$ and decreases $hrd(\pi)$ at most by one. Consequentely, if π has at most one unoriented component, no optimal sequence sorting π contains merge reversals. $\square$

Thus, if a permutation π has c non-trivial components and at least $c-1$ oriented components, each optimal sequence s sorting π has only split and neutral reversals. Since each split or neutral reversal affects one single component of π (Proposition 1 in Chapter 3), each optimal sequence

s sorting π can be partitioned in c subsequences[8], such that the length of s is the sum of the lengths of its c subsequences and each subsequence contains only reversals that are internal to one non-trivial component of π.

Proposition 5 *Let π be a permutation with c non-trivial components and at most one unoriented component. Each optimal sequence s sorting π can be partitioned in c subsequences s_1, s_2, $\ldots$, s_c, where $|s| = |s_1| + |s_2| + \ldots + |s_c|$ and each s_i contains only reversals that are internal to the i^{th} non-trivial component of π.*

Proof. Let $s = \rho_1\rho_2 \ldots \rho_d$ be an optimal sequence of reversals sorting π, with $d = d(\pi)$. The permutation π has at most one unoriented component, thus s has only split or neutral reversals (Proposition 4). Since each split or neutral reversal affects one single component of π (Proposition 1 in Chapter 3), we can construct the component-specific subsequences s_1, s_2, $\ldots$, s_c as follows. In the beginning, each s_i is an empty sequence. Take each reversal ρ_k of s (k from 1 to d) and concatenate ρ_k to the sequence s_i, corresponding to the i^{th} component of π, that is affected by ρ_k. Each ρ_k is concatenated to exactly one subsequence s_i, thus $|s| = |s_1| + |s_2| + \ldots + |s_c|$ and each s_i contains only reversals that are internal to the i^{th} non-trivial component of π. □

Observe that each reversal in a subsequence s_i that sorts the i^{th} component commutes with each reversal in a subsequence s_j, that sorts the j^{th} component, thus we say that the subsequences s_i and s_j *commute*.

For example, the permutation $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$ has two non-trivial oriented components, as we can see in Figure 19. We call C_1 the first oriented component, which contains the points (black edges) that are before -4, after -1 and between the values $(-3, 12)$, $(12, -11)$, $(-5, 2)$, $(2, -1)$. The second oriented component is thus called C_2 and contains the points that are between the values $(-11, -8)$, $(-8, 10)$, $(10, 9)$, $(9, 7)$ and $(7, -6)$. The points between the values $(-4, -3)$ and $(-6, -5)$ are adjacencies. One optimal sorting sequence for π is $s = \{1, 2\}\{8, 10\}\{2\}\{12\}\{7\}\{1, 2, 5, 6, \ldots, 11, 12\}\{1, 2, 3, 4\}\{8, 9\}$, that can be partitioned in two subsequences $s_1 = \{1, 2\}\{2\}\{12\}\{1, 2, 5, 6, \ldots, 11, 12\}\{1, 2, 3, 4\}$, that sorts

[8]A "subsequence" of a sequence s is obtained by eliminating some of the elements (here reversals) of s while preserving the order of the remaining elements.

C_1, and $s_2 = \{8,10\}\{7\}\{8,9\}$, that sorts C_2. It is easy to see that s_1 commutes with s_2. The set of all traces sorting π is represented in Table 6.

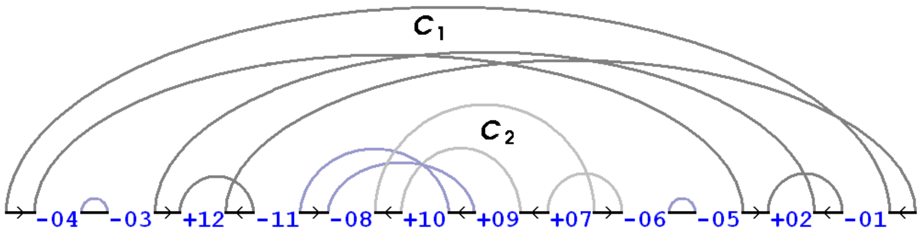


Figure 19: The breakpoint graph of the permutation $(-4,-3,12,-11,-8,10,9,7,-6,-5,2,-1)$ *with two oriented components.*

Trace	Trace normal form	# seq.
1.	$f = \{1,2\}\{1,2,5,\ldots,12\}\{2\}\{7\}\{8,10\}\{12\} < \{1,2,3,4\}\{8,9\}$	10,080
2.	$f = \{1,\ldots,12\}\{2\}\{3,4,12\}\{5,\ldots,11\}\{7\}\{8,10\} < \{3,\ldots,11\}\{8,9\}$	10,080
3.	$f = \{1,\ldots,12\}\{2,\ldots,12\}\{2,5,\ldots,12\}\{7\}\{8,10\}\{12\} < \{2,3,4\}\{8,9\}$	10,080
4.	$f = \{1,2\}\{7\}\{8,10\} < \{1,5,\ldots,11\}\{8,9\} < \{1,3,4,12\} < \{2,\ldots,12\}\{3,\ldots,11\}$	336
5.	$f = \{2,\ldots,12\}\{7\}\{8,10\} < \{1,3,4,12\}\{8,9\} < \{1,5,\ldots,11\} < \{1,2\}\{3,\ldots,11\}$	336
6.	$f = \{2,5,\ldots,12\}\{5,\ldots,11\}\{7\}\{8,10\} < \{1,12\}\{8,9\} < \{1,5,\ldots,11\} < \{1,2,3,4\}$	840
	Total	31,752

Table 6: The 31,752 *optimal sequences of reversals for sorting* $(-4,-3,12,-11,-8,10,9,7,-6,-5,2,-1)$ *are distributed in* 6 *traces. Each trace is represented by its normal form. The third column indicates the number of sequences in each trace.*

Let π be a permutation with c non-trivial components and at least $c-1$ oriented components. Since each sequence sorting the i^{th} component commutes with each sequence sorting the j^{th} component of π, we can group all sequences that sort one component of π in a set of traces. We denote by $\mathcal{T}_i$ the set of all traces of sequences sorting the i^{th} component of a permutation π. Table 7 represents, for instance, the sets of traces $\mathcal{T}_1$ and $\mathcal{T}_2$ that sort respectively the components C_1 and C_2 of the permutation $(-4,-3,12,-11,-8,10,9,7,-6,-5,2,-1)$ (illustrated in Figure 19).

Let $T_i \in \mathcal{T}_i$ be a trace of optimal sequences sorting the i^{th} component of π and $T_j \in \mathcal{T}_j$ be a trace of optimal sorting sequences sorting the j^{th} component of π. We know that each sequence

Component C_1

Trace	**Trace normal form**	**# seq.**
C_1^1	$f = \{1,2\}\{1,2,5,\dots,12\}\{2\}\{12\} < \{1,2,3,4\}$	60
C_1^2	$f = \{1,\dots,12\}\{2\}\{3,4,12\}\{5,\dots,11\} < \{3,\dots,11\}$	60
C_1^3	$f = \{1,\dots,12\}\{2,\dots,12\}\{2,5,\dots,12\}\{12\} < \{2,3,4\}$	60
C_1^4	$f = \{1,2\} < \{1,5,\dots,11\} < \{1,3,4,12\} < \{2,\dots,12\}\{3,\dots,11\}$	2
C_1^5	$f = \{2,\dots,12\} < \{1,3,4,12\} < \{1,5,\dots,11\} < \{1,2\}\{3,\dots,11\}$	2
C_1^6	$f = \{2,5,\dots,12\}\{5,\dots,11\} < \{1,12\} < \{1,5,\dots,11\} < \{1,2,3,4\}$	5
	Total	189

Component C_2

Trace	**Trace normal form**	**# seq.**
C_2^1	$f = \{7\}\{8,10\} < \{8,9\}$	3
	Total	3

Table 7: The traces of optimal sequences of reversals for sorting the components C_1 and C_2 of the permutation $(-4,-3,12,-11,-8,10,9,7,-6,-5,2,-1)$. Each trace is represented by its normal form. The third column indicates the number of sequences in each trace.

in T_i commutes with each sequence in T_j, thus we say that the traces T_i and T_j *commute*. We denote by $T_i \otimes T_j$ the multiplication of two traces T_i and T_j, defined as the set of all sequences that are the result of all possible combinations of each sequence in T_i with each sequence in T_j. Observe that $T_i \otimes T_j$ is equivalent to $T_j \otimes T_i$. We denote by $||T_i||$ the number of sequences in a trace T_i, and by ℓ_i the length of each sequence of a trace T_i (all sequences in all traces of T_i have the same length ℓ_i). Then the number of sequences in $T_i \otimes T_j$ corresponds to $||T_i|| * ||T_j|| * M(\ell_i, \ell_j)$, where $M(\ell_i, \ell_j)$ is the number of possible ways to merge a sequence of length ℓ_i with a sequence of length ℓ_j, such that the merged sequences are subsequences of all resulting sequences (it is easy to see that $M(\ell_i, \ell_j)$ is equivalent to $M(\ell_j, \ell_i)$). The normal form of the trace $T_i \otimes T_j$ can be obtained by adding each reversal in the normal form of T_j to the normal form of T_i with Algorithm 3.

For example, if $s_1 = \rho_1\rho_2$ and $s_2 = \theta_1\theta_2$, then all possible ways of merging s_1 and s_2 are the 6 sequences $\rho_1\rho_2\theta_1\theta_2$, $\rho_1\theta_1\rho_2\theta_2$, $\rho_1\theta_1\theta_2\rho_2$, $\theta_1\rho_1\rho_2\theta_2$, $\theta_1\rho_1\theta_2\rho_2$, and $\theta_1\theta_2\rho_1\rho_2$. Thus, the number

$M(2,2)$ is equal to 6. Moreover, consider the trace $T_1 = \rho_1\rho_2$ and the trace $T_2 = \theta_1 < \theta_2$. Since T_1 has 2 sequences ($\rho_1\rho_2$ and $\rho_2\rho_1$) and T_2 has only one sequence ($\theta_1\theta_2$), the number of sequences in $T_1 \otimes T_2$ corresponds to $||T_1|| * ||T_2|| * M(2,2) = 2 * 1 * 6 = 12$. If we suppose that θ_1 is lexicographically higher than ρ_1 and ρ_2, then the normal form of $T_1 \otimes T_2$ is $\rho_1\rho_2\theta_1 < \theta_2$.

We also denote by $\mathcal{T}_i \otimes \mathcal{T}_j$ the multiplication of two sets of traces $\mathcal{T}_i$ and $\mathcal{T}_j$, that sort respectively the i^{th} and the j^{th} components of π, such that $\mathcal{T}_i \otimes \mathcal{T}_j = \{ T_i \otimes T_j \mid T_i \in \mathcal{T}_i \textit{ and } T_j \in \mathcal{T}_j \}$. If a permutation π has at most one non-trivial unoriented component, the set of traces of optimal sequences sorting π can be obtained by the subsequent multiplication of the sets of traces sorting its components.

Proposition 6 *Let π be a permutation with c non-trivial components and at most one non-trivial unoriented component. If $\mathcal{T}$ is the set of all traces sorting π, then $\mathcal{T} = \mathcal{T}_1 \otimes \mathcal{T}_2 \otimes \ldots \otimes \mathcal{T}_{c-1} \otimes \mathcal{T}_c$.*

Proof. It is easy to see that any trace in $\mathcal{T}_1 \otimes \mathcal{T}_2 \otimes \ldots \otimes \mathcal{T}_{c-1} \otimes \mathcal{T}_c$ is also in $\mathcal{T}$. By contradiction, we show that any trace in $\mathcal{T}$ is also in $\mathcal{T}_1 \otimes \mathcal{T}_2 \otimes \ldots \otimes \mathcal{T}_{c-1} \otimes \mathcal{T}_c$. Suppose a trace $T \in \mathcal{T}$ that is not in $\mathcal{T}_1 \otimes \mathcal{T}_2 \otimes \ldots \otimes \mathcal{T}_{c-1} \otimes \mathcal{T}_c$. Since π has at most one unoriented component, each optimal sequence $s \in T$ sorting π can be partitioned in c subsequences s_1, s_2, $\ldots$, s_c, where $|s| = |s_1| + |s_2| + \ldots + |s_c|$ and each s_i contains only reversals that are internal to the i^{th} non-trivial component of π (Proposition 5). Thus T is in $\mathcal{T}_1 \otimes \mathcal{T}_2 \otimes \ldots \otimes \mathcal{T}_{c-1} \otimes \mathcal{T}_c$, which is a contradiction. □

Table 8 shows the multiplication of the sets of traces sorting the components C_1 and C_2 of the permutation $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$ to generate all the traces sorting π.

When we compute directly the traces that sort a permutation π, the traces of each component of π are computed several times. In contrast, when we compute and multiply the traces of the components to obtain the traces that sort a permutation π, the traces of each component are computed once. Thus, computing and multiplying the traces of the components to obtain the traces that sort a permutation π may be more efficient than computing directly the traces, and this is confirmed by the experimental results, as we will see in the next section.

Trace	Trace composition and normal form	# seq.
$C_2^1 \otimes C_1^1$	$\{7\}\{8,10\} < \{8,9\} \otimes \{1,2\}\{1,2,5,\ldots,12\}\{2\}\{12\} < \{1,2,3,4\}$ $f = \{1,2\}\{1,2,5,\ldots,12\}\{2\}\{7\}\{8,10\}\{12\} < \{1,2,3,4\}\{8,9\}$	$3 \otimes 60 = 10,080$ $(3 * 60 * M(3,5))$
$C_2^1 \otimes C_1^2$	$\{7\}\{8,10\} < \{8,9\} \otimes \{1,\ldots,12\}\{2\}\{3,4,12\}\{5,\ldots,11\} < \{3,\ldots,11\}$ $f = \{1,\ldots,12\}\{2\}\{3,4,12\}\{5,\ldots,11\}\{7\}\{8,10\} < \{3,\ldots,11\}\{8,9\}$	$3 \otimes 60 = 10,080$ $(3 * 60 * M(3,5))$
$C_2^1 \otimes C_1^3$	$\{7\}\{8,10\} < \{8,9\} \otimes \{1,\ldots,12\}\{2,\ldots,12\}\{2,5,\ldots,12\}\{12\} < \{2,3,4\}$ $f = \{1,\ldots,12\}\{2,\ldots,12\}\{2,5,\ldots,12\}\{7\}\{8,10\}\{12\} < \{2,3,4\}\{8,9\}$	$3 \otimes 60 = 10,080$ $(3 * 60 * M(3,5))$
$C_2^1 \otimes C_1^4$	$\{7\}\{8,10\} < \{8,9\} \otimes \{1,2\} < \{1,5,\ldots,11\} < \{1,3,4,12\} < \{2,\ldots,12\}\{3,\ldots,11\}$ $f = \{1,2\}\{7\}\{8,10\} < \{1,5,\ldots,11\}\{8,9\} < \{1,3,4,12\} < \{2,\ldots,12\}\{3,\ldots,11\}$	$3 \otimes 2 = 336$ $(3 * 2 * M(3,5))$
$C_2^1 \otimes C_1^5$	$\{7\}\{8,10\} < \{8,9\} \otimes \{2,\ldots,12\} < \{1,3,4,12\} < \{1,5,\ldots,11\} < \{1,2\}\{3,\ldots,11\}$ $f = \{2,\ldots,12\}\{7\}\{8,10\} < \{1,3,4,12\}\{8,9\} < \{1,5,\ldots,11\} < \{1,2\}\{3,\ldots,11\}$	$3 \otimes 2 = 336$ $(3 * 2 * M(3,5))$
$C_2^1 \otimes C_1^6$	$\{7\}\{8,10\} < \{8,9\} \otimes \{2,5,\ldots,12\}\{5,\ldots,11\} < \{1,12\} < \{1,5,\ldots,11\} < \{1,2,3,4\}$ $f = \{2,5,\ldots,12\}\{5,\ldots,11\}\{7\}\{8,10\} < \{1,12\}\{8,9\} < \{1,5,\ldots,11\} < \{1,2,3,4\}$	$3 \otimes 5 = 840$ $(3 * 5 * M(3,5))$
	Total	$31,752$ $(567 * M(3,5))$

Table 8: Obtaining the traces of sequences of reversals for sorting the permutation $(-4,-3,12,-11,-8,10,9,7,-6,-5,2,-1)$ *by the composition approach. The value* $M(3,5)$ *is equal to* 56.

4.5 Implementation and performance

The implementation of our algorithm to generate directly the traces[9] is part of the baobabLuna package, described in Chapter 6. We run several tests on artificial permutations to evaluate the algorithm's performance. Some results are recorded in Table 9. These numbers may be useful to give an idea of the quantities that we are dealing with, given by the numbers of sorting sequences and number of traces.

Even if we are quickly limited in the size of the permutations that it is possible to treat, there is a solid gain in relation to the previous existing methods. Observe that the main limit concerns the amount of memory that needs to be used, more than the time. In the next chapter, we present a method to deal with this problem that consists in pruning the sequences of reversals according to some biological constraints.

[9] Since fortresses are very rare in permutations that represent real genomes, the implementation of the algorithm does not deal with fortresses.

PERMUT.	N$_C$	N$_S$	N$_T$	Algorithm	Execution time
π_F $n = 12$ $d = 10$	2 $(1+9)$	$8,278,540$ $(827,854 * M(1,9))$	$2,151$	`enum seq.`	$\simeq$ `13.5 min`
				`enum+traces`	$\simeq$ `30.1 min`
				`traces`	$\simeq$ `27 sec`
				`traceCompos`	$\simeq$ `13 sec`
π_G $n = 16$ $d = 12$	2 $(1+11)$	$505,634,256$ $(42,136,188 * M(1,11))$	$21,902$	`enum seq.`	$\simeq$ `16 h`
				`enum+traces`	$\simeq$ `43.5 h`
				`traces`	$\simeq$ `7.3 min`
				`traceCompos`	$\simeq$ `3.2 min`
π_H $n = 16$ $d = 13$	2 $(1+12)$	$40,313,272,766$ $(3,101,020,982 * M(1,12))$	$567,524$	`enum seq.`	-
				`enum+traces`	-
				`traces`	$\simeq$ `4.1 hours`
				`traceCompos`	$\simeq$ `1.7 hours`

Table 9: Computation results (1). Columns from left to right contain: 1- the permutation, its number of elements and reversal distance; 2- the number of components and how the reversal distance is divided between components; 3- the number of sorting sequences (in parenthesis the number of sorting sequences computed by `traceCompos`*); 4- the number of traces; 5- the algorithm (*`enum seq` *is the algorithm that enumerates all the sorting sequences,* `enum+traces` *is the algorithm that computes the traces by enumerating all sorting sequences,* `traces` *is Algorithm 5, that enumerates directly the traces,* `traceCompos` *is the algorithm that composes a trace of a permutation π by multiplying the traces of the components of π); 6- the execution time of each algorithm. The three analyzed permutations are $\pi_F = (-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$, $\pi_G = (-12, 11, -10, -1, 16, -4, -3, 15, -14, 9, -8, -7, -2, -13, 5, -6)$, and finally the permutation $\pi_H = (-12, 11, -10, 6, -5, 13, 2, 7, 8, -9, 14, -15, 3, 4, -16, 1)$, that can not be processed by the algorithms* `enum` *and* `enum+traces` *due to its huge number of sorting sequences. All algorithms are implemented as part of the* BAOBABLUNA *package.*

4.6 FINAL REMARKS

In this chapter we presented the method previously proposed by Siepel [59] to generate the space of all sorting sequences for a given permutation π. Then we gave one of our most important contributions, that is a method for generating directly a more compact representation of the space of all sorting sequences, in which the sequences are grouped in equivalence classes called traces, using a model previously proposed by Bergeron *et al.* [9]. We showed that the theoretical complexity of our algorithm is exponential on the width (represented by k), that is a property of the resulting traces.

The algorithm was implemented, integrated to BAOBABLUNA, a framework to deal with permutations and reversals, that will be described in Chapter 6. The experimental results show that the number of traces is considerably smaller than the number of sorting sequences. Consequently, computing directly the traces runs considerably faster than computing all sequences. However, the number of traces may be still too big for being interpreted, and in some cases, too big for being computed. Indeed, we verified that currently we are unable to compute traces for permutations with a reversal distance of about 20 or higher. Nevertheless, for small reversal distances, which is the case of a permutation analyzed by Ross *et al.* [55] in the study of the evolution of human sexual chromosomes X and Y, and of the permutations analyzed by Blanc *et al.* [13] in the study of the evolution of *Rickettsia* bacteria, as we will see in the next chapter, our program may give a more interesting result than programs that give a unique solution, such as `GRIMM` [64].

We also showed that, for any permutation π that has at most one unoriented component, we can compute the set of traces that sort each component of π independently, and then obtain the traces sorting π by multiplying the traces sorting its components. This approach runs faster than computing directly the traces.

In the next chapter we will introduce the use of some biological constraints in the enumeration of traces, to reduce the universe of generated traces, that is however not compatible with the strategy of constructing traces by composition.

Chapter 5

Biological constraints and applications

Summary

5.1 Modeling traces with biological constraints 54
5.2 Common intervals 55
5.2.1 Initial detection of common intervals 57
5.2.2 Progressive detection of common intervals 58
5.2.3 Theoretical complexity and experiments 61
5.2.4 Accepting interval breaks 64
5.3 Replication origin and terminus in prokaryotic circular chromosomes 65
5.3.1 Analysis of the *Rickettsia* bacterium 70
5.3.2 Evaluating the execution time 75
5.4 Stratification on evolution of sexual chromosomes 75
5.4.1 Model of evolution by strata 77
5.4.2 Algorithm for exploring the sequences that stratify a permutation 78
5.4.3 Analysis of the results 79
5.4.4 On the execution time of the strata variant 82
5.4.5 A deeper study of the human X and Y sexual chromosomes 82
5.4.5.1 Analysis of strata in extended permutations 83
5.4.5.2 Simulations to estimate the stratification likelihood 84
5.5 Symmetry *versus* asymmetry when applying constraints 88
5.6 Compatibility between constraints 89
5.7 Final remarks 90

The space of sorting sequences is dramatically reduced when dealing with traces, but it is often still too big and can not be computed. Even when it can be computed, frequently it is too big to be handled by biologists on large permutations. An idea to try to solve these problems is to add further biological constraints to reduce the number of traces. In order to be able to also reduce the amount of required memory and push further the limits of our algorithm, we may check the constraints during the computation of traces, and not only *a posteriori*. We should be able to filter the reversals at each step, selecting only those that are in agreement with the given constraints.

We considered several different biological constraints. One of these constraints is the list of common intervals detected between the two initial permutations, that may correspond to the list of clusters of co-localised genes between the considered genomes - an optimal sequence of reversals that does not break the common intervals may be more realistic than one that does break [26]. This approach was previously used in other studies that take common intervals in consideration when sorting by reversals [6, 7, 10, 26]. We used the common intervals initially detected as a constraint and we also proposed a new variation of this approach, that is the list of common intervals progressively detected when sorting one permutation into another by reversals. Other constraints were defined according to the practical problems we were interested in. In particular, we are able to characterize the reversals with respect to the replication terminus in circular chromosomes (this method was used to analyze the evolution of the *Rickettsia* bacterium), and to apply a constraint to analyze directly the stratification process in the evolution of the sexual chromosomes X and Y in human [18, 41].

All variants of the algorithm to generate traces taking biological constraints in consideration are implemented as part of BAOBABLUNA, that will be described in Chapter 6. The experimental results show that the number of traces is considerably reduced when the biological constraints are applied. Consequently, these variants run faster than the algorithm that generates all the traces (all experiments were made on a 64 bit personal computer with two 3GHz CPUs and 2GB of RAM). As mentioned, we applied our methods to analyze two real cases, the evolution of sexual chromosomes X and Y and the *Rickettsia* bacterium, obtaining a better characterization of these evolutionary scenarios than previous studies, that were based on a single sorting sequence.

5.1 Modeling traces with biological constraints

Besides two signed permutations π and π_T, this method requires a list of compatible q constraints $C = (C_1, C_2, \ldots, C_q)$ for selecting the sequences that sort π into π_T. We search for traces of sorting sequences that are in agreement with the given constraints. However, frequently only a subset of the sorting sequences of a trace is in agreement with the constraints in C, and this subset is called a $C-$induced subtrace. The trace construction remains unchanged, but, as a consequence of the selection of the reversals to be performed, we in fact construct the $C-$induced subtraces, that compute only the sorting sequences that are in agreement with all the constraints in C. The result of applying this method is the complete set of non-empty $C-$induced subtraces and their sizes for the two given permutations and a list of constraints C. Generally we have no guarantee that a sorting sequence that respects all constraints exists, thus we may have an empty result.

Moreover, frequently the normal form of a trace T is not part of its $C-$induced subtrace t. Due to this, when constructing $C-$induced subtraces, we also give at least one valid representative of each $C-$induced subtrace t, besides the normal form of the trace T that contains t. A $C-$induced subtrace t can be thus represented by a $2-$tuple (e, f), where e is any sorting sequence in t and f is the normal form of the trace T that contains t.

We analyzed qualitatively how the constraints may affect the chronology of the reversals, showing that some of these constraints lead to symmetric (when the results of sorting a permutation π into a permutation π_T can be obtained from the results of sorting π_T into π) and others lead to asymmetric approaches. Analogously to the notation used with traces, for a given subtrace t of optimal sequences sorting π into π_T, we define the inverse of t as $inv(t) = \{ \ inv(s) \mid s \in t \ \})$. A list of constraints C is said to be *symmetric* when we have a $C-$induced subtrace t sorting π into π_T if, and only if, $inv(t)$ is also a $C-$induced subtrace sorting π_T into π. Otherwise, C is said to be *asymmetric*.

5.2 Common intervals

Clusters of co-localised genes are intervals of the genomes composed by the same genes but not necessarily in the same order and orientations. These clusters are modeled as common intervals. The *common intervals* of two permutations π and π_T are the intervals of π that are present in π_T. For example, the interval $\{2, 3, \ldots, 7, 8\}$ is common to the permutations $\pi = (-5, -2, -7, 4, -8, 3, 6, -1)$ and $\mathcal{I}_8 = (1, 2, 3, 4, 5, 6, 7, 8)$. The idea behind common intervals is that, if these genes are together in both species, then probably they were together in the common ancestor of the two species and were not separated by evolution.

A reversal ρ breaks an interval θ if ρ and θ overlap. Considering, for instance, the permutation $(-5, -2, -7, 4, -8, 3, 6, -1)$, we observe that the reversal $\{1, 3, 4, 6, 7, 8\}$ breaks the interval $\{2, \ldots, 8\}$. We say that all intervals with size equal to 1 and the interval with size n, that comprises the entire permutation, are trivial common intervals (observe that a reversal never breaks a trivial common interval).

The concept of *irreducible common intervals* has been introduced by Heber and Stoye [34]. The authors observed that any common interval may contain several smaller common intervals, and defined as *irreducible common interval* a common interval that does not contain any other common interval different from itself. Then, the authors showed that any common interval θ between two permutations π and π_T has a generating chain of irreducible intervals $(\gamma_1, \gamma_2, \ldots, \gamma_k)$, such that the irreducible intervals γ_1, γ_2, $\ldots$, γ_k are listed in lexicographic order, and, for each pair of consecutive irreducible intervals γ_j, γ_{j+1}, we have $\gamma_j \cap \gamma_{j+1} \neq \emptyset$. A *reducible common interval* is a common interval whose generating chain has length at least two, otherwise the common interval is irreducible. For example, the generating chain of the reducible common interval $\{1, 2, 3\}$ between the permutations $(-3, 2, 1, -4)$ and $\mathcal{I}_4$ is $(\{1, 2\}, \{2, 3\})$ (the common intervals $\{1, 2\}$ and $\{2, 3\}$ are irreducible). Testing whether a reversal breaks an irreducible common interval is sufficient to determine whether it breaks a common interval.

Proposition 7 *A reversal ρ breaks a reducible interval θ, if, and only if, ρ breaks at least one irreducible interval in the chain that generates θ.*

Proof. It is easy to see that breaking a irreducible interval in the chain that generates a reducible interval θ also breaks θ. Since each pair of consecutive irreducible intervals in the chain that generates θ have a non-empty intersection, breaking θ breaks at least one irreducible interval in the chain that generates θ. □

As a consequence of Proposition 7, if ρ does not break any irreducible interval between two permutations π and π_T, then ρ does not break any reducible interval between π and π_T as well. While the number of common intervals is bounded by n^2, the number of irreducible common intervals is bounded by n [34], where n is the size of the input permutations.

5.2.1 Initial detection of common intervals

Common intervals between genomes have been the topic of several studies [6, 7, 10, 26]. Nevertheless, in the comparison of two permutations, the detection of common intervals is usually done at the beginning of the analysis, an approach that we call *initial detection of common intervals.* An optimal sequence of reversals sorting a permutation π into π_T that does not break any (irreducible) common interval initially detected between π and π_T is called a *perfect sorting sequence.* Figure 20 shows a non-perfect and a perfect optimal sequences of reversals.

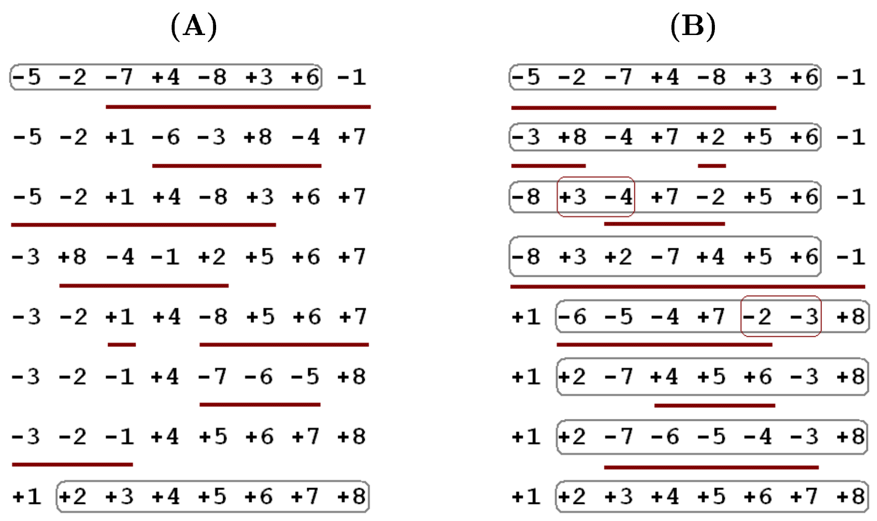


Figure 20: The permutations $(-5, -2, -7, 4, -8, 3, 6, -1)$ *and* $(1, 2, 3, 4, 5, 6, 7, 8)$ *have only one initially detected irreducible common interval, which is* $\{2, \ldots, 8\}$*. (A) The reversals* $\{1, 3, 4, 6, 7, 8\}$*,* $\{3, 4, 6, 8\}$*,* $\{1, \ldots, 5, 8\}$*,* $\{1, 2, 4, 8\}$*,* $\{1\}$*,* $\{5, \ldots, 8\}$*,* $\{5, 6, 7\}$ *and* $\{1, 2, 3\}$ *sort the permutation, but do not preserve the initially detected common interval. (B) The sequence of reversals* $\{2, \ldots, 5, 7, 8\}$*,* $\{3, 8\}$*,* $\{2\}$*,* $\{2, 4, 7\}$*,* $\{1, \ldots, 8\}$*,* $\{2, 4, \ldots, 7\}$*,* $\{4, 5, 6\}$ *and* $\{3, \ldots, 7\}$ *is a perfect sorting sequence that preserves the initially detected common interval, but does not preserve the new common intervals that appear during the sorting process (such as* $\{3, 4\}$ *and* $\{2, 3\}$*).*

We analyze the behaviour of traces with respect to sequences that do not break the initially detected common intervals [18]. First, we remark that either all sequences of a trace do not break common intervals initially detected, or all sequences of a trace break at least one common interval initially detected.

Proposition 8 *Every trace of optimal sequences for sorting a signed permutation by reversals contains either only perfect sorting sequences or no perfect sorting sequence.*

Proof. If any sequence $s = \rho_1 \ldots \rho_k$ for sorting a permutation π is perfect, by definition none of $\rho_1, \ldots, \rho_k$ overlap some common interval of π. So any sequence with the same reversals in a different order is perfect. This is the case for all the sequences of a trace, so if one sequence of a trace is perfect, they all are. □

Due to this property, a trace that contains perfect sorting sequences of length $d(\pi)$ is called a *perfect trace* (the normal form of a perfect trace is thus a perfect sorting sequence). Such a trace does not always exist: all optimal sequences may break common intervals (see [26]).

In addition, given two permutations π and π_T, we observe that searching for perfect traces is a symmetric approach. Indeed, since the list of common intervals do not change when the reversals are applied, if s is a perfect sequence of reversals s that sorts π into π_T, then $inv(s)$ is a perfect sequence of reversals that sorts π_T into π.

To compute the perfect traces, we need to introduce a few modifications to the original algorithm. We should first compute the initial irreducible common intervals between the two given permutations. Then, each time we compute the 1−sequences with Siepel's algorithm, we need to verify whether each one of the resulting 1−sequences breaks or not an irreducible common interval (the 1−sequences that break irreducible common intervals are simply discarded). At the end, we have only the perfect traces, if at least one perfect trace exists. If no perfect trace exists for the given permutations, we have an empty result.

For comparison purposes, the experimental results of applying this method will be presented together with the results of the next method.

5.2.2 Progressive detection of common intervals

In the previous approach, the new common intervals that could appear between an intermediary permutation, after applying some reversals to the initial permutation, and the target permutation, are not considered. Thus, if a common interval appears between an intermediary permutation and the target permutation, there is no constraint on the selection of a reversal that breaks this new interval (see Figure 20 (B)). Alternatively to the initial detection, in this work we introduce the *progressive detection of common intervals* [17], that consists in updating the list of (irreducible) common intervals between the permutations after each reversal. An optimal sorting sequence that does not break the progressively detected irreducible common intervals is called *progressive perfect sorting sequence.* Figure 21 shows an example of this approach.

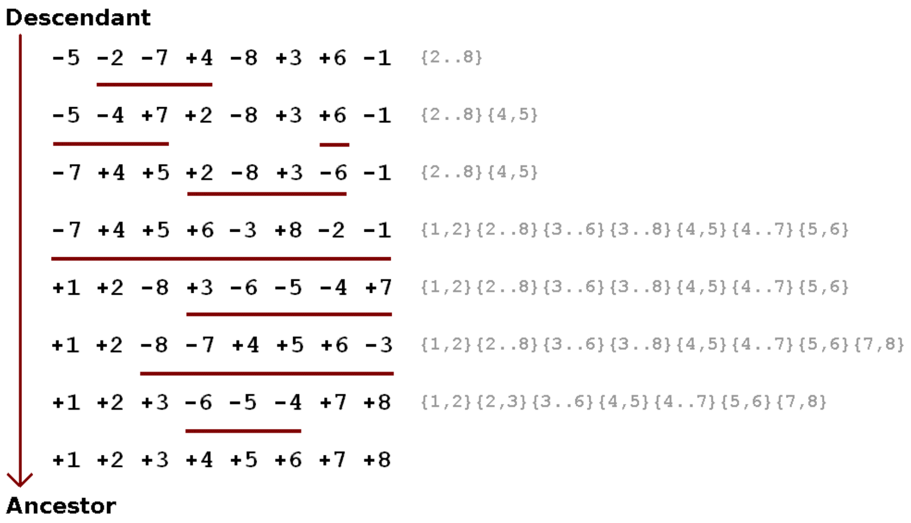


Figure 21: An optimal sequence of reversals to sort the permutation $(-5, -2, -7, 4, -8, 3, 6, -1)$ *without breaking the progressively detected irreducible common intervals (listed on the right side).*

If we consider the progressive detection of common intervals in the construction of traces, Proposition 8 does not hold anymore. Considering the permutation $(-5, -2, -7, 4, -8, 3, 6, -1)$, for instance, the sequences of reversals $\{2, \dots, 5, 7, 8\}$, $\{3, 8\}$, $\{3, 4, 7\}$, $\{1, \dots, 8\}$, $\{2\}$, $\{4\}$, $\{2, 3, 4\}$, $\{2, \dots, 6\}$ and $\{3, 8\}$, $\{3, 4, 7\}$, $\{2, \dots, 5, 7, 8\}$, $\{1, \dots, 8\}$, $\{2\}$, $\{4\}$, $\{2, 3, 4\}$, $\{2, \dots, 6\}$ are in the same trace but, while the first preserves the progressively detected common intervals (as we can see in Figure 21), the second does not (after applying the two first reversals, $\{3, 8\}$ and $\{3, 4, 7\}$, we have the permutation $(-5, -2, 3, -4, 7, 8, 6)$ with the common interval $\{6, 7, 8\}$ which overlaps with the third reversal, $\{2, \dots, 5, 7, 8\}$). Thus, when we take the progressively detected common intervals in consideration, for each trace, only a subset of its sorting sequences is selected. We call this subset a *progressive perfect subtrace.*

In addition, inverting a progressive perfect sorting sequence that sorts a first into a second permutation generally does not result in a progressive perfect sorting sequence that sorts the second permutation into the first. An example is given in Figure 21. Observe that, applying the last reversal $\{4, 5, 6\}$ on the permutation $(1, 2, 3, 4, 5, 6, 7, 8)$ results in the permutation $(1, 2, 3, -6, -5, -4, 7, 8)$, that has the common interval $\{4, 7, 8\}$ with respect to the permutation $(-5, -2, -7, 4, -8, 3, 6, -1)$. The reversal $\{3, \dots, 7\}$ (the third from bottom to top in Figure 21) overlaps with $\{4, 7, 8\}$, thus inverting the progressive perfect sequence of reversals $\{2, 4, 7\}$, $\{4, 5, 7\}$, $\{6\}$, $\{2, 3, 6, 8\}$, $\{1, \ldots, 8\}$, $\{3, \ldots, 7\}$, $\{3, \ldots, 8\}$, $\{4, 5, 6\}$ that sorts $(-5, -2, -7, 4, -8, 3, 6, -1)$ into $(1, 2, 3, 4, 5, 6, 7, 8)$ does not result in a progressive perfect sequence of reversals that sorts $(1, 2, 3, 4, 5, 6, 7, 8)$ into $(-5, -2, -7, 4, -8, 3, 6, -1)$. Thus, differently from searching for perfect traces, searching for progressive perfect traces is an asymmetric approach.

When we compare current species, it is not possible to determine a direction to the analysis. In this case, considering common intervals that appear in intermediary states is meaningless and a symmetric approach is more adequate. Symmetry is thus an advantage that supports the initial detection of common intervals in many applications. We suggest however that, when the relation ancestor-descendant between the analyzed genomes is clear, the progressive detection of common intervals may be more realistic than the initial detection of common intervals. In this case, the analysis should be done from the descendant to the ancestor, since the objective is to regroup intervals that may have existed in a past time.

To construct the progressive perfect subtraces, we need to modify Algorithm 5. Analogously to the notation previously introduced, a progressive perfect subtrace whose sorting sequences have i reversals is called progressive perfect $i-$subtrace, and a progressive perfect $k-$subtrace t' is a $k-$*prefix* of a progressive perfect $i-$subtrace t $(k \leq i)$ if each $k-$sequence of t' is a prefix of an $i-$sequence of t. To compute the progressive perfect subtraces, at each step we use the algorithm of Siepel [59] to list all possible $1-$sequences. Then we filter these $1-$sequences to discard those that break irreducible common intervals. As a result of this procedure (see Algorithm 6), we construct directly the progressive perfect subtraces.

As in the original algorithm, we may need to compare subtraces to verify whether a new subtrace t is present in the list of already constructed subtraces (Algorithm 6, step COMPARISON). In order to do that, we may obtain the normal form f of the trace T that contains t, and compare f to the normal forms of the traces that contain the already constructed subtraces (the normal form of an $i-$trace is constructed incrementally, from the normal form of one of its $(i-1)-$prefixes; see Algorithm 3). Since there is no guarantee that the normal form is part of a progressive perfect subtrace, we also give one arbitrary valid sorting sequence in t as a representative (the representative of an $i-$subtrace is also constructed incrementally, by concatenating a reversal in the end of the sequence that represents one of its $(i-1)-$prefixes). A progressive perfect subtrace t is thus represented by a $2-$tuple (e, f), where e is any progressive perfect sorting sequence in t and f is the normal form of the trace T that contains t.

The sequence $\{1,\dots,8\}\{2,4,7\}\{6\} < \{2,3,6,8\}\{4,5,7\} < \{3,\dots,7\}\{3,\dots,8\}\{4,5,6\}$ is the normal form of the sorting sequence described in Figure 21. The normal form is not a progressive perfect sequence, because after applying the reversals $\{1,\dots,8\}$, $\{2,4,7\}$, $\{6\}$ and $\{2,3,6,8\}$ on the permutation $(-5,-2,-7,4,-8,3,6,-1)$ we obtain the permutation $(1,2,-8,3,-6,-7,4,5)$, that has the irreducible common interval $\{6,7\}$ with respect to the target permutation $\mathcal{I}_8 = (1,2,3,4,5,6,7,8)$. The next reversal in the normal form is $\{4,5,7\}$, that breaks this new interval. In this example, the normal form is not a valid representative. However, the progressive perfect subtrace that contains the sorting sequence described in Figure 21 can be represented by the $2-$tuple (e, f), where e is a valid progressive perfect sequence representative) and f is the normal form (see Table 10).

2-tuple
$e = \{2,4,7\}\{4,5,7\}\{6\}\{2,3,6,8\}\{1,\dots,8\}\{3,\dots,7\}\{3,\dots,8\}\{4,5,6\}$
$f = \{1,\dots,8\}\{2,4,7\}\{6\} < \{2,3,6,8\}\{4,5,7\} < \{3,\dots,7\}\{3,\dots,8\}\{4,5,6\}$

Table 10: The 2-tuple representing one progressive perfect subtrace of optimal sequences that sort the permutation $(-5,-2,-7,4,-8,3,6,-1)$.

Thus, for two given permutations π and π_T, at the end of Algorithm 6, we have the list of all non-empty progressive perfect subtraces. If no progressive perfect sequence exists for sorting π into π_T, we have an empty result.

5.2.3 Theoretical complexity and experiments

As we have seen before, Algorithm 5 has complexity $O(Nn^{k_{max}+4})$, where n is the size of the input permutation π, N is the number of computed final traces and k_{max} is the maximum value for the *width* of a final trace. The 4 in the exponent of this formula is due to the processing of each $(i-1)-$trace T to generate the subsequent $i-$traces, given by the following procedure: (1) apply the sequences of reversals of f, which is the normal form of T, on the initial permutation π to obtain π_f; (2) run Siepel's algorithm [59] over π_f; (3) add each one of the $O(n^2)$ reversals

Algorithm 6 Enumerating all the progressive perfect subtraces of two signed permutations

Input: Two signed permutations π, π_T

Output: The representative, normal form and counter (e, f, c) of each progressive perfect subtrace of sequences of reversals sorting π into π_T

```
$d \leftarrow$ reversal distance of $(\pi, \pi_T)$
$\mathcal{T} \leftarrow \emptyset$
$I \leftarrow \{\theta \mid \theta$ is a irreducible common interval between $\pi$ and $\pi_T\}$   [computing irred. common interv. [34]]
$S \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $\pi \rightarrow \pi_T\}$   [Siepel [59]]
for each $1-$sequence $\rho \in S$ do
  if $\rho$ does not break an interval in $I$   [filtering]   then
    insert $(\rho, \rho, 1)$ in $\mathcal{T}$   [each perfect first $1-$sequence is a progressive perfect $1-$subtrace]
  end if
end for
for each integer $i$ from 2 to $d$ do
  $\mathcal{T}' \leftarrow \emptyset$   [contains the representatives/normal forms/counters of all the progressive perfect $i-$subtraces]
  for each $(e, f, c)$ in $\mathcal{T}$   [$(e, f)$ repr. the prog. perfect $(i-1)-$subtrace $t$; $c$ is the counter of $t$]   do
    $\pi_f \leftarrow \pi \circ f$   [apply the $(i-1)-$sequence $f$ to $\pi$]
    $I_f \leftarrow \{\theta \mid \theta$ is a irreducible common interval between $\pi_f$ and $\pi_T\}$   [comp. irred. common interv. [34]]
    $S_f \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $\pi_f \rightarrow \pi_T\}$   [Siepel [59]]
    for each $1-$sequence $\rho \in S_f$ do
      if $\rho$ does not break an interval in $I_f$   [filtering]   then
        $f_\rho \leftarrow f + \rho$   [extend the normal form $f$ by adding the reversal $\rho$; see Algorithm 3]
        if there exists $(e', f', c') \in \mathcal{T}'$ such that $f' = f_\rho$   [COMPARISON]   then
          $c' \leftarrow c' + c$   [upd. the counter of the progressive perfect $i-$subtrace $t'$ repr. by $(e', f')$]
        else
          $e_\rho \leftarrow e \cdot \rho$   [simply concatenate $\rho$ to the sequence $e$]
          insert $(e_\rho, f_\rho, c)$ in $\mathcal{T}'$   [$(e_\rho, f_\rho)$ repr. the prog. perfect $i-$subtrace $t_\rho$; $c$ is the counter of $t_\rho$]
        end if
      end if
    end for
  end for
  $\mathcal{T} \leftarrow \mathcal{T}'$
end for
return $\mathcal{T}$   [$\mathcal{T}$ is the final set of progressive perfect $d-$subtraces sorting $\pi$ into $\pi_T$]
```

returned by Siepel's algorithm to T to compute a new $i-$trace. The complexity of this procedure is $(1) + (2) + (3) = n^2 + n^3 + n^2.n^2$, that results in $O(n^4)$.

With respect to the original algorithm, in order to compute progressive perfect subtraces we added two new steps to the processing of an $(i-1)-$subtrace t to generate the following $i-$subtraces: (1B) computing the irreducible common intervals in π_f; (2B) filtering each reversal returned by Siepel's algorithm. Computing the irreducible common intervals can be done in $O(n)$ time [34]. Filtering the reversals, that is, testing whether each one of the $O(n^2)$ reversals returned by Siepel's algorithm overlaps with each one of the irreducible common intervals can take $n^2.n.n$, because comparing two intervals (a reversal and a common interval) takes $O(n)$ and each reversal has to be compared to $O(n)$ [34] irreducible common intervals. Thus, the complexity of processing an $(i-1)$–subtrace is given by $(1)+(1B)+(2)+(2B)+(3) = n^2+n+n^3+n^4+n^4$, that results in $O(n^4)$. Consequently, the complexity of the modified algorithm is $O(Ln^{k_{max}+4})$, where L is $O(N)$ and represents the number of computed final progressive perfect subtraces.

Observe that, for calculating perfect traces, we compute the irreducible common intervals once for the input permutation π, and then we only have to introduce the filtering step, whose complexity is $O(n^4)$, in the original algorithm. Thus, the theoretical complexity in this case is $O(Mn^{k_{max}+4})$, where M, the number of computed final perfect traces, is also $O(N)$.

We implemented both algorithms, to compute perfect traces and progressive perfect subtraces, integrated to the BAOBABLUNA package (described in chapter 6), which also contains the implementation of computing traces. Although the theoretical complexity of the new approaches is equal to the original approach, the experimental results, presented in Table 11, revealed that searching for reversals that do not break common intervals is a constraint that usually reduces the number of traces and sorting sequences, and consequently, the execution time. Moreover, the reduction is considerably higher when we apply the progressive detection of common intervals (usually $L < M << N$).

Permutation	**Algorithm**	**N$_S$**	**N$_T$**	**Execution time**
A and $\mathcal{I}_8$	`all traces` $(A \leftrightarrow \mathcal{I}_8)$	$81,869$	377	$\simeq$ `5 seconds`
$d(A,\mathcal{I}_8) = 8$	`perfect traces` $(A \leftrightarrow \mathcal{I}_8)$	$51,304$	92	$\simeq$ `5 seconds`
	`p. perf. subtr.` $(A \rightarrow \mathcal{I}_8)$	$11,568$	12	$\simeq$ `3 seconds`
	`p. perf. subtr.` $(\mathcal{I}_8 \rightarrow A)$	$8,400$	5	$\simeq$ `2 seconds`
B and $\mathcal{I}_{16}$	`all traces` $(B \leftrightarrow \mathcal{I}_{16})$	$505,634,256$	$21,902$	$\simeq$ `7.3 minutes`
$d(B,\mathcal{I}_{16}) = 12$	`perfect traces` $(B \leftrightarrow \mathcal{I}_{16})$	$122,862,960$	171	$\simeq$ `27 seconds`
	`p. perf. subtr.` $(B \rightarrow \mathcal{I}_{16})$	$5,963,760$	6	$\simeq$ `14 seconds`
	`p. perf. subtr.` $(\mathcal{I}_{16} \rightarrow B)$	$5,393,520$	9	$\simeq$ `16 seconds`

Table 11: The experimental results of computing traces, perfect traces and progressive perfect subtraces (in both directions), considering the pairs of permutations given by $(A, \mathcal{I}_8)$, where $A = (-5,-2,-7,4,-8,3,6,-1)$, and by $(B, \mathcal{I}_{16})$, where $B = (-12,11,-10,-1,16,-4,-3,15,-14,9,-8,-7,-2,-13,5,-6)$. All algorithms are part of the BAOBABLUNA *package.*

5.2.4 Accepting interval breaks

As mentioned, searching for perfect traces or for progressive perfect subtraces may reduce the number of sorting sequences and traces. However, there is no guarantee that a perfect sorting sequence or a progressive perfect sorting sequence exists, thus those approaches may eventually lead to empty results. For example, the permutation $(1, 3, -2, -11, 5, -9, -10, 8, 6, -7, -4, 12)$, whose reversal distance is 9, has no perfect sorting sequence and no progressive perfect sorting sequence.

Due to this, we propose the construction of near-perfect traces, accepting a bounded number of breaking reversals per trace. A reversal can have a score of 0 if it does not break any common interval, or a score of 1 if one of its extremities breaks common intervals, or of 2 if both extremities break common intervals (see Figure 22 (B)). The score of a sequence of reversals is given by the sum of the scores of its reversals and is bounded by k.

Differently from the perfect sequences, the near-perfect sequences of reversals are asymmetric, that is, inverting a near-perfect sequence of reversals sorting a permutation π into π_T with score equal to k does not necessarily result in a near-perfect sequence of reversals sorting π_T into π with the same score k. The reason is that, after being broken, a common interval is no longer common and should be removed from the initial list of common intervals (see Figure 22 (A)). Thus, the list of common intervals may be different at each step and depends on the order the reversals are applied.

For example, when sorting $(1, 3, -2, -11, 5, -9, -10, 8, 6, -7, -4, 12)$ into $\mathcal{I}_{12}$ there is no perfect sequence of reversals, and we must accept at least two interval breaks. The irreducible common intervals between these two permutations are $\{1, 2, 3\}$, $\{2, 3\}$, $\{2, \ldots, 11\}$, $\{2, \ldots, 12\}$, $\{4, \ldots, 10\}$, $\{4, \ldots, 11\}$, $\{4, \ldots, 12\}$, $\{5, \ldots, 10\}$, $\{5, \ldots, 11\}$, $\{5, \ldots, 12\}$, $\{5, \ldots, 11\}$, $\{5, \ldots, 12\}$, $\{6, 7\}$, $\{6, 7, 8\}$, $\{6, 7\}$, $\{8, 9, 10\}$, $\{9, 10\}$. To construct a sequence of score 2, we can first apply the non-breakings reversals $\{2, 3\}$, $\{3\}$, $\{4, \ldots, 11\}$, $\{5, \ldots, 10\}$, $\{7\}$ and $\{9\}$ and obtain $(1, 2, 3, 4, 5, 9, -10, 8, 6, 7, 11, 12)$. Then we apply the reversal $\{6, 7, 8, 10\}$, with score equal to one, that breaks the intervals $\{8, 9, 10\}$ and $\{9, 10\}$. The next reversal is $\{6, 7, 9\}$, that breaks the interval $\{6, 7, 8\}$ and also has score equal to one. Then the last reversal is $\{8, 9\}$, which is non breaking. But observe that if we do not remove the already broken intervals from the initial list, the last reversal should be considered a breaking one (it also "breaks" the interval $\{9, 10\}$), and this sequence would have a score of 3 instead of 2.

A consequence of updating the list of common intervals when we accept a number of interval breaks bounded by k is that we have near-perfect subtraces instead of traces. Similarly of what happens when we use the progressive detection, only a subset of the sequences in a trace may achieve the given score k, and this process is not symmetric. Thus, when we accept interval breaks, we are not able to keep the symmetry. In other words, although the perfect traces are symmetric, the near-perfect subtraces are asymmetric and this should be taken in consideration when we apply this method in the analysis of real cases.

We can also accept interval breaks when searching for progressive perfect subtraces. As for the progressive perfect subtraces, the progressive near-perfect subtraces are also asymmetric.

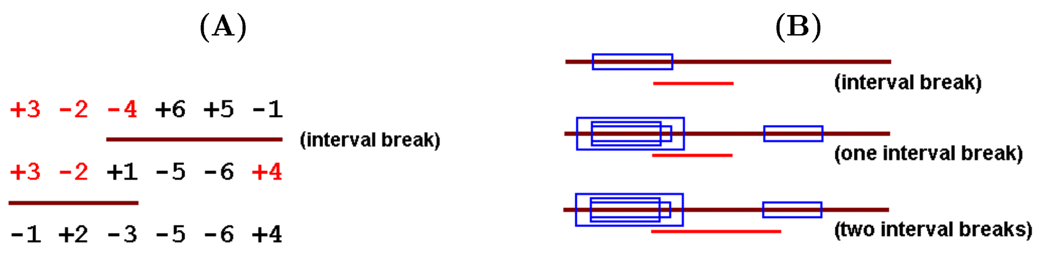


Figure 22: (A) After being broken, an interval is no longer common. (B) A reversal may cause at most two interval breaks.

In the next section we will see how the conservation of progressively detected common intervals, accepting some interval breaks, can be combined with another constraint to analyze the evolutionary scenario between the bacterium *Rickettsia felis* and one of its ancestors, that have been reconstructed by Blanc *et al.* [13].

5.3 Replication origin and terminus in prokaryotic circular chromosomes

In prokaryotic circular chromosomes, it was observed that reversals are more likely to happen symmetrically around the replication terminus; such a reversal is called *terminus-symmetric reversal*. Thus, the other types of reversals, that we call *external*, when it does not contain neither the replication origin nor the replication terminus, and *terminus-asymmetric*, when it happens asymmetrically around the replication terminus, are more rare than terminus-symmetric reversals.

Although the existence of a mechanism that could favor the occurrence of terminus-symmetric reversals remains a possibility [27], Mackiewicz *et al.* [42] showed several arguments that indicate that natural selection plays an important role in this process. They observed that terminus-symmetric reversals are the only that are able to preserve two important properties of a gene, that are its distance with respect to the replication terminus and origin, and its orientation with respect to the replication direction. The terminus-asymmetric reversals also preserve the orientation of a gene with respect to the replication direction, but may displace the replication terminus (analogously, replication origin). The external reversals do not affect the positions of the replication terminus and origin, but may change the distance of a gene with respect to the replication terminus and origin and also its orientation with respect to the replication direction. Thus, according to Mackiewicz *et al.* [42], terminus-asymmetric and external reversals may often lead to deleterious mutations. The three types of reversals are illustrated in Figure 23.

In general, terminus-symmetric reversals are not perfectly symmetric. If a reversal occurs around the replication terminus, we can denote by a the distance between the first extremity of the reversal to the replication terminus and by b the distance between the replication terminus and the last extremity of the reversal. We define as *terminus-symmetry rate* the value between 0 and 1, given by the minimum of a and b, divided by the maximum of a and b. We differentiate terminus-symmetric and terminus-asymmetric reversals by a threshold on the terminus-symmetry rate. Figure 24 illustrates the terminus-symmetric, external and terminus-asymmetric reversals.

We assume that the values of a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, that represents a circular chromosome, are given according to the positions of the markers represented by $\pi_1, \pi_2, \ldots, \pi_n$ with respect to the replication origin (that is, if we do a tour on the chromosome from 5' to 3', π_1 represents the marker that follows the replication origin, and so on, thus π_n represents the marker that precedes the replication origin). The same is valid for the target permutation π_T. The length of the markers can be set (each marker is assumed to have the same length in both genomes), and, since prokaryotic genomes are dense in genes, we may ignore the space between markers. Then, the replication terminus is inferred to be exactly in the middle of the

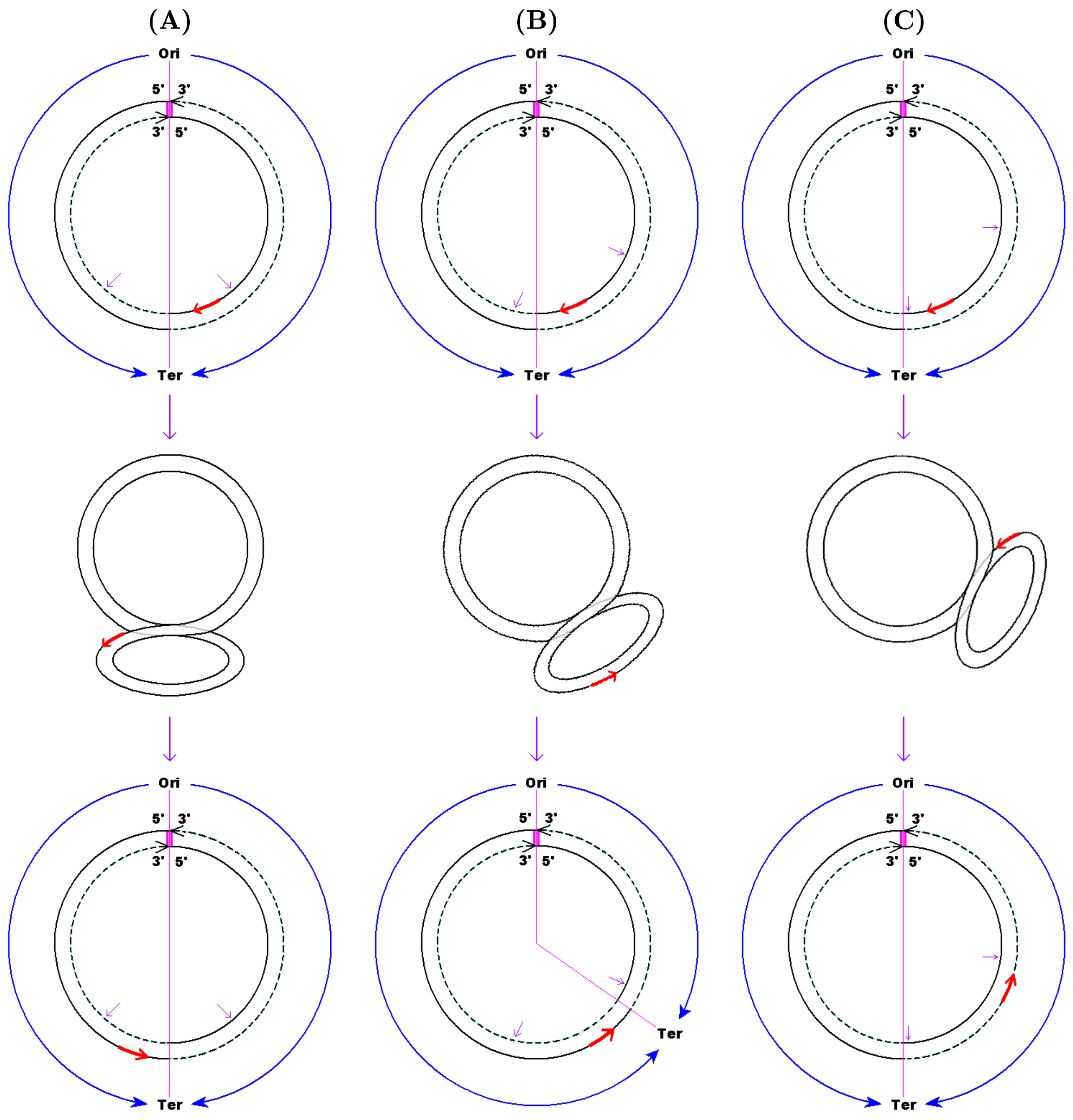


Figure 23: With respect to the replication terminus, a reversal can be symmetric, asymmetric or external. The rounded blue arrows indicate the replication directions. The red arrow over the chromosome represents a gene that is within the portion of the chromosome that is affected by a reversal. (A) A terminus-symmetric reversal preserves the distance of a gene with respect to the replication terminus and origin, and its orientation with respect to the replication direction. (B) A terminus-asymmetric reversal preserves the orientation of a gene with respect to the replication direction, but may displace the replication terminus (analogously, replication origin). (C) An external reversal does not affect the positions of the replication terminus and origin, but may change the distance of a gene with respect to the replication terminus and origin and also its orientation with respect to the replication direction.

permutations π and π_T, taking into account the length of the markers. For example, if we have a permutation $\pi = \{1, 2, 3, 4, 5\}$, with the respective lengths 1, 1, 1, 3 and 2, then the replication

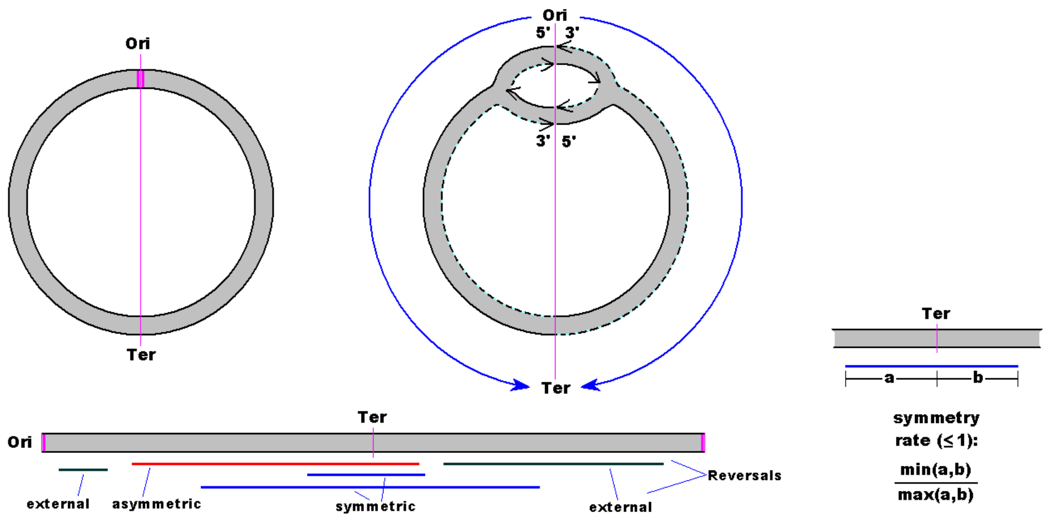


Figure 24: A threshold on the terminus-symmetry rate determines whether a reversal is terminus-symmetric or terminus-asymmetric.

terminus is inferred to be at the position given by $1+1+1+3+2/2 = 4$, that is, the replication terminus is placed within the marker 4 (Figure 25).

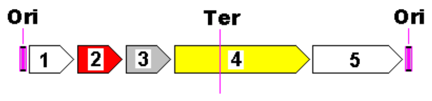


Figure 25: The inferred position of the replication terminus in a circular chromosome. The circular chromosome is represented by the permutation $\pi = \{1, 2, 3, 4, 5\}$*, with the respective lengths* 1, 1, 1, 3 *and* 2*. The replication terminus is inferred to be at the position given by* $1 + 1 + 1 + 3 + 2/2 = 4$*, thus the replication terminus is placed within the marker* 4*.*

With these assumptions, we are able to detect whether a reversal is external or not. If a reversal is not external, we can compute its terminus-symmetry rate and determine whether it is terminus-symmetric or terminus-asymmetric. We can use the maximum number of external and terminus-asymmetric reversals as a constraint to adapt the algorithm that generates the traces of sorting sequences. Given a threshold r to the terminus-symmetry rate and two integers p and q, we search for the sequences sorting π into π_T that contain at most p external and at most q terminus-asymmetric reversals, according to the threshold r to the terminus-symmetry rate. Observe that if a reversal ρ has a status X in a sequence of reversals s sorting π into π_T, there is no guarantee that ρ has also status X in a sequence s' that is equivalent to s (X can be external, terminus-symmetric or terminus-asymmetric). An example is given in Figure 26.

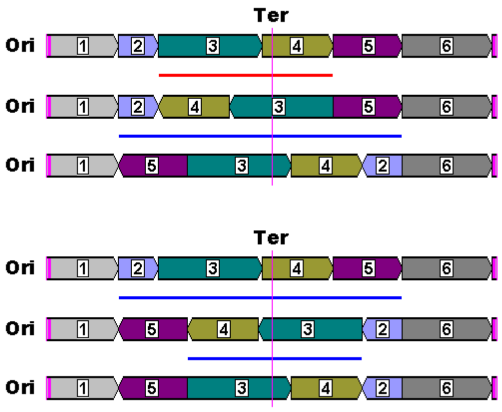


Figure 26: Two equivalent optimal sequences of reversals sorting the circular permutation $\mathcal{I}_6$ *into* $(1, -5, 3, 4, -2, 6)$. *While in the first sequence we have a terminus-asymmetric reversal (*$\{3, 4\}$*) followed by a terminus-symmetric reversal (*$\{2, 3, 4, 5\}$*), in the second sequence both reversals are terminus-symmetric.*

Thus, frequently only a subset of the sequences in a trace T are in agreement with the constraint (p, q, r) and this subset is called a $(p, q, r)-$induced subtrace. The adapted algorithm generates directly the $(p, q, r)-$induced subtraces of sequences sorting π into π_T. Applying the constraint (p, q, r) is a symmetric approach: if a sequence s that sorts π into π_T is in agreement with the constraint (p, q, r), then $inv(s)$, that sorts π_T into π, is also in agreement with (p, q, r) (see Figure 27).

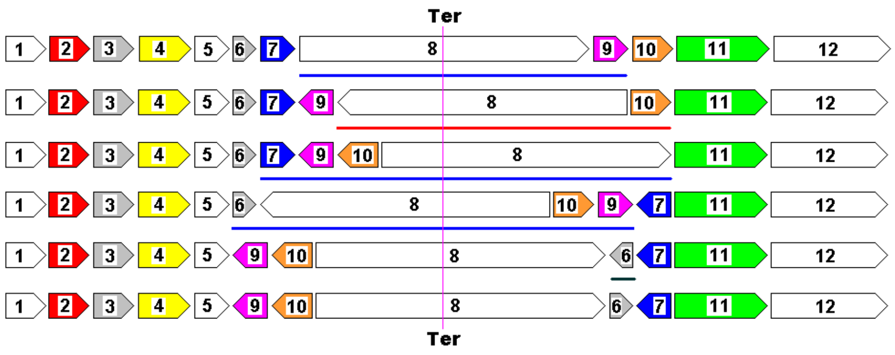


Figure 27: An example of one terminus-asymmetric, one external and three terminus-symmetric reversals in a genome with 12 markers. Observe that each reversal has the same status, independently of applying the sequence of five reversals from top to bottom or from bottom to top, thus the terminus-symmetry itself is a symmetric approach.

This method was combined to the progressive detection of common intervals to analyze the evolution of *Rickettsia* bacterium, as we will see in the next section.

5.3.1 Analysis of the *Rickettsia* bacterium

We used our methods with two biological constraints, the common intervals progressively detected (described in the previous section) and the terminus-symmetry, to analyze the evolutionary scenario of the *Rickettsia* bacterium.

Rickettsia is a group of obligate intracellular parasites, that is, microorganisms that cannot live outside a host cell. *Rickettsia* species are carried as parasites by many vectors, that are frequently hematophagous arthropods, such as ticks, fleas, and lice. The parasites are occasionally transmitted from the vector to mammalians (including humans), causing several diseases (typhus, spotted fever, etc) [47, 48].

The genomes of intracellular parasites such as *Rickettsia* are observed to have a reductive evolution, that is, the process by which genomes shrink and undergo extreme levels of gene degradation and loss [2]. There are several completely sequenced *Rickettsia* genomes, and most of them are closely related[10]. Recently, Blanc *et al.* [13] studied the evolutionary scenario of six *Rickettsia* species and reconstructed their ancestors $R1$, $R2$, $R3$, $R4$ and $R5$ (represented in Figure 28).

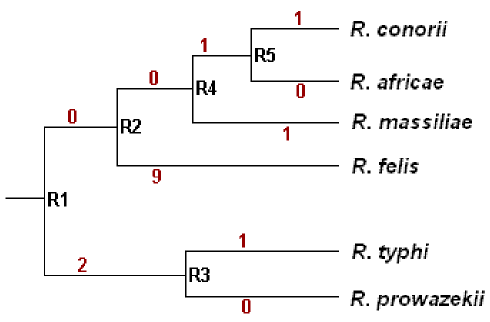


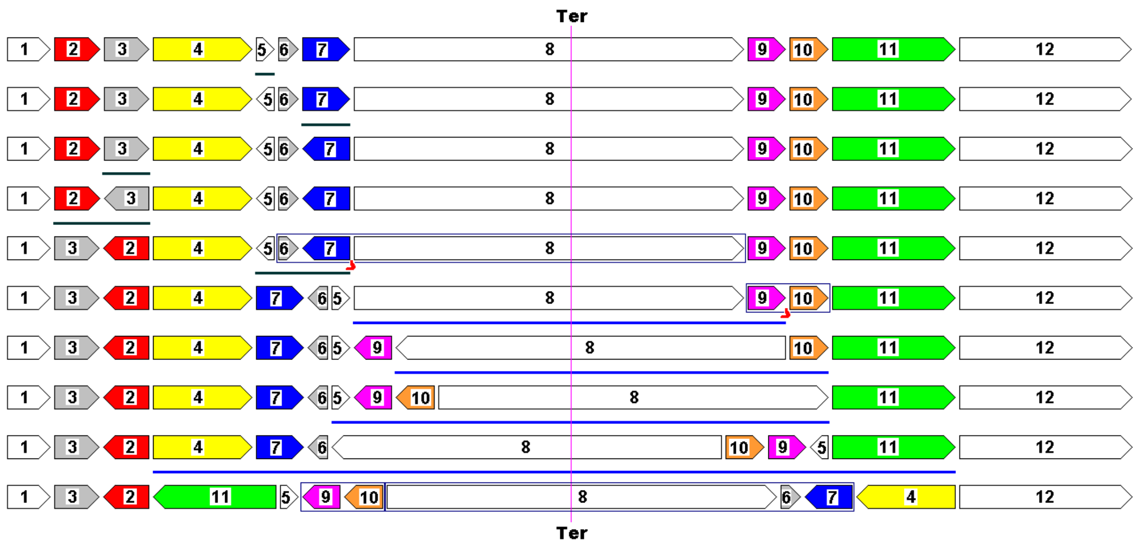
Figure 28: Phylogenetic tree of six Rickettsia (extracted from [13]). The numbers on the edges give the reversal distance between the genomes on the vertices, which could be either a current species or an ancestor ($R1$, $R2$, $R3$, $R4$ and $R5$).

In particular, the ancestor $R2$ was compared to *Rickettsia felis*. According to the reconstruction of $R2$ done by Blanc *et al.* [13], these genomes have 12 blocks of contiguous homologous genes, mapped as the permutation $(1, 3, -2, -11, 5, -9, -10, 8, 6, -7, -4, 12)$ for *R. felis*, and the identity permutation $\mathcal{I}_{12}$ for $R2$. Blanc *et al.* [13] used the software `GRIMM` [64] to propose

[10] A database of *Rickettsia* genomes is available on-line at `http://www.igs.cnrs-mrs.fr/mgdb/Rickettsia`.

an optimal sequence of reversals to transform $R2$ into *Rickettsia felis* (see Figure 29) and used this one sequence in their results, including to account for the number of terminus-symmetric reversals in the whole analysis. However, several other sequences exist and were not examined. The reversal distance between these two genomes is equal to 9, and the complete analysis of the traces of sequences sorting *R. felis* into $R2$ resulted in $546,840$ sorting sequences, distributed in 13 traces (Table 12). We did the analysis from *R. felis* to the ancestor $R2$ due to the further use of the progressively detected common intervals to constrain the traces of sorting sequences. The sequence of reversals proposed by Blanc *et al.* [13] sorts $R2$ into *R. felis* with five external and four terminus-symmetric reversals (Figure 29), and is the inverse of a sequence of trace 1 in Table 12.

$R2$ (ancestor)



Rickettsia felis (descendant)

Figure 29: An optimal sequence of reversals to transform the ancestor $R2$ into Ricketsia felis, with five external and four terminus-symmetric reversals (proposed in [13]). The two common interval breaks are indicated by the red signs.

First, we applied only the progressive detection of common intervals. To compute the universe of optimal sequences sorting *Rickettsia felis* into $R2$, taking into account the progressively detected common intervals, we had to relax the constraint to accept two interval breaks, because the result of searching for progressive perfect subtraces that do not break any common interval or that break one common interval per sorting sequence is empty. Accepting two interval breaks per

Trace	Trace normal form	# seq.
1.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5\}\{5,8,9,10\}\{7\}\{8,10\} < \{5,6,7\}\{8,9\}$	90,720
2.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\} < \{6,\ldots,9\}\{7,8\}$	90,720
3.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,8,9,10\}\{8,10\} < \{7,\ldots,10\}\{8,9\}$	90,720
4.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6,7,8,10\}\{7\}\{9\} < \{6,7,9\} < \{8,9\}$	60,480
5.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6,8\}\{9\}\{10\} < \{6,9,10\} < \{7,\ldots,10\}$	60,480
6.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{7\}\{8,10\}\{10\} < \{6,7,10\} < \{6,\ldots,9\}$	60,480
7.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,9,10\}\{7\}\{9\}\{10\} < \{5,8\} < \{5,6,7\}$	60,480
8.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,8,9,10\}\{5,9,10\}\{7\} < \{5,6,7,9,10\} < \{6,7,8,10\} < \{6,\ldots,9\}$	9,072
9.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,8,9,10\}\{6\}\{8,10\} < \{5,6,8,9\} < \{5,7,8,9\} < \{6,\ldots,9\}$	6,048
10.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,9,10\}\{6,8\}\{10\} < \{5,6,9\} < \{5,7,8,9\} < \{6,\ldots,9\}$	6,048
11.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{6\}\{6,8,9,10\} < \{5,6,8,10\} < \{5,6,8,9\} < \{5,\ldots,8\}\{7,8\}$	6,048
12.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{5\}\{6,8,9,10\} < \{5,6,8,10\} < \{5,7,9\} < \{6,7,9\} < \{8,9\}$	3,024
13.	$f = \{2,3\}\{3\}\{4,\ldots,11\}\{6,8\} < \{6,9,10\}\{7,8\} < \{5,6,10\} < \{5,6,9\} < \{5,\ldots,8\}$	2,520
	Total	546,840

Table 12: The 546,840 possible sequences of reversals for transforming Rickettsia felis, represented by the permutation $Rfe = (1,3,-2,-11,5,-9,-10,8,6,-7,-4,12)$, into the ancestor $R2 = \mathcal{I}_{12}$ are distributed in 13 traces. Each trace is represented by its normal form. The third column indicates the number of sequences in each trace.

sorting sequence, more than half of the sorting sequences and traces from the complete solution space is discarded (see the results in Table 13).

We observed that the sequence proposed in [13] (Figure 29) was selected by the construction of progressive near-perfect subtraces accepting two common interval breaks per sequence (it is the inverse of a sequence in subtrace A of Table 13). However, there are still many other possibilities that have the same score with respect to progressively detected common interval breaks.

Then we applied only the terminus-symmetry constraint, also from the descendant *Rickettsia felis* to the ancestor $R2$, in order to be able to compare the results with the previous analysis with progressive detection of common intervals (since the terminus-symmetry is itself a symmetric constraint, the analyses done from the descendant to the ancestor and from the ancestor to the descendant lead to symmetric results). We tested different values for the parameters p (the maximum number of external reversals), q (the maximum number of terminus-asymmetric

Sub	**Trace**	**Subtrace 2-tuple**	**# seq.**
A	1.	$e = \{4,\ldots,11\}\{5,8,9,10\}\{8,10\}\{8,9\}\{5,6,7\}\{2,3\}\{3\}\{7\}\{5\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5\}\{5,8,9,10\}\{7\}\{8,10\} < \{5,6,7\}\{8,9\}$	$45,360$
B	2.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\}\{6,\ldots,9\}\{7,8\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\} < \{6,\ldots,9\}\{7,8\}$	$45,360$
C	3.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,8,9,10\}\{7,\ldots,10\}\{8,10\}\{8,9\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,8,9,10\}\{8,10\} < \{7,\ldots,10\}\{8,9\}$	$45,360$
D	4.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6,7,8,10\}\{7\}\{9\}\{6,7,9\}\{8,9\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6,7,8,10\}\{7\}\{9\} < \{6,7,9\} < \{8,9\}$	$60,480$
E	7.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{5,9,10\}\{7\}\{9\}\{10\}\{5,8\}\{5,6,7\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,9,10\}\{7\}\{9\}\{10\} < \{5,8\} < \{5,6,7\}$	$60,480$
F	11.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{6\}\{6,8,9,10\}\{5,6,8,10\}\{5,6,8,9\}\{5,\ldots,8\}\{7,8\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{6\}\{6,8,9,10\} < \{5,6,8,10\} < \{5,6,8,9\} < \{5,\ldots,8\}\{7,8\}$	$6,048$
		Total	$263,088$

Table 13: The 6 subtraces containing 263,088 possible sequences of reversals for transforming the descendant Rickettsia felis into the ancestor $R2$ with progressive detection of common intervals, accepting at most two common interval breaks. Each progressive near-perfect subtrace is represented by a $2-$tuple (e is the subtrace representative, f is the trace normal form). The second column indicates the corresponding trace in Table 12. The fourth column gives the number of sequences in each subtrace.

reversals) and r (the terminus-symmetry threshold). The higher level of r and lowest values of p and q that gave a non-empty result are $p = 3$, $q = 0$ and $r = 0.7$ (see the results of the analysis with these parameters in Table 14).

Searching for $(3, 0, 0.7)-$induced subtraces revealed the existence of several optimal sorting sequences that have six terminus-symmetric reversals, while the sequence proposed by Blanc *et al.* [13] has only four (see Figure 29). We also observe that three over thirteen traces have sequences that were selected in both analyses, that is, progressive near-perfect subtraces accepting two interval breaks per sequence and $(3, 0, 0.7)-$induced subtraces (see Tables 13 and 14).

In order to determine whether each pair of subtraces (B, B'), (C, C') and (F, F'), has a non-empty intersection, we combined both constraints in the same analysis, searching directly for subtraces whose sequences are progressive near-perfect sequences with two interval breaks, composed by at most 3 external and no terminus-asymmetric reversals, according to a threshold of 0.7 to the terminus-symmetry rate. The results are given in Table 15.

With these results, we can say that, when sorting *Rickettsia felis* into the ancestor $R2$, $54,936$ over $546,840$ optimal sorting sequences have six terminus-symmetric reversals (considering a

Sub'	Sub	Trace	Subtrace 2-tuple	# seq.
B'	B	2.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\}\{6,\ldots,9\}\{7,8\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\} < \{6,\ldots,9\}\{7,8\}$	$60,480$
C'	C	3.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{6,8,9,10\}\{8,10\}\{5,\ldots,10\}\{6\}\{7,\ldots,10\}\{8,9\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,8,9,10\}\{8,10\} < \{7,\ldots,10\}\{8,9\}$	$50,904$
F'	F	11.	$e = \{2,3\}\{3\}\{6,8,9,10\}\{5,6,8,10\}\{5,6,8,9\}\{5,\ldots,8\}\{4,\ldots,11\}\{6\}\{7,8\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{6\}\{6,8,9,10\} < \{5,6,8,10\} < \{5,6,8,9\} < \{5,\ldots,8\}\{7,8\}$	$1,512$
G'		9.	$e = \{2,3\}\{3\}\{8,10\}\{5,8,9,10\}\{5,6,8,9\}\{5,7,8,9\}\{4,\ldots,11\}\{6\}\{6,\ldots,9\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,8,9,10\}\{6\}\{8,10\} < \{5,6,8,9\} < \{5,7,8,9\} < \{6,\ldots,9\}$	$1,008$
			Total	$113,904$

Table 14: The 4 subtraces containing 113,904 possible sequences of reversals for transforming the descendant Rickettsia felis into the ancestor $R2$ *with terminus-symmetry contraint* (p,q,r), *where* $p = 3$ *(the maximum number of external reversals),* $q = 0$ *(the maximum number of terminus-asymmetric reversals) and* $r = 0.7$ *(the threshold for the terminus-symmetry rate). Each* $(p,q,r)-$*induced subtrace is represented by a* $2-$*tuple (*e *is the subtrace representative,* f *is the trace normal form). The second column indicates the corresponding progressive near-perfect subtrace in Table 13. The third column indicates the corresponding trace in Table 12. The fifth column gives the number of sequences in each subtrace. All the 113904 sequences selected with this criteria have six terminus-symmetric reversals, while the sequence proposed by Blanc et al. [13] has only four (see Figure 29).*

SUB	Trace	Subtrace 2-tuple	# seq.
B ∩ B'	2.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\}\{6,\ldots,9\}\{7,8\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,7,8,10\}\{6,8\} < \{6,\ldots,9\}\{7,8\}$	$28,224$
C ∩ C'	3.	$e = \{2,3\}\{3\}\{4,\ldots,11\}\{6,8,9,10\}\{8,10\}\{5,\ldots,10\}\{6\}\{7,\ldots,10\}\{8,9\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{5,\ldots,10\}\{6\}\{6,8,9,10\}\{8,10\} < \{7,\ldots,10\}\{8,9\}$	$25,200$
F ∩ F'	11.	$e = \{2,3\}\{3\}\{6,8,9,10\}\{5,6,8,10\}\{5,6,8,9\}\{5,\ldots,8\}\{4,\ldots,11\}\{6\}\{7,8\}$ $f = \{2,3\}\{3\}\{4,\ldots,11\}\{6\}\{6,8,9,10\} < \{5,6,8,10\} < \{5,6,8,9\} < \{5,\ldots,8\}\{7,8\}$	$1,512$
		Total	$54,936$

Table 15: The 3 subtraces containing $54,936$ *possible sequences of reversals for transforming the descendant Rickettsia felis into the ancestor* $R2$ *with terminus-symmetry contraint* (p,q,r), *where* $p = 3$ *(the maximum number of external reversals),* $q = 0$ *(the maximum number of terminus-asymmetric reversals) and* $r = 0.7$ *(the threshold for the terminus-symmetry rate), and progressive detection of common intervals, accepting two interval breaks per sequence. Each subtrace is represented by a* $2-$*tuple (*e *is the subtrace representative,* f *is the trace normal form). The second column indicates the corresponding trace in Table 12. The fourth column gives the number of sequences in each subtrace. All the* $54,936$ *sequences selected with this criteria have six terminus-symmetric reversals, while the sequence proposed by Blanc et al. [13] have only four (see Figure 29).*

threshold of 0.7 for the terminus-symmetry rate) and break two common intervals progressively detected. Since the sequence proposed by Blanc *et al.* [13] also breaks two common intervals progressively detected, but have only four terminus-symmetric reversals, we propose an alternative sorting sequence, which is the inverse of a sequence extracted from the subtrace in the first line of Table 15 (see Figure 30).

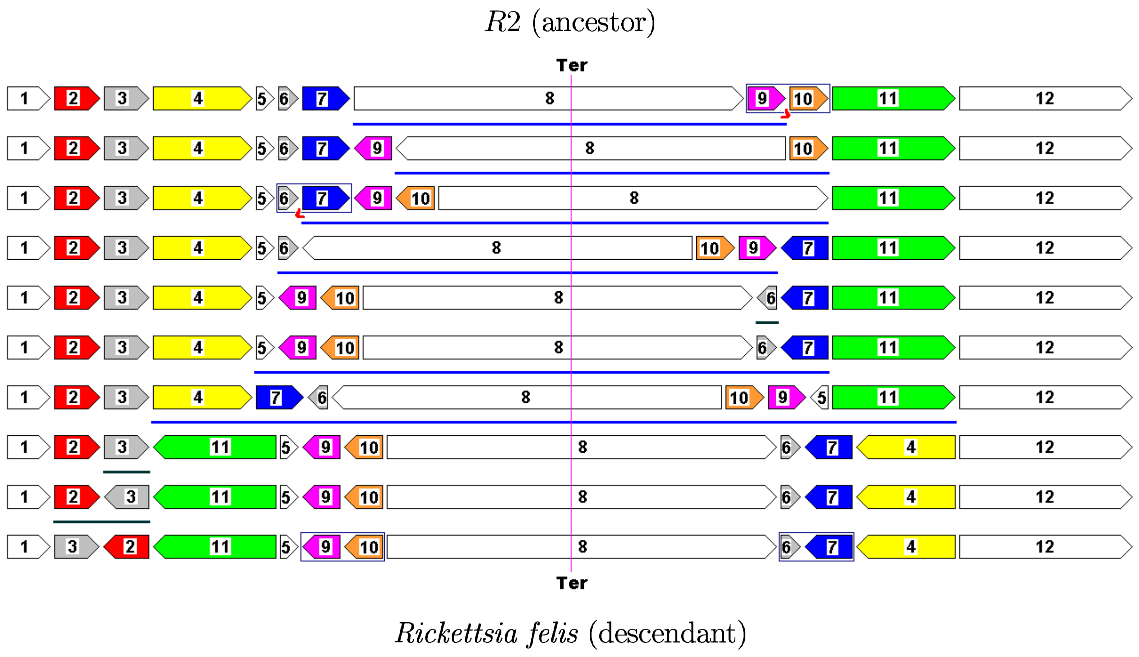


Figure 30: An alternative optimal sequence of reversals to transform the ancestor R2 into Ricketsia felis, with three external and six terminus-symmetric reversals, while the sequence proposed by Blanc et al. [13] has five external and only four terminus-symmetric reversals (see Figure 29). The two common interval breaks are indicated by the red signs.

5.3.2 Evaluating the execution time

Our experimental results show that the execution time of the variant of the program that uses the terminus symmetry as a constraint to compute traces is compatible with other variants, as we can see in Table 16.

5.4 Stratification on evolution of sexual chromosomes

We also applied our method to analyze the evolution of the human X and Y chromosomes. These chromosomes are very different, and, while the X chromosome is 155 Mbps long, the Y chromosome is 58 Mbps long. Nevertheless, both are believed to have evolved from an identical autosomal pair[11] [50]. This process is at the origin of sexual differentiation: the female XX and the male XY pairs. Due to the recombination mechanism, female organization favours

[11] Autosomes are all non-sex chromosomes.

Input	Algorithm	N_S	N_T	Execution time
$R.fe \to R2$	`all traces`	546,840	13	$\simeq$ 3 seconds
	`prog. near-2-perf. subtraces`	263,088	6	$\simeq$ 2 seconds
	$(3,0,0.7)-$`induced subtraces`	113,904	4	$\simeq$ 2 seconds
	$(3,0,0.7)-$`ind.+prog.near-2-perf.subt.`	54,936	3	$\simeq$ 2 seconds
Test	`all traces`	3,089,198,988	60,063	$\simeq$ 25.2 minutes
	`prog. near-1-perf. subtraces`	157,903,560	164	$\simeq$ 2 minutes
	$(2,3,0.8)-$`induced subtraces`	2,976	3	$\simeq$ 1.7 minutes
	$(2,3,0.8)-$`ind.+prog.near-1-perf.subt.`	0	0	$\simeq$ 21 seconds

Table 16: Computation results. Columns from left to right contain: 1- the input permutation 2- the algorithm 3- the number of sequences according to the algorithm; 4- the number of traces according to the algorithm; 5- the execution time of the algorithm. All algorithms are part of the BAOBABLUNA *package. Circular permutation $R.fe \to R2$ is* $(1,3,-2,-11,5,-9,-10,8,6,-7,-4,12)$ *with lengths* 39*,* 42*,* 44*,* 125*,* 14*,* 30*,* 32*,* 391*,* 18*,* 47*,* 100*, and* 176*. Circular permutation $Test$ is* $(-12,11,-10,6,-5,13,2,7,8,-9,14,-15,3,4,-16,1)$.

conservation of the X chromosome. On the other hand, evolution of the male XY pair causes the divergence of the Y chromosome, as it gradually loses the capacity of recombining with its X partner.

The X and Y chromosomes still share a main "pseudo-autosomal" region at one of their extremities, where recombination occurs as between autosomes. Ninety percent of the Y chromosome is however male-specific, and shows major differences in sequence as well as in gene order with the X. Current theories suggest that the pseudo-autosomal region, which originally covered the whole chromosomes, was successively pruned by a few big reversals on the Y chromosome [40], whose extremities stood on each side of the limit of the pseudo-autosomal region. The successive limits of the pseudo-autosomal region on the X chromosome, from the origin to where it is located now, represent the limits of what have been called the "evolutionary strata" of the sex chromosomes.

Several indices seem to indicate the presence of at least five strata on the X chromosome [55, 60]. The strata are ordered according to their creation time. Thus, the stratum that is the closest to the pseudo-autosomal region is numbered 5, while the stratum which is at the other extremity of the X chromosome is numbered 1. A sequence of reversals on a signed permutation representing the relative ordering of the genes common to chromosomes X and Y, obtained thanks to the software `GRIMM` [64], has been published in study of Ross *et al.* [55], and is given as an argument to support the existence and bounds of the most recent strata. The sequence is represented in Figure 31.

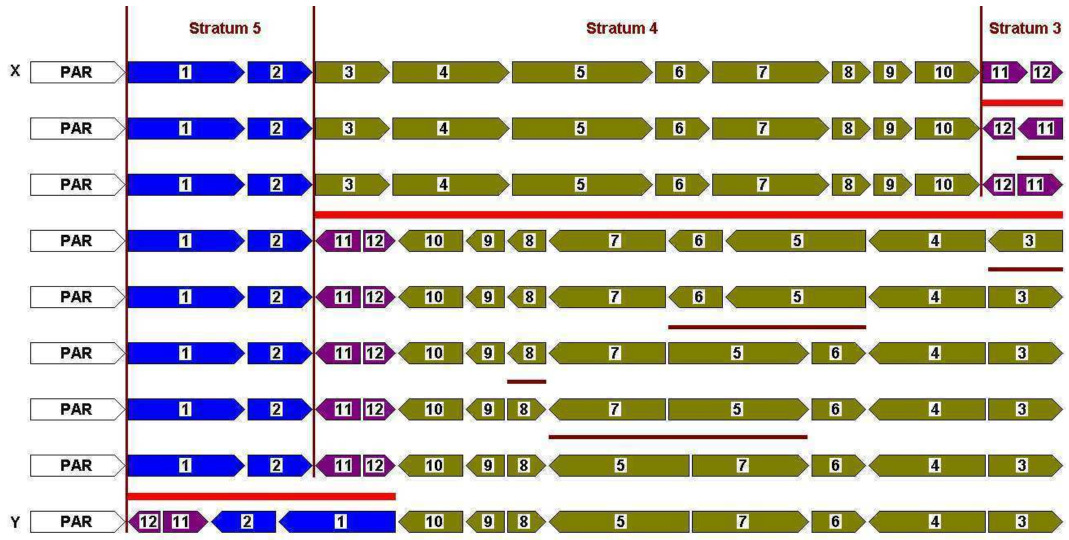


Figure 31: Sequence of reversals transforming human X into human Y chromosome, that shows the formation of the last three strata (numbered 3, 4 and 5) on X chromosome (extracted from Ross et al. [55]). The PAR symbol represents the pseudo-autosomal region in each chromosome.

However for the same permutation, there are many sequences that are possible, including others sequences that are in agreement with a model of evolution by strata, which we now describe.

5.4.1 Model of evolution by strata

For a signed permutation $X = (X_1, \dots, X_n)$, a $k-$*strata* is defined as a partition of X into a sorted set $B = (I_k, I_{k-1}, \dots, I_1)$ of k intervals, such that $I_k = \{|X_1|, \dots, |X_{n_k}|\}$, $I_{k-1} = \{|X_{n_k+1}|, \dots, |X_{n_k+n_{k-1}}|\}$, ..., $I_1 = \{|X_{n_k+\dots+n_2+1}|, \dots, |X_{n_k+\dots+n_1}|\}$, where n_i is the size of the interval I_i. Observe that the intervals are ordered by their positions, but they are indexed in a decreasing way from the beginning to the end of the permutation. We define a $B-$*stratifying sequence of reversals* as follows.

Definition 1 *Given a signed permutation $X = (X_1, \dots, X_n)$ and a $k-$strata $B = (I_k, I_{k-1}, \dots, I_1)$, we say that a sequence of reversals $r = \rho_1\rho_2 \dots \rho_d$ is a $B-$*stratifying sequence *if:*

1. *The sequence r has a subsequence*[12] *$b = \theta_1\theta_2 \dots \theta_k$, such that for $1 \leq i \leq k$, the reversal θ_i*

[12] Recall that a “subsequence” b of a sequence r is obtained by eliminating some of the elements (here reversals) of r while preserving the order of the remaining elements.

contains the interval I_i and, for any $j > i$, no element of I_j is in θ_i.

2. *For any two consecutive reversals θ_i and θ_{i+1} of b, if ρ is a reversal that occurs between θ_i and θ_{i+1} in r, then ρ is a subset of $I_1 \cup I_2 \ldots \cup I_i$.*

The reversals in b are said to be *big reversals* (each big reversal creates a new stratum), while the reversals of r that are not in b are said to be *small reversals*. A sequence of reversals that produces a $k-$strata has k big reversals and $d-k$ small reversals (we recall that d is the reversal distance for the given permutation).

Consider a permutation X, a $k-$strata $B = (I_k, I_{k-1}, \ldots, I_1)$ for X and a target permutation Y. If T is a trace of optimal sequences of reversals sorting X into Y, we call $B-$*induced subtrace* T_B the subset of T defined as $T_B = \{s | s \in T$ and s produces the $k-$strata B in $X\}$.

This approach is conceptually asymmetric, since the stratification is supposed to take place in the ancestor genome. Observe that actually X chromosome is not the ancestor of Y, but it is assumed to be the ancestor state of Y, at least for the analyzed portion. Indeed, it has been observed that a considerable portion of X chromosome is highly similar to a portion of chromosome 1 of chicken [55], thus this region, that coincides with strata 5 and 4 and a part of stratum 3, is assumed to be free of rearrangements after the differentiation of sexual chromosomes in humans. Thus, the analysis should be done from the stratified (ancestor) genome to the other (descendant), and reflects directly the real chronology of the events (remember that the asymmetric progressive perfect sequences should be inverted to reflect the real chronology).

5.4.2 Algorithm for exploring the sequences that stratify a permutation

We have developed a version of our exploration algorithm that, given a $k-$strata B, outputs the set of traces whose sequences produce B. This requires a slight modification of Algorithm 5, described as follows.

Besides the two signed permutations X and Y, the modified algorithm requires a $k-$strata $B = (I_k, I_{k-1}, \ldots, I_1)$ for X. The algorithm returns the traces whose $B-$induced subtraces are not empty, and, for each trace, the size of its $B-$induced subtrace.

It is the first step (Siepel's step) that is mainly modified. After searching all next reversals, we must select only those that are in agreement with the given $k-$strata: the first reversal is fixed, and corresponds exactly to the first stratum (it is a big reversal); then, at each step, suppose stratum I_z has been moved by a big reversal and not stratum I_{z+1}; we can choose between performing a big reversal including I_{z+1} and no elements from the following ones, or a small reversal, included in $I_1 \cup \cdots \cup I_z$. The procedure is described in Algorithm 7.

At the end of the execution, we have the complete set of non-empty $B-$induced subtraces (each subtrace is represented by a 2-tuple containing the normal form of the corresponding trace and a valid $B-$stratifying sequence as a representative) and their sizes for a given permutation and a $k-$strata B. There is no guarantee that a strata-induced subtrace exists, thus the algorithm can lead to an empty result (an empty result means that it is not possible to produce the proposed stratification in the genome by an optimal sequence of reversals).

Theoretical complexity of Algorithm 7. The complexity is the same of Algorithm 5: since the position of a stratum in the genome remains unchanged until the stratum is created, to select a reversal we only need to compare its boundaries to the boundaries of the next stratum to be created, which takes constant time. Thus, the selection step can be done in time $O(n^2)$ since the number of reversals returned by Siepel's algorithm is $O(n^2)$ (n is the size of the permutation). The number of selected big and small reversals is also bounded by n^2. □

5.4.3 Analysis of the results

We applied our modified algorithm to the permutations $X = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)$ and $Y = (-12, 11, -2, -1, -10, -9, 8, -5, 7, 6, -4, 3)$, derived from the genes that are shared by the last three strata of the human X and Y chromosomes (Figure 31). It was used in a study of Ross *et al.* [55] to account for the positions of the strata $3, 4, 5$ in the human sex chromosomes, by giving one optimal sorting sequence of reversals.

We are now able to handle the whole set of sorting sequences: the solution space of sorting this permutation by reversals contains $31,752$ sequences, distributed among 6 traces (see Table 17).

When we search only for the sequences of reversals that sort X into Y and respect the formation of the last three strata as they are defined by Ross *et al.* [55], that is, $B_{xy} = (\{1, 2\}, \{3, \ldots, 10\}, \{11, 12\})$, we get 420 sequences, corresponding to a unique $B_{xy}-$induced subtrace, represented by the $2-$tuple (e, f) where e is a valid stratifying sequence and f is the

Algorithm 7 Enumerating all the strata-induced subtraces of two signed permutations and a given $k-$strata in the origin permutation

Input: Two signed permutations X, Y and a $k-$strata $B = (I_k, I_{k-1}, \dots, I_1)$ for X

Output: The representative, normal form and counter (e, f, c) of each $B-$induced subtrace of sequences of reversals sorting X into Y

```
$d \leftarrow$ reversal distance of $(X, Y)$
$\mathcal{T} \leftarrow \emptyset$
$S \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $X \rightarrow Y\}$   [Siepel [59]]
if $S$ contains a reversal $\rho = I_1$   [the first reversal may create the first stratum]   then
  insert $(\rho, \rho, 1)$ in $\mathcal{T}$   [the $1-$sequence that creates the first stratum is a $B-$induced $1-$subtrace]
end if
for each integer $i$ from 2 to $d$ do
  $\mathcal{T}' \leftarrow \emptyset$   [contains the representatives/normal forms/counters of all the $B-$induced $i-$subtraces]
  for each $(e, f, c)$ in $\mathcal{T}$   [$(e, f)$ repr. the $B-$induced $(i-1)-$subtrace $t$; $c$ is the counter of $t$]   do
    $X' \leftarrow X$
    $z \leftarrow 1$   [to store the index of the last created stratum]
    for each reversal $\rho$ in $e = \rho_1 \rho 2 \dots \rho_{i-1}$ do
      $X' \leftarrow X' \circ \rho$   [apply the reversal $\rho$ to $X'$]
      $w \leftarrow$ the biggest index of a stratum in $I_k, I_{k-1}, \dots, I_z$ such that $\rho$ overlaps $I_w$
      $z \leftarrow w$
    end for
    $b \leftarrow k - z$   [number of remaining big reversals]
    $S' \leftarrow \{\rho \mid \rho$ is an optimal $1-$sequence for $X' \rightarrow Y\}$   [Siepel [59]]
    for each $1-$sequence $\rho \in S'$ do
      $accept \leftarrow false$
      if $\rho$ contains the interval $I_{z+1}$ and $\rho$ does not overlap an interval in $I_{z+2}, I_{z+3}, \dots, I_k$ then
        $accept \leftarrow true$   [$\rho$ is the next big reversal]
      else
        if $d - i + 1 - b > 0$   [test whether it can apply a small reversal]   and $\rho$ does not overlap an
        interval in $I_{z+1}, I_{z+2}, \dots, I_k$ then
          $accept \leftarrow true$   [$\rho$ is a small reversal]
        end if
      end if
      if $accept$ then
        $f_\rho \leftarrow f + \rho$   [extend the normal form $f$ by adding the reversal $\rho$; see Algorithm 3]
        if there exists $(e', f', c') \in \mathcal{T}'$ such that $f' = f_\rho$   [COMPARISON]   then
          $c' \leftarrow c' + c$   [upd. the counter of the $B-$induced $i-$subtrace $t'$ repr. by $(e', f')$]
        else
          $e_\rho \leftarrow e \cdot \rho$   [simply concatenate $\rho$ to the sequence $e$]
          insert $(e_\rho, f_\rho, c)$ in $\mathcal{T}'$   [$(e_\rho, f_\rho)$ repr. the $B-$ind. $i-$subtr. $t_\rho$; $c$ is the initial counter of
          $t_\rho$]
        end if
      end if
    end for
  end for
  $\mathcal{T} \leftarrow \mathcal{T}'$
end for
return $\mathcal{T}$   [$\mathcal{T}$ is the final set of $B-$induced $d-$subtraces sorting $X$ into $Y$]
```

Trace	Trace normal form	# seq.
1.	$f = \{3\}\{3,\dots,12\}\{5,6\}\{8\}\{11\}\{11,12\} < \{1,2,11,12\}\{5,7\}$	$10,080$
2.	$f = \{1,2,3\}\{1,\dots,12\}\{4,\dots,10\}\{5,6\}\{8\}\{11\} < \{1,2,4,\dots,10\}\{5,7\}$	$10,080$
3.	$f = \{1,\dots,11\}\{1,\dots,12\}\{3,\dots,11\}\{3\}\{5,6\}\{8\} < \{1,2,11\}\{5,7\}$	$10,080$
4.	$f = \{3,\dots,11\}\{4,\dots,10\}\{5,6\}\{8\} < \{3,12\}\{5,7\} < \{4,\dots,10,12\} < \{1,2,11,12\}$	840
5.	$f = \{5,6\}\{8\}\{11,12\} < \{4,\dots,10,12\}\{5,7\} < \{1,2,3,12\} < \{1,\dots,11\}\{1,2,4,\dots,10\}$	336
6.	$f = \{1,\dots,11\}\{5,6\}\{8\} < \{1,2,3,12\}\{5,7\} < \{4,\dots,10,12\} < \{1,2,4,\dots,10\}\{11,12\}$	336
	Total	$31,752$

Table 17: The $31,752$ possible sequences of reversals for transforming $X = \mathcal{I}_{12}$ into $Y = (-12,11,-2,-1,-10,-9,8,-5,7,6,-4,3)$ are distributed in 6 traces. Each trace is represented by its normal form. The third column indicates the number of sequences in each trace.

normal form of its corresponding trace, as we can see in Table 18.

Trace	Subtrace 2-tuple	# seq.
1.	$e = \{11,12\}\{11\}\{3,\dots,12\}\{3\}\{5,6\}\{8\}\{5,7\}\{1,2,11,12\}$ $f = \{3\}\{3,\dots,12\}\{5,6\}\{8\}\{11\}\{11,12\} < \{1,2,11,12\}\{5,7\}$	420

Table 18: The $B_{xy}-$induced subtrace of optimal sequences that sort the chromosome $X = \mathcal{I}_{12}$ into $Y = (-12,11,-2,-1,-10,-9,8,-5,7,6,-4,3)$ and produce the $3-$strata $B_{xy} = (\{1,2\},\{3,\dots,10\},\{11,12\})$ on X.

There are in consequence 420 sequences out of $31,752$ that support the bounds given in [55], and they are all in the same trace. Here, more relevant than the number of sequences is the fact that they are all part of a unique subtrace, which means that the reversals have been identified correctly. So if we suppose the bounds are known, the sequence given by Ross *et al.* [55] is accurate.

Nevertheless, the limits between strata may be not so clear, and one could be interested in testing other hypotheses. For instance, we could extend strata 4 incorporating to it the markers 1 and 2 which were part of stratum 3 in the previous analysis. According to this hypothesis, which could be biologicaly meaningful as well, we have $B = (\{1,2\},\{3,\dots,12\})$, and there are $2,520$ sequences in the $B-$induced subtrace (see Table 19).

Thus, using our algorithm we are able to evaluate the different hypotheses of stratification and find all the subtraces (that is, a representation of all sequences) that produce each stratification.

Trace	Subtrace 2-tuple	# seq.
1.	$e = \{3,\dots,12\}\{11,12\}\{11\}\{3\}\{5,6\}\{8\}\{5,7\}\{1,2,11,12\}$ $f = \{3\}\{3,\dots,12\}\{5,6\}\{8\}\{11\}\{11,12\} < \{1,2,11,12\}\{5,7\}$	2,520

Table 19: The $B-$induced subtrace of optimal sequences that sort the chromosome $X = \mathcal{I}_{12}$ into $Y = (-12, 11, -2, -1, -10, -9, 8, -5, 7, 6, -4, 3)$ and produce the $2-$strata $B = (\{1,2\}, \{3,\dots,12\})$ on X.

5.4.4 On the execution time of the strata variant

In order to evaluate the execution time of this modified algorithm, we ran both the original algorithm (that searches for all the traces) and the modified one (that searches for the strata-induced subtraces) over the previous permutation and over an extended permutation, also extracted from the evolution of human X and Y chromosomes. The results are presented in Table 20 and shows that searching for strata-subtraces runs much faster than searching for all traces.

Input	Algorithm	N$_S$	N$_T$	Execution time
XY	`all traces`	31,752	6	$\simeq$ `1.3 seconds`
	B_{xy}`-induced subtraces`	420	1	$\simeq$ `0.5 seconds`
$extXY$	`all traces`	316,793,943,648	87,983	$\simeq$ `5 hours`
	B`-induced subtraces`	608,343,606	2,284	$\simeq$ `4 min`

Table 20: Computation results. Columns from left to right contain: 1- the input (permutation and strata) 2- the algorithm 3- the number of sequences according to the algorithm; 4- the number of traces according to the algorithm; 5- the execution time of the algorithm. Both algorithms are part of the BAOBABLUNA*package. Permutations XY are $X = \mathcal{I}_{12}$ with $3-$strata $B_{xy} = (\{1,2\}, \{3,\dots,10\}, \{11,12\})$, and $Y = (-12, 11, -2, -1, -10, -9, 8, -5, 7, 6, -4, 3)$ (human X and Y chromosomes, as the scenario proposed in [55]). Permutations extXY are $X_{ext} = \mathcal{I}_{15}$ with $3-$strata $B = (\{1,2\}, \{3,\dots,7\}, \{8,\dots,15\})$, and the permutation $Y_{ext} = (14, -10, 8, -1, 9, 11, -7, 6, -4, 5, -3, 2, 15, 12, -13)$ (extended human X and Y chromosomes, adding markers to stratum 3).*

5.4.5 A deeper study of the human X and Y sexual chromosomes

The permutations representing the X and Y chromosomes as proposed by Ross *et al.* [55] cover only the first 11.2 Mbps on X, and even for this small portion of the chromosome there are alternative strata boundaries. Moreover, if we extend the permutations to cover a bigger portion of the chromosome, the number of possibilities for placing the strata boundaries increases. Indeed, only the boundary between the pseudo-autosomal region (PAR) and stratum 5 and the boundary between strata 5 and 4 are well established. The other boundaries are still controversial, and even the number of ancient strata is discussed.

We partipated in a collaborative work of Lemaitre *et al.* [41], that intends to go further in the study of these points (the submitted version of this study is included in Appendix D, in the end of this manuscript). In this work, our method for analyzing strata-induced subtraces, or more generally, our knowhow in analyzing sequences of reversals with respect to a genome stratification were applied to analyze extended permutations representing the chromosomes X and Y and as part of a probabilistic analysis to verify whether the occurrence of a sequence of reversals that stratify a genome is more likely to be true than the occurrence of a sequence of reversals that do not.

5.4.5.1 Analysis of strata in extended permutations

As we have seen, according to Ross et al [55], the first 11.2 Mbps of X chromosome has 12 homologous markers with the Y chromosome, resulting in the signed permutations $X = \mathcal{I}_{12}$ and $Y = (-12, 11, -2, -1, -10, -9, 8, -5, 7, 6, -4, 3)$ (see figure 31). With the strata $B_{xy} = (\{1,2\}, \{3,4,5,6,7,8,9,10\}, \{11,12\})$ on X, there is one B_{xy}−induced subtrace with 420 parsimonious stratifying sequences of reversals, that is, all sequences sorting X into Y according to the strata B_{xy} are in the same subtrace. So if we suppose that B_{xy} is accurate, the reversals in the sequence given by Ross *et al.* [55] were identified correctly.

Nevertheless, the stratum 3, as proposed in B_{xy}, is incomplete. In fact, to cover the stratum 3 entirely we should consider the first 45 Mbps of X chromosome and there is no guarantee that we can find the same strata boundaries if we extend the permutations. Although the boundaries between the pseudo-autosomal region (PAR) and the stratum 5 (the point before marker 1 in X permutation) and between strata 5 and 4 (the point between markers 2 and 3 in X permutation) are well established, the other boundaries (including the boundary between strata 4 and 3, that is present in the study of Ross *et al.* [55]) are still controversial. Moreover, since it is hard to recognize the most ancient rearrangement events, even the number of ancient strata is still under discussion.

Lemaitre *et al.* [41] analyzed the strata boundaries and found more evidences of the reversals that created strata 5 and 4, that reaffirm the estabished boundaries between PAR and stratum

5 and between strata 5 and 4, but were not able to clarify the boundary between strata 4 and 3. The authors revised the permutations representing X and Y proposed by Ross et al [55], using a whole chromosome alignment, and found out that, for the same 11.2 Mbps of X chromosome, there are three homologous markers that have not been considered by Ross *et al.* [55]. Extending the permutations to cover the first 45 Mbps of X chromosome, Lemaitre *et al.* [41] identified other 8 homologous markers. All markers considered in the analysis of Lemaitre *et al.* [41] are listed in Table 21.

With the extended permutations, Lemaitre *et al.* [41] considered in particular two hypotheses for placing the boundary between strata 4 and 3. The first is to maintain this boundary in the same position proposed by Ross *et al.* [55], that is, between markers KAL1 and TBL1 (see Table 21). The second is to put the boundary in the place occupied by the marker AMEL (there are biological evidences to justify this hypothesis [36], see details in Lemaitre *et al.* [41]). In both cases, our method for searching directly the strata-induced subtraces were applied. It was verified that it is not possible to find a reversal that creates the whole stratum 3 with the boundary as proposed by Ross *et al.* [55]. However, it is possible to create a shorter stratum with this boundary and this may suggest that the number of strata is higher than supposed. The same result was found for the analysis when the boundary between strata 4 and 3 was placed in the position occupied by AMEL. Lemaitre *et al.* [41] consider that this indeed indicates that the number of strata in the human X chromosome can be higher than 5. Moreover, due to several biological arguments [36, 41], the authors consider that placing the boundary between strata 4 and 3 in the position occupied by AMEL is an acceptable hypothesis.

5.4.5.2 Simulations to estimate the stratification likelihood

Lemaitre *et al.* [41] used our knowhow in analyzing sequences of reversals with respect to a genome stratification as part of a probabilistic analysis to verify whether the occurrence of a sequence of reversals that stratify a genome is more likely to be true than the occurrence of a sequence of reversals that do not. This approach is based on the distribution of all the sorting sequences with respect to the stratification process, and we helped to generate the information that was further used to compute the probabilistic values.

Marker name	X start	X end	Y start	Y end	X	Y	Str	Refs
PAR	0	2,709,520	0	2,709,520	0	0	PAR	Lahn & Page [40]
GYG*,ARSD*, ARSE*,ARSF**, ADLICAN** (1)	2,672,359	3,346,731	12,492,110	13,139,179	1	−6	5	*Lahn & Page [40] **Skaletsky *et al.* [60]
PRK (2)	3,345,018	3,848,954	7,068,601	7,506,089	2	−5	5	Lahn & Page [40]
Anonymous	3,662,755	3,909,738	19,743,211	19,851,471	3	−20	4	Lemaitre *et al.* [41]
Anonymous (3)	4,110,549	4,490,406	17,583,377	18,076,812	4	19	4	Ross *et al.* [55]
Anonymous (4)	4,602,689	5,384,111	16,706,206	17,570,219	5	−18	4	Ross *et al.* [55]
around NLGN4 (5)	5,384,848	6,313,029	14,981,290	15,805,945	6	−15	4	Skaletsky *et al.* [60]
Anonymous (6)	6,594,680	6,624,810	16,664,668	16,691,008	7	17	4	Ross *et al.* [55]
around STS (7)	6,625,496	7,448,677	15,807,027	16,376,778	8	16	4	Lahn & Page [40]
Anonymous (8)	7,449,397	7,646,086	14,794,314	14,971,778	9	14	4	Ross *et al.* [55]
around VC (9)	7,731,889	7,952,770	14,681,748	14,772,569	10	−13	4	Skaletsky *et al.* [60]
around KAL1 (10)	8,388,775	8,678,660	14,456,224	14,455,780	11	−12	4	Lahn & Page [40]
TBL1 (11)	9,367,582	9,694,004	6,818,075	7,040,054	12	4	3/4	Skaletsky *et al.* [60]
APXL	9,803,943	9,836,714	13,139,984	13,177,590	13	7	3/4	Skaletsky *et al.* [60]
Anonymous	9,925,052	10,026,743	2,935,524	6,736,276	14	2	3/4	Lemaitre *et al.* [41]
AMEL (12)	11,221,454	11,228,802	6,756,180	6,804,332	15	−3	3/−	Lahn & Page [40]
TMSB4	12,893,995	12,914,689	14,259,652	14,336,452	16	11	3	Lahn & Page [40]
TXNLG	16,713,573	16,773,411	20,187,740	20,234,258	17	22	3	Skaletsky *et al.* [60]
EIF1A	20,052,557	20,069,887	21,146,999	21,164,428	18	−23	3	Lahn & Page [40]
ZF	24,071,318	24,144,376	2,855,296	2,922,379	19	1	3	Lahn & Page [40]
MAP3/TAB3	30,755,480	30,819,301	13,771,944	13,828,537	20	9	3	Lemaitre *et al.* [41]
BCoR	39,795,364	39,917,376	20,076,630	20,184,596	21	21	3	Skaletsky *et al.* [60]
CRSP2P-CASK	40,392,502	41,667,660	13,240,309	13,592,325	22	8	3	Lahn & Page [40]
UT	44,617,701	44,856,791	13,869,035	14,101,947	23	−10	3	Lahn & Page [40]

Table 21: List of markers on X and Y chromosomes in humans considered by Lemaitre et al. [41]. Columns from left to right represent: 1- The marker name (in parenthesis are indicated marker numbers in Ross et al. [55]); 2- start position on X chromosome; 3- end position on X chromosome; 4- start position on Y chromosome; 5- end position on Y chromosome; 6- order of markers on X chromosome; 7- order of markers on Y chromosome; 8- Stratum on X to which the marker belongs (some markers have two different values, due to the two hypotheses analyzed by Lemaitre et al. [41]); 9- The reference mentioning the marker for the first time.

Analyzing a genome

We developed a complete analysis of a genome with respect to all possible stratifications, that can be described as follows. First, we enumerate all optimal sorting sequences of reversals using Algorithm 2. For each resultant sorting sequence s, we construct the normal form of the trace T that contains s (iterating Algorithm 3 over s), and we verify whether s is a stratifying sequence, that is, whether there is a $k-$strata $B = (I_k, I_{k-1}, \ldots, I_1)$ on the initial genome, such that s has the properties given in Definition 1 with respect to B. If this is the case and the $k-$strata B is already associated to the trace T, we increment the number of occurrences of the pair $< T, B >$ (that is, the number of sequences in T that produce the stratification B in the original genome). If the $k-$strata B is not yet associated to T, then we start the pair $< T, B >$ with one occurrence. If s does not produce any stratification, the procedure is analogous. If the absence of strata is already associated to the trace T, we increment the number of occurrences of the pair $< T, null >$ (that is, the number of sequences in T that do not produce a stratification in the original genome). If the absence of strata is not yet associated to T, then we start the pair $< T, null >$ with one occurrence.

The results of applying this analysis to the permutations representing human X and Y chromosomes proposed by Ross *et al.* [55] are given in Table 22.

The random genome generators

We also developed random genome reversal-generators, that create new genomes by applying a defined number of random reversals on an initial genome with a certain number of markers. The construction is parsimonious: if we apply d reversals to create a genome A_d from an initial genome A_0, it means that the minimum number of reversals to sort A_d back into A_0 is d. This construction can be represented by the following schema:

$$A_0 \rightarrow \rho_1 \rightarrow A_1 \rightarrow \rho_2 \rightarrow A_2 \rightarrow \ldots \rightarrow A_{d-1} \rightarrow \rho_d \rightarrow A_d$$

To select a reversal ρ_i during this construction, we take two random positions in the genome A_{i-1}, and test whether inverting the interval comprised between these two positions in A_{i-1} to obtain A_i is parsimonious (that is, whether the most parsimonious sequence of reversals to sort

Trace normal form / Number of strata	3 B_{xy}	3 A_3	2 A_2	1 A_1	0 *null*	Total
$\{3\}\{3,\ldots,12\}\{5,6\}\{8\}\{11\}\{11,12\} < \{1,2,11,12\}\{5,7\}$	420	0	$2,520$	0	$7,140$	$10,080$
$\{1,2,3\}\{1,\ldots,12\}\{4,\ldots,10\}\{5,6\}\{8\}\{11\} < \{1,2,4,\ldots,10\}\{5,7\}$	0	0	0	$1,260$	$8,820$	$10,080$
$\{1,\ldots,11\}\{1,\ldots,12\}\{3,\ldots,11\}\{3\}\{5,6\}\{8\} < \{1,2,11\}\{5,7\}$	0	0	0	$1,260$	$8,820$	$10,080$
$\{5,6\}\{8\}\{11,12\} < \{4,\ldots,10,12\}\{5,7\} < \{1,2,3,12\} < \{1,\ldots,11\}\{1,2,4,\ldots,10\}$	0	120	0	0	216	336
$\{1,\ldots,11\}\{5,6\}\{8\} < \{1,2,3,12\}\{5,7\} < \{4,\ldots,10,12\} < \{1,2,4,\ldots,10\}\{11,12\}$	0	0	0	0	336	336
$\{3,\ldots,11\}\{4,\ldots,10\}\{5,6\}\{8\} < \{3,12\}\{5,7\} < \{4,\ldots,10,12\} < \{1,2,11,12\}$	0	0	0	0	840	840
Total	420	120	$2,520$	$2,520$	$26,172$	$31,752$

Table 22: Analysis of all optimal sequences of reversals that sort the human chromosome $X = \mathcal{I}_{12}$ into $Y = (-12,11,-2,-1,-10,-9,8,-5,7,6,-4,3)$ with respect to all possible stratifications. Columns from left to right are: 1- A sequence of reversals that represent the trace; 2- Sequences that produce the $3-$strata $B_{xy} = (\{1,2\},\{3,\ldots,10\},\{11,12\})$ (strata boundaries proposed by Ross et al [55]); 3- Sequences that produce the alternative $3-$strata $A_3 = (\{1,2,3\},\{4,\ldots,10\},\{11,12\})$; 4- Sequences that produce the alternative $2-$strata $A_2 = (\{1,2\},\{3,\ldots,12\})$; 5- Sequences that produce the alternative $1-$strata $A_1 = (\{1,\ldots,12\}$; 6- Sequences that do not stratify the X chromosome (null); 7- The total number of sequences in the trace.

the genome A_i back into A_0 has i reversals). If a reversal is not valid under this criterion, we repeat the process of taking two random positions in the base genome A_{i-1} until finding a valid reversal for the step i.

We built two different generators: a *free-generator*, that does not impose additional constraints to the selection of reversals, accepting any optimal sequence of reversals; and also a $B-$*constrained-generator*, that imposes a second constraint in the selection of reversals, accepting only an optimal sequence of reversals that produces the given $k-$strata $B = (I_k, I_{k-1}, \ldots, I_1)$ on the initial genome. Thus the $B-$constrained-generator generates a genome through a $B-$stratifying sequence of reversals, according to the previous Definition 1.

Although the two positions selected before each reversal are uniformily distributed, we can not guarantee that the resulting simulated permutations are also uniformily distributed.

Strata likelihood analysis

In order to determine whether the stratification process, as proposed by Ross *et al.* [55], with $X = (1,2,3,4,5,6,7,8,9,10,11,12)$, $Y = (-12,11,-2,-1,-10,-9,8,-5,7,6,-4,3)$ and the $3-$ strata $B_{xy} = (\{1,2\},\{3,\ldots,10\}\{11,12\})$ on X, is likely to be true, the free and the

$B_{xy}-$constrained generators were used in a probabilistic study of Lemaitre *et al.* [41]. The simulated genomes were generated over the initial genome $X = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)$ (12 markers), by the application of 8 reversals (the reversal distance between the chromosomes X and Y of Ross *et al.* [55] is 8).

The distributions of optimal sequences of reversals with respect to the stratification for free versus $B_{xy}-$constrained simulations were compared by using non-parametric statistics. Although the two sets of simulations generated are extreme cases and intermediary cases were not investigated, and, moreover, there is no guarantee that the simulated permutations are uniformily distributed, the results suggest that the hypothesis that evolutionary strata 3, 4 and 5 have been formed by Y inversions is a likely hypothesis [41].

5.5 Symmetry *versus* asymmetry when applying constraints

In the previous sections, all the biological constraints were analyzed qualitatively, in order to determine whether they were symmetric (when the results of the analysis of the sequences sorting a first genome into the second can be obtained from the analysis of the sequences sorting the second genome into the first) or asymmetric.

We argued that when a constraint is asymmetric, it can only be applied when the relation ancestor-descendant between the analyzed genomes is known and, in this case, a direction to the analysis must be defined. We summarize below the characteristics of each one of the constraints that we considered.

1. *Initial detection of common intervals without interval breaks*: can be applied to linear or circular chromosomes and is symmetric.

2. *Initial detection of common intervals with a bounded number of interval breaks*: can be applied to linear or circular chromosomes and is asymmetric (the relation ancestor-descendant between the analyzed genomes must be known, and the analysis must be done from the descendant to the ancestor).

3. *Progressive detection of common intervals without or with a bounded number of interval breaks*: can be applied to linear or circular chromosomes and is asymmetric (the relation

ancestor-descendant between the analyzed genomes must be known, and the analysis must be done from the descendant to the ancestor).

4. *Terminus-symmetry*: can be applied only to circular chromosomes and is symmetric.

5. *Strata on sexual XY chromosomes*: can be applied only to linear chromosomes and is conceptually asymmetric (the analysis must be done from the X to the Y chromosome).

5.6 Compatibility between constraints

A list of different constraints can be applied together under the condition that they are compatible. Two symmetric constraints are frequently compatible and result in a symmetric approach, for example terminus-symmetry and initial detection of common intervals without interval breaks. An asymmetric and a symmetric constraints are also frequently compatible, and applied together they result in an asymmetric approach. This was the case of applying the progressive detection of common intervals, which is an asymmetric constraint, together with the terminus-symmetry, which is a symmetric constraint, in the previous analysis of *Rickettsia* bacterium.

On the other hand, the strata constraint can not be applied together with the terminus-symmetry, because while the first is defined only for linear chromosomes, the second is defined only for circular chromosomes. Moreover, the asymmetric strata and progressively detected common intervals are not compatible, because the analysis when applying the strata constraint may be done from the ancestor to the descendant, while the analysis when applying the progressive detection of common intervals may be done from the descendant to the ancestor.

Thus, using a list of different constraints at once may be interesting when analyzing traces, but we may first verify the compatibility in order to select the constraints to be applied. We summarize here the compatibility analysis of the constraints that we considered in this chapter.

Initial and progressive detection of common intervals are alternative approaches, that can not be applied together. When interval breaks are not accepted, initial detection of common intervals is symmetric and can be combined with terminus-symmetry or strata constraints. The initial detection of common intervals with a bounded number of interval breaks, as well as the progressive detection of common intervals (with or without interval breaks), can be combined with the terminus-symmetry constraint.

Strata and terminus-symmetry are incompatible because the first is proper to linear while the second is proper to circular chromosomes. Strata is conceptually applied in the analysis from X (closer to the ancestor) to Y chromosome and is also incompatible with the progressive detection of common intervals, that must be applied from the descendant to the ancestor.

5.7 Final remarks

In this chapter we proposed the use of different biological constraints to reduce the universe of sequences and classes, and show how to apply these methods to analyze real cases in evolution. One of these constraints is the list of common intervals, which are the clusters of co-localised genes between the considered genomes. We used common intervals in two different approaches, one that searchs for perfect sorting sequences [18], that is, sequences that do not break the common intervals detected between the two initial genomes, and one that searchs for progressive perfect sorting sequences, that do not break the common intervals progressively detected [17] when sorting a genome into another. Other constraints were defined according to the practical problems we were interested in. In particular, we are able to characterize the reversals with respect to the replication terminus in circular chromosomes (this method was used to analyze the evolution of the *Rickettsia* bacterium), and to apply a constraint to analyze directly the stratification process in the evolution of the sexual chromosomes X and Y in human [18, 41].

The implementation of all these variants are part of baobabLuna framework, that will be described in Chapter 6. Several experiments show that indeed these variants reduce considerably the number of generated traces. Consequently, they run faster than the algorithm that generates the traces without contraints and are able to analyze permutations with higher reversal distances. The constraints were also analyzed in order to determine whether they lead to symmetric or to asymmetric approaches, and to verify which constraints are compatible and which constraints are incompatible.

We then applied our method that generates traces according to biological constraints to analyze two real cases, the evolution of human sexual chromosomes X and Y [55] and the evolution of the *Rickettsia* bacterium, which is an intracellular parasite. In both cases, we obtained a better characterization of the evolutionary scenarios of these genomes [17, 18, 41], with respect to the results of previous studies [13, 55], that were based on a single sorting sequence.

The use of biological constraints has some important limitations. First, there is no guarantee that a sequence that respects the given contraints exists, thus this approach may lead to empty results, which is undesirable. Relaxing the biological constraints in order to obtain a non-empty result is generally possible, but this approach may require several essays of the relaxing parameters, which costs computation time.

Moreover, in the use of the terminus-symmetry constraint, the choice of the values for the parameters p (the maximum number of external reversals), q (the maximum number of terminus-asymmetric reversals) and r (the threshold for the terminus-symmetry rate) is not trivial and often requires several essays. In addition, we adopt some simplifications such as ignoring the space between markers and the replication terminus shift produced by each terminus-asymmetric reversal. We must have these simplifications in mind when interpreting the results of the analyses.

Chapter 6

BAOBABLUNA

Summary

6.1 Optimization of memory use **94**

6.1.1 The compressible sorted set 94

6.1.2 Freezing operations 96

6.1.3 Performance 96

6.2 Architecture **97**

6.3 Test **100**

6.4 Download and setup **101**

6.5 Final remarks **102**

The implementation of all the algorithms presented in this work is integrated to the software BAOBABLUNA [16], a framework to deal with permutations. This framework was implemented in a object-oriented paradigm, using the Java technology[13].

Besides the implementation of Siepel's algorithm and of all algorithms for analyzing traces, BAOBABLUNA contains a collection of classes for dealing with permutations and their associate breakpoint graphs, performing reversals, calculating reversal distances and sorting permutations by reversals. The interface of BAOBABLUNA, including a detailed list of the main executable programs, is described in appendix A.

[13]The full documentation of Java technology is available online at `http://java.sun.com`.

6.1 Optimization of memory use

When analyzing traces, for a reversal distance of about 20, we may need to handle more than 200 million partial traces at one step. Since the java standard data structures were inefficient and rapidly saturated when dealing with this amount of data, we needed to develop a new structure, that we called `CompressibleSortedSet`.

6.1.1 The compressible sorted set

The `CompressibleSortedSet` can store more elements than the standard data structures due to its internal organization, that allows automatic partial compression of its data. In general, insertions in a `CompressibleSortedSet` are not processed one by one, but accumulated and then processed in batch mode, from time to time.

Internally, the elements are stored in subsets, such that for any pair of subsets S_1 and S_2, either all elements in S_1 are smaller than all elements in S_2, or all elements in S_1 are greater than all elements in S_2. The elements in a subset are sorted in ascending order, and the scope of a subset is given by its first and last elements. The subsets themselves are also sorted in ascending order. Searching an element (or the position to insert an element) in a `CompressibleSortedSet` is done by a double application of the binary search method [39], that is, a first binary search is performed to determine in which subset the element may be inserted, then a second binary search is performed in the selected subset, to determine the exact insert position. In addition, each one of the subsets can be in compressed or uncompressed state. The scope of a subset is visible even when it is compressed, so it is possible to know whether a new element should be inserted in a subset or not.

Inserting an element in an uncompressed subset is immediate. However, if the subset is compressed, the element is kept in a pending list for a while. After a certain number of insertions in an compressed subset, the subset is uncompressed and all the elements in the pending list are inserted at once (this procedure amortizes the cost of uncompressing a subset). The number of uncompressed subsets is limited, thus generally a subset decompression may cause another subset compression. At any time, the uncompressed subsets are those which were accessed later. Figure 32 illustrates the structure of `CompressibleSortedSet`.

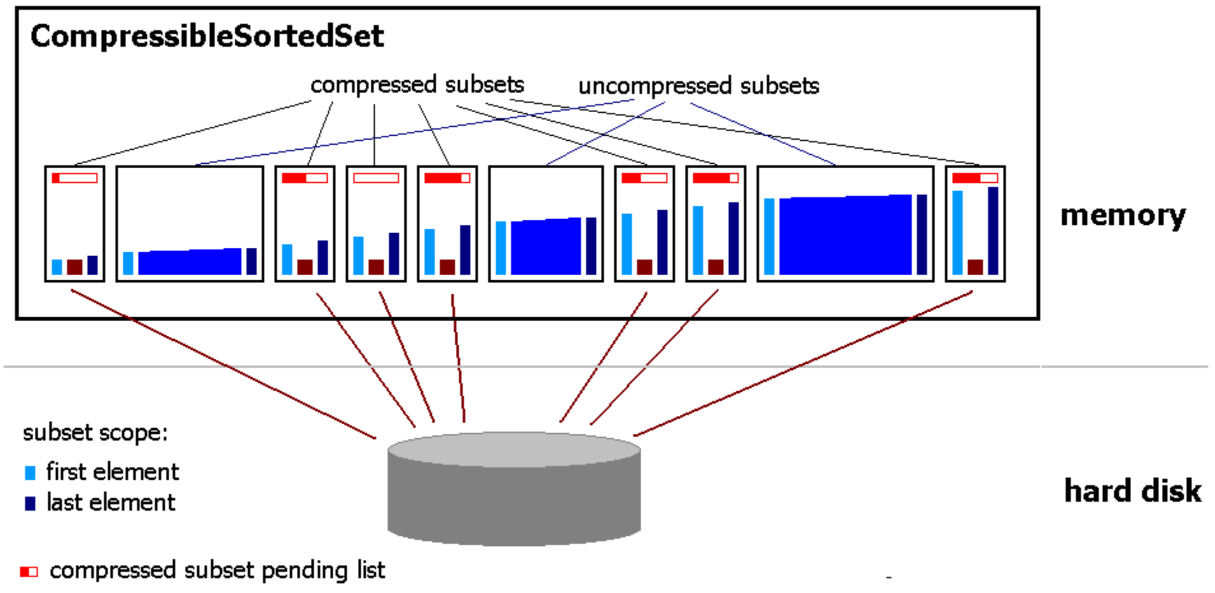


Figure 32: The structure of `CompressibleSortedSet`.

When an internal subset of `CompressibleSortedSet` achieves a certain number of elements, it is splitted in two. The split operation is done according to a given balance x, such that $0 < x < 1$. If the original subset has n elements, it is splitted in two subsets such that the first nx elements of the original subset are put in one new subset and the last $n(1-x)$ elements are put in the other new subset. The default value of the balance is 0.5. However, in some applications, it can be interesting to define a different value, thus the balance is a parameter of `CompressibleSortedSet`. The maximum number of uncompressed subsets, the maximum number of elements in a subset and the maximum number of elements in a subset pending list are also parameters that can be defined. The operations of compression, decompression and split of a subset are represented in Figure 33.

The compression of a subset is done as follows: first we may concatenate all elements of the subset in a bidimensional array of bytes, then we compress each line of the resultant array. The compressed data is written in the hard disk, so the memory that the elements occupied can be released. The algorithm of compression is for general purpose, implemented in package java.util.zip (classes Deflater and Inflater) using the ZLIB compression library. The ZLIB compression library is not protected by patents and is fully described in the specifications at the

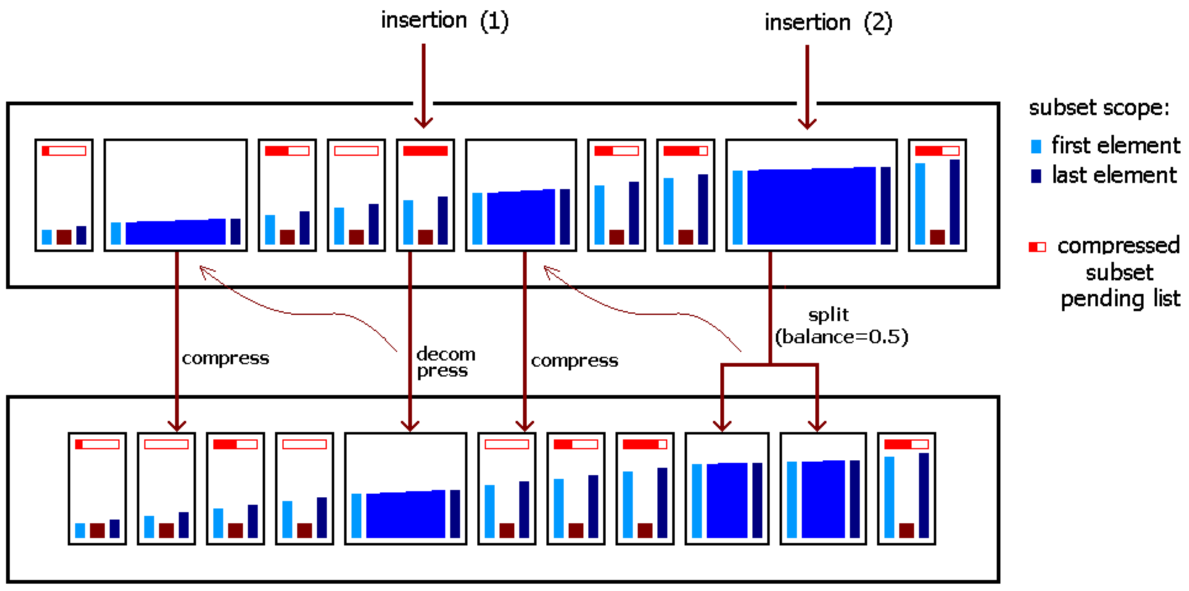


Figure 33: An insertion in a `CompressibleSortedSet` *may launch a subset decompression (when the subset is compressed and its pending list is completely filled, such as insertion 1) or a subset split (when the subset is decompressed and the maximum number of elements in the subset is achieved, such as insertion 2). A subset decompression or split may eventually launch another subset compression, if the maximum number of decompressed subsets is achieved.*

java.util.zip package description[14].

6.1.2 Freezing operations

Besides the class `CompressibleSortedSet`, the implementation of the algorithm that enumerates traces also perform additional I/O operations between iterations to save memory. At the end of an interation i, the $i-$traces are registered in the hard disk, by a procedure that we call *freezing*. Then, in the following iteration $i+1$, the $i-$traces are read little by little from the hard disk to generate the $(i+1)-$traces.

6.1.3 Performance

The standard implementation of the algorithm that enumerates traces in BAOBABLUNA, that we call **Compression+Freezing**, uses the class `CompressibleSortedSet` and freezing operations between iterations to save memory.

In order to evaluate the performance of this implementation with respect to memory use

[14]See the java API of the package java.util.zip in `http://java.sun.com`.

and execution time, we have done a comparison with three other implementations of the same algorithm that enumerates traces (two of them uses the class `java.util.TreeSet`, which is a java standard implementation of a sorted set, instead of `CompressibleSortedSet` to store traces). The three test versions are:

- **Compression**: uses the class `CompressibleSortedSet` but does not use freezing operations between iterations.
- **TreeSet**: uses the class `TreeSet` and does not use freezing operations between iterations.
- **TreeSet+Freezing**: uses the class `TreeSet` and freezing operations between iterations.

We run the standard version and the three test versions for computing traces of three different permutations. The implementations that use the class `CompressibleSortedSet` (the standard Compression+Freezing and the test version Compression) were set with the following parameters:

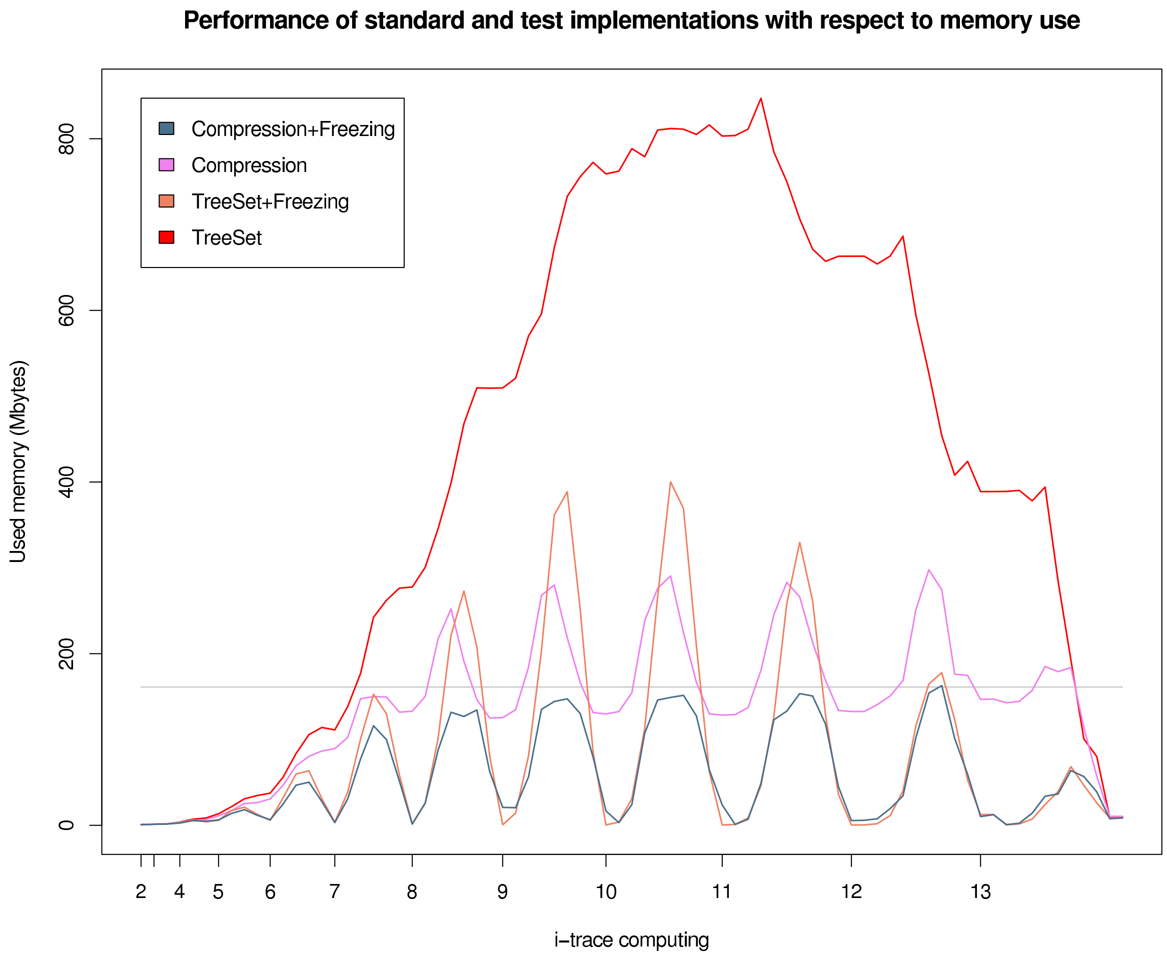
1. Maximum number of elements by compressible subset: $3,000$
2. Maximum number of uncompressed subsets: 400
3. Maximum number of elements in the pending insertion list of each compressed subset: 200

All experiments were made on a 64 bit personal computer with two 3GHz CPUs and 2GB of RAM. Experimental results show that the standard implementation (Compression+Freezing) indeed saves a considerable amount of memory (see Figure 34).

In addition, as we can see in Table 23, the results show that, although `CompressibleSortedSet` may proceed several compressions, decompressions and I/O operations to register and read compressed subsets in the hard disk, it runs at least as fast as `TreeSet` (freezing operations takes longer with `TreeSet`, because this implementation registers and reads each trace individually, while `CompressibleSortedSet` can register and read the elements in a compressed subset at once). This indicates that the amortization provided by insertion pending lists works well.

6.2 Architecture

All executable programs in BAOBABLUNA have the same multilayered architecture [21], organized in two layers. The topmost level is responsible for the user interface. The programs in



Total number of i–traces

2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
690	6331	38986	172257	567851	1413316	2669032	3824570	4047048	3298406	1958839	567524

Figure 34: The memory use of standard implementation and test versions when computing the $13-$*traces that sort* $(-12, 11, -10, 6, -5, 13, 2, 7, 8, -9, 14, -15, 3, 4, -16, 1)$*, which is permutation* π_H *in table 23. The labels in the* X *axis indicates the beginning of* $i-$*traces computing in the execution of each implementation (for* $i = 2, 3, \ldots, 13$*; the total number of* $i-$*traces is given below the graphic). The valleys in some lines are between each pair of iterations* $i-1$ *and* i*, and are more accentuated for the implementations that perform freezing operations. This graphic shows that Compression versions are more stable with respect to memory use and, for greater amounts of data, use much less memory than* `TreeSet` *versions. The* BAOBABLUNA *standard implementation (Compression+Freezing) has the best performance among all implementations.*

BAOBABLUNA have a light textual interface, described in Appendix A. The second layer is responsible for implementing the logic behind the applications. Thus, all the algorithms described in this manuscript are implemented in the second layer. The dependency relation of these two layers are from top to bottom, that is, the second layer has no dependency with respect to the top layer. In addition, theses two layers depend on the core library of BAOBABLUNA, mostly

Input (π)	$d(\pi)$	N_T	Implementation	Execution time
π_F $n = 13$	10	$2,151$	`Compression+Freezing`	$\simeq$ `27 seconds`
			`Compression`	$\simeq$ `25 seconds`
			`TreeSet+Freezing`	$\simeq$ `31 seconds`
			`TreeSet`	$\simeq$ `25 seconds`
π_G $n = 16$	12	$21,902$	`Compression+Freezing`	$\simeq$ `7.3 minutes`
			`Compression`	$\simeq$ `7.1 minutes`
			`TreeSet+Freezing`	$\simeq$ `8.7 minutes`
			`TreeSet`	$\simeq$ `7.2 minutes`
π_H $n = 16$	13	$567,524$	`Compression+Freezing`	$\simeq$ `4.1 hours`
			`Compression`	$\simeq$ `4.1 hours`
			`TreeSet+Freezing`	$\simeq$ `4.8 hours`
			`TreeSet`	$\simeq$ `4.2 hours`

Table 23: Computation results. Columns from left to right contain: 1- the input (permutation and number of markers); 2- the reversal distance; 3- the number of traces; 4- the implementation; 5- the execution time of the implementation. All implementations of the algorithm are part of the BAOBABLUNA *package. The three analyzed permutations are* $\pi_F = (-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$, $\pi_G = (-12, 11, -10, -1, 16, -4, -3, 15, -14, 9, -8, -7, -2, -13, 5, -6)$, *and* $\pi_H = (-12, 11, -10, 6, -5, 13, 2, 7, 8, -9, 14, -15, 3, 4, -16, 1)$.

composed by a component to deal with permutations, their corresponding breakpoint graphs and reversals, a component to deal with traces, and a utilitary component that contains the class `CompressibleSortedSet`. There is no dependency between these three components of the core library. The pattern of architecture of BAOBABLUNA is illustrated in Figure 35.

The organization of the architecture in components with well defined dependency relations favors the development and maintainance of BAOBABLUNA. If a component `X` depends on another component `Y`, then the modifications done in `Y` can affect `X`. On the other hand, if `X` does not depend on `Y`, then the modifications done in `Y` does not affect `X`. Observe that each component in the core library does not depend on any other component of BAOBABLUNA and, consequently, is not affected by the modifications done outside itself. Moreover, the separation of the user interface from the application logic permits the substitution of the interface with no consequences to the other components. Neither the application logic nor the core library depend on the user interface, thus modifying the interface does not affect the remaining components of BAOBABLUNA.

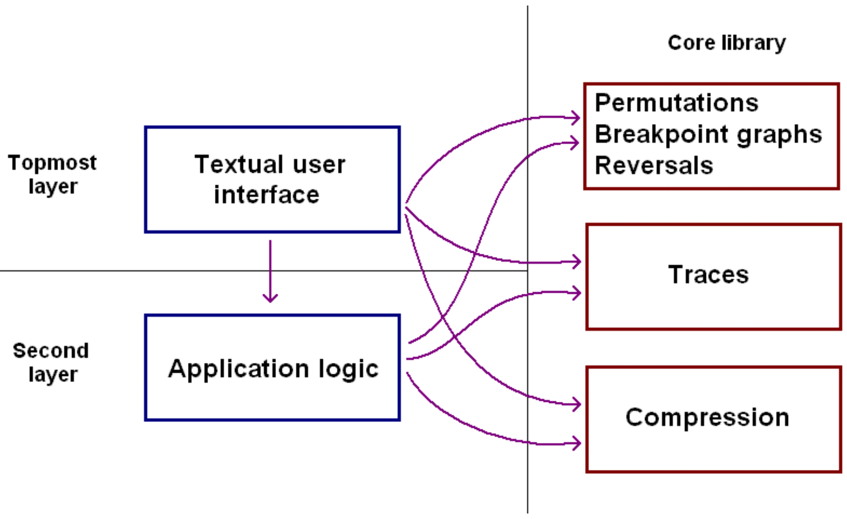


Figure 35: The pattern of the architecture of executable programs in BAOBABLUNA. *The components are represented by rectangles. The lines separate the two layers of the architecture and the core library, which is transversal to the layers. An arrow from a component A to a component B indicates that A depends on B.*

6.3 TEST

An important issue when implementing a program that deals with a huge amount of information is preventing erroneous results, that are very hard to detect. A good strategy to solve this problem is the use of an auditor, that is a program that contains an alternative implementation that may produce the same result as the implementation which is being tested. The auditor can also contain errors, but it is very improbable that it reproduces exactly the same errors as the tested implementation, so when both lead the same results, we assume that both are correct, at least for the examined test cases.

In BAOBABLUNA we implemented auditors for the two main critical points. One is the algorithm that generates directly the traces of sorting sequences (Algorithm 5), that here we call `traceStandard`. In this case, the auditor is the program that generates traces by generating all sorting sequences (Algorithm 4), named `traceAuditor`, that was also used to evaluate the performance of `traceStandard` in Chapter 4. Observe that `traceAuditor` can not process all the permutations that are processed by the `traceStandard`, that is, with respect to the auditor, `traceStandard` is able to process permutations with higher reversal distances. Nevertheless, `traceAuditor` can be used as a good indicator of the quality of `traceStandard`. The second critical point is the class `CompressibleSortedSet`, whose auditor is the class `TreeSet`, which is a java standard implementation of a sorted set. The class `TreeSet` was also used to evaluate the performance of `CompressibleSortedSet` with respect to memory use, as we saw in Section 6.1.1.

All versions of BAOBABLUNA were tested by these two auditors for a set of 8 test permutations (see Table 24), that helped us to find several errors during the implementation process. The errors were fixed, and the versions were released only after being aproved by these test cases.

Permutation (π)	$d(\pi)$	$\mathbf{N}_T$	$\mathbf{N}_S$
$\pi_A = (-3, 2, 1, -4)$	4	2	28
$\pi_B = (-4, 1, -3, 6, -7, -5, 2)$	6	5	204
$\pi_C = (-6, 5, 7, -1, -4, 3, 2)$	6	8	496
$\pi_D = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$	8	6	31,752
$\pi_E = (-4, 3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$	9	14	407,232
$\pi_F = (-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$	10	2,151	8,278,540
$\pi_G = (-12, 11, -10, -1, 16, -4, -3, 15, -14, 9, -8, -7, -2, -13, 5, -6)$	12	21,902	505,634,256
$\pi_H = (-12, 11, -10, 6, -5, 13, 2, 7, 8, -9, 14, -15, 3, 4, -16, 1)$ *	13	567,524	40,313,272,766

Table 24: Test cases used to assure the quality of BAOBABLUNA. *In all released versions,* `traceStandard` *and its auditor* `traceAuditor`*, as well as* `CompressibleSortedSet` *and its auditor* `TreeSet` *lead to the same results when computing traces for these test cases. (*)Due to its huge number of sorting sequences, the last permutation can not be processed by* `traceAuditor` *and was only used to audit* `CompressibleSortedSet`.

6.4 Download and setup

The package BAOBABLUNA is available at `http://pbil.univ-lyon1.fr/software/luna/` for online download, encapsulated in a **.jar** file (Java ARchive format). Together with the Java 6 standard library[15], this file contains all required resources to run the programs described in this manuscript. The current version of BAOBABLUNA, named `1.1 beta`, was released on November 2008[16]. A tutorial for downloading and running the programs is available in Appendix A.

For java programmers, the API (application programming interface) of all classes in the package BAOBABLUNA can also be downloaded at `http://pbil.univ-lyon1.fr/software/luna/`.

The package containing the class `CompressibleSortedSet` is part of BAOBABLUNA. However, although `CompressibleSortedSet` was conceived for dealing with traces, it is for general purposes

[15]Java 6 must to be installed in the computer, in order to run the programs in BAOBABLUNA.

[16]The first version of BAOBABLUNA, named `0.1 beta`, was released on November 2006 and did not contain the variants that deal with biological constraints.

and is able to store any kind of object. For java programmers interested in this functionality, a subpackage containing only these features is available at `http://biomserv.univ-lyon1.fr/~marilia/compressible.htm`.

6.5 FINAL REMARKS

In this chapter we presented BAOBABLUNA [16], a framework to deal with permutations, their corresponding breakpoint graphs, and reversals, that contains all algorithms described in this manuscript. In order to be able to handle the huge amount of data when computing traces, the implementation of BAOBABLUNA optimizes the memory use with compression and freezing operations, that were shown to save a considerable amount of memory without losing in execution time. The core library of BAOBABLUNA is mostly composed by three independent components, one that deals with permutations, breakpoint graphs and reversals, one that deals with traces, and one that deals with compression. In addition, the architecture of all the executable programs BAOBABLUNA is organized in two layers, a topmost layer with the user interface and a second layer with the application logic, that encloses our algorithms.

Chapter 7

Conclusions, limitations and perspectives

Summary

7.1 Main results 103
7.1.1 An algorithm to enumerate all the traces 103
7.1.2 Biological constraints 104
7.1.3 Applications 106
7.1.3.1 Analysis of the human sexual chromosomes X and Y 106
7.1.3.2 Analysis of the *Rickettsia* bacterium 107
7.1.4 baobabLuna 107
7.2 Limitations 108
7.3 Future perspectives 108

7.1 Main results

7.1.1 An algorithm to enumerate all the traces

We studied the algorithmic aspect of a common problem in genome rearrangements, called sorting a genome by reversals, which consists in finding an optimal sequence of reversals that transforms one genome into another. When only genomes without gene duplications are considered, there are efficient algorithms to find a unique solution [32].

We worked mostly on the problem of exploring the space of all optimal sequences of reversals that sort one genome into another. The universe of all sorting sequences can be enumerated thanks to an algorithm by Siepel [59]. However, the number of different sorting sequences is usually huge, and we developed an algorithm that builds a compact representation of the space of sequences, using a model previously proposed by Bergeron *et al.* to group the sorting sequences into equivalence classes called traces [9]. This algorithm, that is one of the most important results of our work, gives one representative and the number of sequences in each trace without enumerating all sorting sequences [18].

We also showed that, for any permutation π that has at most one unoriented component, we can compute the set of traces that sort each component of π independently, and then obtain the traces sorting π by multiplying the traces sorting its components. This approach runs faster than computing directly the traces.

7.1.2 Biological constraints

We then proposed the use of different biological constraints to reduce the universe of sequences and classes, and showed how to apply these methods to analyze real cases in evolution. One of these constraints is the list of common intervals, which are the clusters of co-localised genes between the considered genomes - an optimal sequence of reversals that does not break the common intervals may be more realistic than one that does break them [26]. We used common intervals in two different approaches, one that searches for perfect sorting sequences [18], that is, sequences that do not break the common intervals detected between the two initial genomes, and one that searches for progressive perfect sorting sequences, that do not break the common intervals progressively detected [17] when sorting a genome into another. Other constraints were defined according to the practical problems we were interested in. In particular, we are able to characterize the reversals with respect to the replication terminus in circular chromosomes (this method was used to analyze the evolution of the *Rickettsia* bacterium), and to apply a constraint to analyze directly the stratification process in the evolution of the sexual chromosomes X and Y in human [18, 41].

The complexities of the algorithms taking into account the initial or the progressive detection of common intervals and the strata on the XY chromosomes were examined and were proven to be the same as for the original algorithm, that enumerates all the traces. Moreover, several experiments show that all the variants that take biological constraints in consideration run faster than the algorithm that generates all the traces.

All the constraints were analyzed qualitatively, in order to determine whether they were symmetric (when the results of the analysis of the sequences sorting a first genome into the second can be obtained from the analysis of the sequences sorting the second genome into the first) or asymmetric. We argued that when a constraint is asymmetric, it can only be applied when the relation ancestor-descendant between the analyzed genomes is known and, in this case, a direction to the analysis must be defined. We summarize below the characteristics of each one of the constraints that we considered.

1. *Initial detection of common intervals without interval breaks*: can be applied to linear or circular chromosomes and is symmetric.
2. *Initial detection of common intervals with a bounded number of interval breaks*: can be applied to linear or circular chromosomes and is asymmetric (the relation ancestor-descendant between the analyzed genomes must be known, and the analysis must be done from the descendant to the ancestor).
3. *Progressive detection of common intervals without or with a bounded number of interval breaks*: can be applied to linear or circular chromosomes and is asymmetric (the relation ancestor-descendant between the analyzed genomes must be known, and the analysis must be done from the descendant to the ancestor).
4. *Terminus-symmetry*: can be applied only to circular chromosomes and is symmetric.
5. *Strata on sexual XY chromosomes*: can be applied only to linear chromosomes and is conceptually asymmetric (the analysis must be done from the X to the Y chromosome).

Moreover, we showed that some constraints can be applied together, under the condition that they are compatible. If one or more symmetric constraints are applied together with at least one asymmetric constraint, the resulting approach is asymmetric.

Initial and progressive detection of common intervals are alternative approaches, that can not be applied together. When interval breaks are not accepted, initial detection of common intervals is symmetric and can be combined with terminus-symmetry or strata constraints. The initial detection of common intervals with a bounded number of interval breaks, as well as the progressive detection of common intervals (with or without interval breaks), can be combined with the terminus-symmetry constraint.

Strata and terminus-symmetry are incompatible because the first is proper to linear while the second is proper to circular chromosomes. Strata is conceptually applied in the analysis from the X (closer to the ancestor) to the Y chromosome and is also incompatible with the progressive detection of common intervals, that must be applied from the descendant to the ancestor.

7.1.3 Applications

We applied our method to analyze two real cases, the evolution of the human sexual chromosomes X and Y and the evolution of the *Rickettsia* bacterium, which is an intracellular parasite. We obtained a better characterization of the evolutionary scenarios of these genomes, with respect to the results of previous studies [13, 55], that were based on a single sorting sequence.

7.1.3.1 Analysis of the human sexual chromosomes X and Y

In the analysis of the human sexual chromosomes X and Y, Ross *et al.* [55] proposed a particular stratification on the X and used a unique sorting sequence as an argument to support the observed strata. However the existence of other traces that could produce the same strata and the existence of other strata boundaries were not examined. Using our algorithm to generate traces and strata on the X chromosome as a parameter, we developed a method that uses the bounds of the given strata to search directly the subtraces of sequences that sort X into Y and produce these strata on the X. With this method, we could test different hypotheses of stratification. In particular, we could verify that all sequences sorting X into Y according to the strata proposed by Ross *et al.* [55] are in the same subtrace. So if we suppose that these bounds are accurate, the reversals in the sequence given in [55] were identified correctly.

However, the permutations representing the X and Y chromosomes as proposed by Ross *et al.* [55] cover only the first 11.2Mbps on the X, and even for this small portion of the chromosome there are alternative strata boundaries. Moreover, if we extend the permutations to cover a bigger portion of the chromosome, the number of possibilities for placing the strata boundaries increases. Indeed, only the boundary between the pseudo-autosomal region (PAR) and stratum 5 and the boundary between the strata 5 and 4 are well established. The other boundaries are still controversial, and even the number of ancient strata is discussed. We participated in a collaborative work of Lemaitre *et al.* [41], that presented more arguments to explain the stratification process. In particular, our method of searching for strata-induced subtraces was used to analyze extended permutations representing the human X and Y chromosomes, in order to verify some hypotheses for placing the boundary between strata 4 and 3, and as part of a probabilistic analysis, that indicated that the occurrence of a sequence of reversals that stratified the human X chromosome is more likely to be true than the occurrence of a sequence of reversals that did not.

7.1.3.2 Analysis of the *Rickettsia* bacterium

The evolutionary scenario of six species of the *Rickettsia* bacterium was reconstructed by Blanc *et al.* [13] and in particular one arbitrary optimal sequence of reversals was proposed to sort the ancestor $R2$ into *Rickettsia felis*. This sequence is composed by five external and four terminus-symmetric reversals. We could analyze the universe of all sorting sequences between these two genomes, and, searching for solutions with at most 3 external and no terminus-asymmetric reversals, with the threshold of 0.7 to the terminus-symmetry rate, combined with the progressive detection of common intervals accepting two interval breaks, we found three subtraces whose sorting sequences have three external and six terminus-symmetric reversals [17], obtaining a better characterization of the evolutionary scenario of these genomes than Blanc *et al.* [13].

7.1.4 baobabLuna

All the algorithms developed in this work are implemented, integrated to baobabLuna [16], a java framework to deal with genomes and reversals. Download and tutorial for baobabLuna are available on-line. In order to be able to deal with the huge amount of data when constructing traces, we developed a structure that is able to efficiently compress and store the traces in a sorted set during the construction. We compared the performance of this structure with a java standard implementation of a sorted set, showing that we are able to save memory without losing in the execution time. With baobabLuna, we run experiments of all the variants of our algorithm, showing the gain in the execution time when the biological constraints are applied.

7.2 LIMITATIONS

The algorithm to generate directly the traces represents an important improvement with respect to the enumeration of all optimal sorting sequences. However, for permutations with a reversal distance above a certain value, the universe of traces is too big for being interpreted and often it can not be computed, even with the optimization in the memory use in the implementation of BAOBABLUNA. Indeed, we are not able to compute traces for permutations with a reversal distance of about 20 or higher.

The use of biological constraints is a good strategy to reduce traces while selecting sequences of reversals that are biologically more meaningful. However, there is no guarantee that a sequence that respects the given contraints exists, thus this approach may lead to empty results, which is undesirable. Relaxing the biological constraints in order to obtain a non-empty result is generally possible, but this approach may require several essays of the relaxing parameters, which costs computation time.

Moreover, the terminus-symmetry, that is one of the most promising constraints, has some other limitations. The choice of the values for the parameters p (the maximum number of external reversals), q (the maximum number of terminus-asymmetric reversals) and r (the threshold for the terminus-symmetry rate) is not trivial and often requires several essays. In addition, we adopt some simplifications such as ignoring the space between markers and the replication terminus shift produced by each terminus-asymmetric reversal. We must have these simplifications in mind when interpreting the results of the analyses.

7.3 FUTURE PERSPECTIVES

Concerning the program to analyze traces, other specific constraints can be introduced. The challenge here is to know more details about the evolutionary process of relatively close related organisms, and isolate properties that could be used to select reversals at each step of the trace construction.

The program with the constraints that are currently implemented, specially the versions that account for the progressively detected common intervals, and for different types of reversals with respect to the replication terminus in circular chromosomes, can also be applied in the analysis of other real cases. We believe that it can reveal interesting details in the rearrangement process of circular chromosomes, but, since the implementation is not able to deal with genomes whose reversal distance is greater than 20, the challenge is to find genomes with the required characteristics. A good candidate is the bacterium *Mycbacterium*, whose ancestor was reconstructed recently [30], which allows us to use, as we did in the analysis of *Rickettsia*, an asymmetric approach combining the progressively detected common intervals and the terminus-symmetry.

Moreover, the variant that takes the replication terminus of circular chromosomes into account have some limitations and could be improved. An idea is to continue to limit the number of external reversals, but instead of limiting the number of terminus-assymetric reversals, we would limit the displacement of the replication terminus to the left or to the right of the chromosome center by a given Δ. This idea is illustrated in Figure 36. The advantage of this approach is that, after applying a reversal that moves the replication terminus, it would favor the choice of a reversal that compensates the previous displacement, that is, the chronology of the reversals would be more strongly constrained.

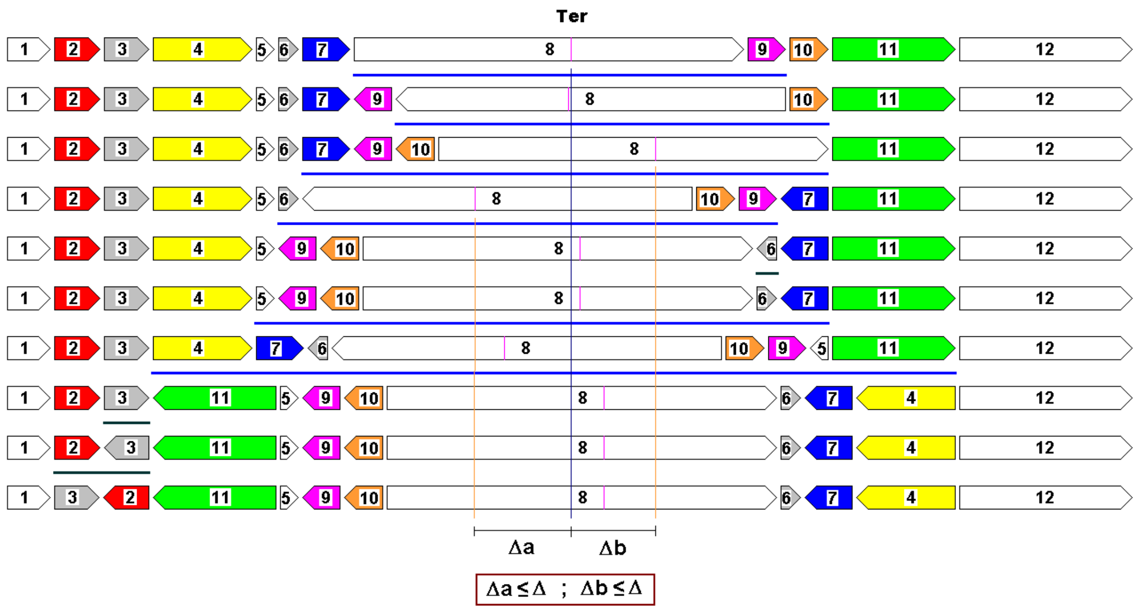


Figure 36: Limiting the displacement of the replication terminus in a circular chromosome by a given Δ.

The variant that searches for strata-induced subtraces can be applied to the study of sexual chromosomes of other species. Very poor data is available currently, but we expect that we will dispose of more and more sequences of sexual chromosomes of different species in the near future.

Appendix A

BAOBABLUNA interface

Summary

A.1 The text representation for the breakpoint graph 111

A.2 Download and setup 112

A.3 Running executable programs 112

A.3.1 `baobab.exec.permutation.analyzeSignedPermutation` 113

A.3.2 `baobab.exec.permutation.sort` 114

A.3.3 `baobab.exec.permutation.performReversals` 115

A.3.4 `baobab.exec.permutation.decomposeSignedPermutation` 115

A.3.5 `baobab.exec.trace.analyzeTraces` 115

The software BAOBABLUNA, described in chapter 6, is a java framework to deal with permutations. It contains a collection of classes for building breakpoint graphs, performing reversals, calculating reversal distances, sorting permutations and analyzing traces (with and without biological contraints).

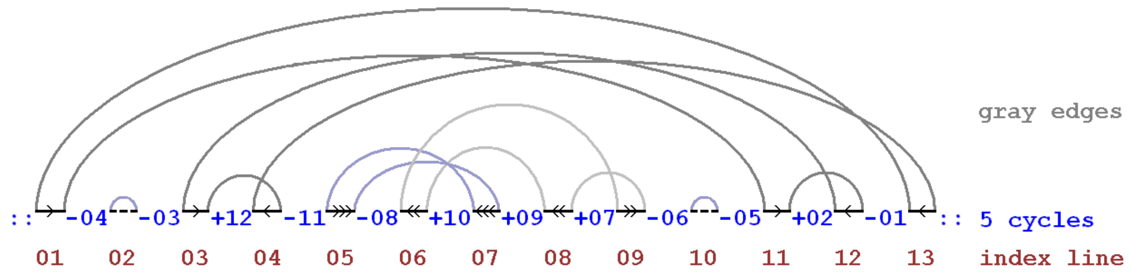
In this appendix we describe the BAOBABLUNA interface, including a list with the main executable programs.

A.1 THE TEXT REPRESENTATION FOR THE BREAKPOINT GRAPH

An important feature of BAOBABLUNA interface is a text representation of the breakpoint graph. In this text form, a breakpoint graph π is represented by a string with four lines. The first line assigns an index from 1 to $pts(\pi)$ to each black edge. The second line represent the black edges' cycles and directions '>' or '<' (two positions with the same value in the second line represent two black edges in the same long cycle; '`...`' represents an adjacency). The third line represent the gray edges of the cycles, since each position in the third line indicates the subsequent black edge in a tour through the cycle (again, '`...`' represents an adjacency). The fourth line contains the permutation and the number of cycles in the graph. Figure 37 shows the graphical and the text representation of the breakpoint graph of a permutation.

The text representation of breakpoint graphs is largely used in our executable programs. It is implemented by the class baobab.bio.permutation.PermutationBPGraphFormatter.

(A) Graphical representation



(B) Text representation

```
      01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.            (1)
      01> ... 01> <01 03> <02 <03 <02 02> ... 01> <01 <01            (2)
      :11 ... :04 :13 :07 :09 :05 :06 :08 ... :12 :03 :01            (3)
   :: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles   (4)
```

Figure 37: The graphical (A) and the text (B) representations of the breakpoint graph for the linear permutation $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$. *In the text form of this graph (B), the lines represent: (1) an index line identifying the black edges (points); (2) the black edges, with each cycle tour in an arbitrary direction (the numbers represent the long cycles, two black edges with the same number are in the same cycle); (3) the gray edges, with each cycle tour in the direction induced by black edges' direction; (4) the permutation. A '. . . ' in lines (2) and (3) represents an adjacency.*

A.2 DOWNLOAD AND SETUP

The package BAOBABLUNA is available at `http://pbil.univ-lyon1.fr/software/luna/` for online download, encapsulated in a **.jar** file (Java ARchive format). Together with the Java 6 standard library, this file contains all required resources to run all the algorithms described in this manuscript. For java programmers, the API (application programming interface) of all classes in the package BAOBABLUNA can also be downloaded at `http://pbil.univ-lyon1.fr/software/luna/`.

In order to run the executable programs in BAOBABLUNA, Java SE 6.0 must to be installed in the computer. Suppose the file that encapsulates the package BAOBABLUNA is called `baobab-luna.jar` and was downloaded into a directory called `$path`. First we must get into the directory $path (`>cd $path`). Then, to run a program called `$program`, the command is as follows:

```
>java -classpath baobab-luna.jar $program.
```

A.3 RUNNING EXECUTABLE PROGRAMS

The package BAOBABLUNA contains a list of executable programs to deal with signed permutations that represent genomes in bioinformatics. Those programs are useful for analyzing permutations over several aspects (calculating reversal distance, finding the components of the breakpoint graph, analyzing traces). Each executable has a particular list of parameters, and, to get the help, we must run the program without arguments (see an example in Figure 38).

All the programs described here analyze a pair of signed permutations, thus the following parameters are common to all of them:

```
>java -classpath baobab-luna.jar baobab.exec.permutation.sort

Sorts the given signed permutation

Required parameters:

   -o originPermutation (values separated by commas, without spaces)
        Ex: -o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1

Optional parameters:

   -t targetPermutation (values separated by commas, without spaces)
        Ex: -t 5,-3,12,10,9,8,-11,7,-6,-4,2,-1

   -l T/F (T=linear [default], F=circular)
        Ex: -l F
```

Figure 38: Get the help of a program in BAOBABLUNA.

- `-o [permutation]`: the origin permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n)$. The values must be separated by commas, without spaces.
  Example: `-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1`
- `-t [permutation]`: the target permutation π_T. The values must be separated by commas, without spaces. This parameter is optional and if it is ommited, π_T is assumed to be $\mathcal{I}_n = (1, 2, 3, \ldots, n)$.
  Example: `-t 5,-3,12,10,9,8,-11,7,-6,-4,2,-1`
- `-l T/F`: a parameter to indicate whether the origin and target permutations are linear (`T`) or circular (`F`). This parameter is optional and if it is ommited, the permutations are assumed to be linear.
  Example: `-l F`

Now we will give a list with the main executable programs that are part of our framework.

A.3.1 `baobab.exec.permutation.analyzeSignedPermutation`

Analyzes a signed permutation and prints its breakpoint graph and reversal distance (see Figures 39 and 40).

```
>java -classpath baobab-luna.jar baobab.exec.permutation.analyzeSignedPermutation -o -3,2,1,-4

  1. 2. 3. 4. 5.
  1> <1 1> <1 1>
  :5 :3 :1 :2 :4
:: -3 +2 +1 -4  :: 1 cycles

# of points........: 5
# of cycles........: 1
# of components....: 1
 > ajacencies......: 0
 > oriented comp...: 1
 > unoriented comp.: 0

Reversal distance..: 4
```

Figure 39: Analyze the linear permutation $(-3, 2, 1, -4)$.

```
>java -classpath baobab-luna.jar baobab.exec.permutation.analyzeSignedPermutation -o -3,2,1,-4 -l F

  1. 2. 3. 4.
  1> <1 1> ..
  :3 :1 :2 ..
:: -1 -2 +3 +4  :: 2 cycles

# of points........: 4
# of cycles........: 2
# of components....: 2
 > ajacencies......: 1
 > oriented comp...: 1
 > unoriented comp.: 0

Reversal distance..: 2
```

Figure 40: Analyze the circular permutation $(-3, 2, 1, -4)$ *(observe that the circular permutations* $(-3, 2, 1, -4)$ *and* $(-1, -2, 3, 4)$ *are equivalent).*

A.3.2 baobab.exec.permutation.sort

Sorts the given signed permutation (see Figures 41 and 42).

```
>java -classpath baobab-luna.jar baobab.exec.permutation.sort -o -3,2,1,-4

  1. 2. 3. 4. 5.
  1> <1 1> <1 1>
  :5 :3 :1 :2 :4
:: -3 +2 +1 -4  :: 1 cycles

Reversal distance: 4

Solution:

  1. 2. 3. 4. 5.
  1> <1 1> <1 1>
  :5 :3 :1 :2 :4
:: -3 +2 +1 -4  :: 1 cycles
   ---             (from [1] -3 to -3 [2])
  1. 2. 3. 4. 5.
  2> <1 2> <1 1>
  :3 :5 :1 :2 :4
:: +3 +2 +1 -4  :: 2 cycles
      ---------    (from [2] +2 to -4 [5])
  1. 2. 3. 4. 5.
  2> .. 1> <2 1>
  :4 .. :5 :1 :3
:: +3 +4 -1 -2  :: 3 cycles
   ---------       (from [1] +3 to -1 [4])
  1. 2. 3. 4. 5.
  .. <1 .. .. 1>
  .. :5 .. .. :2
:: +1 -4 -3 -2  :: 4 cycles
      ---------    (from [2] -4 to -2 [5])
  1. 2. 3. 4. 5.
  .. .. .. .. ..
  .. .. .. .. ..
:: +1 +2 +3 +4  :: 5 cycles
```

Figure 41: Sort the linear permutation $(-3, 2, 1, -4)$.

```
>java -classpath baobab-luna.jar baobab.exec.permutation.sort -l F -o -3,2,1,-4

  1. 2. 3. 4.
  1> <1 1> ..
  :3 :1 :2 ..
:: -1 -2 +3 +4  :: 2 cycles

Reversal distance: 2

Solution:

  1. 2. 3. 4.
  1> <1 1> ..
  :3 :1 :2 ..
:: -1 -2 +3 +4  :: 2 cycles
   ---          (from [1] -1 to -1 [2])
  1. 2. 3. 4.
  .. <1 1> ..
  .. :3 :2 ..
:: +1 -2 +3 +4  :: 3 cycles
      ---       (from [2] -2 to -2 [3])
  1. 2. 3. 4.
  .. .. .. ..
  .. .. .. ..
:: +1 +2 +3 +4  :: 4 cycles
```

Figure 42: Sort the circular permutation $(-3, 2, 1, -4)$ *(observe that the circular permutations* $(-3, 2, 1, -4)$ *and* $(-1, -2, 3, 4)$ *are equivalent).*

A.3.3 baobab.exec.permutation.performReversals

Performs a sequence of reversals in the given order. Each reversal is given by a pair of integers i, j where i is the index of the point that corresponds to the first extremity to the reversal and j is the index of the point that corresponds to the last extremity to the reversal $(i < j)$. Each pair of reversals must be separated by a '|', without spaces. An example is given in Figure 43.

A.3.4 baobab.exec.permutation.decomposeSignedPermutation

Decomposes a breakpoint graph of a signed permutation into its components. In each part, only one component with long cycles is preserved, the others are properly replaced by adjacencies (see Figure 44).

A.3.5 baobab.exec.trace.analyzeTraces

Enumerates the traces of all sorting sequences of reversals for the given signed permutation. This program enumerates all traces without enumerating all solutions, and counts the number of solutions represented by each trace. It returns the list of traces represented by their normal forms and the respective number of solutions in each trace. We can apply constraints to the trace construction, according to the evolutionary properties of the correponding genomes.

Attention:

1. This program does not deal with fortresses.
2. The number of traces may be huge for permutations with reversal distance greater than 12. During its execution, the program has to keep a considerable amount of data, but its intern organization allows automatic partial compression of this data (see more details in Section 6.1.1 of Chapter 6). There are parameters to set the level of compression:

```
>java -classpath baobab-luna.jar baobab.exec.permutation.performReversals
-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1 -r "11,13|3,13"

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 03> <02 <03 <02 02> ... 01> <01 <01
  :11 ... :04 :13 :07 :09 :05 :06 :08 ... :12 :03 :01
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles

Reversal distance: 8

Reversals:

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 03> <02 <03 <02 02> ... 01> <01 <01
  :11 ... :04 :13 :07 :09 :05 :06 :08 ... :12 :03 :01
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles
                                           --------     (from [11] +02 to -01 [13])
  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  04> ... 01> <01 03> <02 <03 <02 02> ... 04> 01> <01
  :11 ... :04 :13 :07 :09 :05 :06 :08 ... :01 :03 :12
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +01 -02  :: 6 cycles
           ----------------------------------------     (from [03] +12 to -02 [13])
  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  04> ... 05> <05 <04 ... <02 02> 03> 02> <03 01> <01
  :05 ... :04 :03 :01 ... :08 :10 :11 :07 :09 :13 :12
:: -04 -03 +02 -01 +05 +06 -07 -09 -10 +08 +11 -12  :: 7 cycles
```

Figure 43: Perform two subsequent reversals, the first with extremities in points 11 *and* 13 *and the second with extremities in points* 3 *and* 13*, on the linear permutation* $(-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$.

- `-X [integer]`: maximum number of traces by compressible subset (default = 3000)
- `-Y [integer]`: maximum number of uncompressed subsets (default = 400)
- `-Z [integer]`: maximum number of rules in pending insertion list by compressed subset (default = 200)

We give two examples of analyzing traces in Figures 45 and 46.

In order to construct the traces of a permutation by multiplying the traces of its components, the parameter `-e 0` must be given. Figure 47 shows an example of this approach.

Applying biological constraints

When a list of biological constraints C is applied, the program analyzes the space of all sorting sequences that respect the constraints in C and construct $C-$induced subtraces instead of full traces. This program enumerates all non-empty $C-$induced subtraces without enumerating all solutions, and counts the number of solutions represented by each $C-$induced subtrace. It returns the list of non-empty $C-$induced subtraces represented by the normal forms of the corresponding traces and the respective number of solutions in each strata-induced subtrace.

1. Analyzing the perfect traces of a permutation: the parameter `-p -1` may be given (see Figure 48). The version of this program that accepts common interval breaks is not implemented in BAOBABLUNA.

2. Analyzing the progressive perfect subtraces of a permutation, accepting at most $i \geq 0$ common interval breaks: the parameter `-p i` may be given (see Figure 49).

```
>java -classpath baobab-luna.jar baobab.exec.permutation.decomposeSignedPermutation
-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 02> 03> <02 03> <03 ... 01> <01 <01
  :11 ... :04 :13 :07 :08 :05 :09 :06 ... :12 :03 :01
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles

# of points......: 13
# of big comp....: 2
# of cycles......: 5
Reversal distance: 8

-------------------------------

Component 1

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  ... ... ... ... ... ... 03> <03 02> <03 <02 ... ...
  ... ... ... ... ... ... :10 :07 :11 :08 :09 ... ...
:: +01 +02 +03 +04 +05 +06 -07 -09 -10 +08 +11 +12  :: 10 cycles

# of points......: 13
# of big comp....: 1
# of cycles......: 10
Reversal distance: 3

-------------------------------

Component 2

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 ... ... ... ... ... ... 01> <01 <01
  :11 ... :04 :13 ... ... ... ... ... ... :12 :03 :01
:: -04 -03 +12 -11 -10 -09 -08 -07 -06 -05 +02 -01  :: 8 cycles

# of points......: 13
# of big comp....: 1
# of cycles......: 8
Reversal distance: 5

-------------------------------
```

Figure 44: Decompose the breakpoint graph of the linear permutation π into its two non-trivial components, for $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1$.

3. Analyzing the $(p, q, r)-$induced subtraces of a permutation (apply the terminus-symmetry constraint where p is the maximum number of external reversals, q is the maximum number of terminus-asymmetric reversals and r is the threshold to the terminus-symmetry rate): the parameter `-h "`$p|q|r$`"` may be given. Optionally, the parameter `-w [originPermutation lengths]` can be set, to define the lengths of the markers with respect to the order given in the origin permutation. The lengths are given in a list of double values separated by commas, without spaces. Example: the parameters `-o -2,3,4,-1 -w 1.5,2,3.8,2.4` assign the length 1.5 to marker 2, 2 to marker 3, 3.8 to marker 4, and 2.4 to marker 1 (see Figure 50).

4. Analyzing the strata-induced subtraces of a permutation: the permutation must be linear, the first stratum is assumed to start in point 1, then each stratum starts in the end point of the previous stratum. The strata end points may be given by the following parameter: `-s [points]`: points may be separated by commas, without spaces. Example:

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces -o -3,2,1,-4

  1. 2. 3. 4. 5.
  1> <1 1> <1 1>
  :5 :3 :1 :2 :4
:: -3 +2 +1 -4  :: 1 cycles

summary:

Reversal distance: 4

4-traces:

{1}{1.-.3}{2}{4} : [24]
{1.2.4}{3} < {1.3.4} < {2.-.4} : [4]

Total number of 4-traces: 2
Total number of solutions: 28
```

Figure 45: Analyze traces of the linear permutation $(-3, 2, 1, -4)$.

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces
-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 03> <02 <03 <02 02> ... 01> <01 <01
  :11 ... :04 :13 :07 :09 :05 :06 :08 ... :12 :03 :01
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles

summary

Reversal distance: 8

8-traces

{1.2}{1.2.5.-.12}{2}{7}{8.10}{12} < {1.-.4}{8.9} : [10080]
{1.-.12}{2}{3.4.12}{5.-.11}{7}{8.10} < {3.-.11}{8.9} : [10080]
{1.-.12}{2.-.12}{2.5.-.12}{7}{8.10}{12} < {2.-.4}{8.9} : [10080]
{1.2}{7}{8.10} < {1.5.-.11}{8.9} < {1.3.4.12} < {2.-.12}{3.-.11} : [336]
{2.-.12}{7}{8.10} < {1.3.4.12}{8.9} < {1.5.-.11} < {1.2}{3.-.11} : [336]
{2.5.-.12}{5.-.11}{7}{8.10} < {1.12}{8.9} < {1.5.-.11} < {1.-.4} : [840]

Total number of 8-traces: 6
Total number of solutions: 31752
```

Figure 46: Analyze traces of the linear permutation $(-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$.

`-s 7,15,23` (see Figure 51).

5. Compatibility between constraints: some contraints are compatible and can be applied together. For example, the terminus-symmetry (defined by the parameter `-h`) and the initial or progressive detection of common intervals (defined by the parameter `-p`) can be used simultaneously (see Figure 52). However, if we try to run the program applying incompatible contraints, such as strata (parameter `-s`) together with progressive detection of common intervals (parameter `-p 0`) the program will not be launched and an error message will be written in the output (Figure 53).

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces
-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1 -e 0

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 03> <02 <03 <02 02> ... 01> <01 <01
  :11 ... :04 :13 :07 :09 :05 :06 :08 ... :12 :03 :01
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles

summary

Reversal distance: 8

---------------------------

Component C1 (Reversal distance: 3)

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  ... ... ... ... ... ... 03> <03 02> <03 <02 ... ...
  ... ... ... ... ... ... :10 :07 :11 :08 :09 ... ...
:: +01 +02 +03 +04 +05 +06 -07 -09 -10 +08 +11 +12  :: 10 cycles

3-traces

{7}{8.10} < {8.9} : [3]

---------------------------

Component C2 (Reversal distance: 5)

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 ... ... ... ... ... ... 01> <01 <01
  :11 ... :04 :13 ... ... ... ... ... ... :12 :03 :01
:: -04 -03 +12 -11 -10 -09 -08 -07 -06 -05 +02 -01  :: 8 cycles

5-traces

{1.2}{1.2.5.-.12}{2}{12} < {1.-.4} : [60]
{1.-.12}{2}{3.4.12}{5.-.11} < {3.-.11} : [60]
{1.-.12}{2.-.12}{2.5.-.12}{12} < {2.-.4} : [60]
{1.2} < {1.5.-.11} < {1.3.4.12} < {2.-.12}{3.-.11} : [2]
{2.-.12} < {1.3.4.12} < {1.5.-.11} < {1.2}{3.-.11} : [2]
{2.5.-.12}{5.-.11} < {1.12} < {1.5.-.11} < {1.-.4} : [5]

---------------------------

8-traces (obtained from the multiplication of the traces of C1 and C2)

{1.2}{1.2.5.-.12}{2}{7}{8.10}{12} < {1.-.4}{8.9} : [180]
{1.-.12}{2}{3.4.12}{5.-.11}{7}{8.10} < {3.-.11}{8.9} : [180]
{1.-.12}{2.-.12}{2.5.-.12}{7}{8.10}{12} < {2.-.4}{8.9} : [180]
{1.2}{7}{8.10} < {1.5.-.11}{8.9} < {1.3.4.12} < {2.-.12}{3.-.11} : [6]
{2.-.12}{7}{8.10} < {1.3.4.12}{8.9} < {1.5.-.11} < {1.2}{3.-.11} : [6]
{2.5.-.12}{5.-.11}{7}{8.10} < {1.12}{8.9} < {1.5.-.11} < {1.-.4} : [15]

Total number of 8-traces: 6
Total number of solutions: 567
```

Figure 47: Analyze traces of the linear permutation $(-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$ *by multiplying the traces of its components* $C1$ *and* $C2$*. To obtain the real number of sequences in each trace and the total number of solutions, we must multiply the given number of sequences in each trace and the given number of solutions by* $M(3, 5) = 56$*.*

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces -p -1
-o -5,-2,-7,4,-8,3,6,-1

1. 2. 3. 4. 5. 6. 7. 8. 9.
  1> <1 1> <1 1> 1> <1 1> <1
  :7 :5 :6 :8 :9 :2 :4 :3 :1
:: -5 -2 -7 +4 -8 +3 +6 -1  :: 1 cycles

summary

Reversal distance: 8

8-traces

{1.-.8}{2}{2.-.5.7.8}{2.4.7} < {2.3.7.8}{2.8} < {2.-.6}{7.8} : [1288]
{1.-.8}{2}{2.-.5.7.8}{2.4.7}{3.8} < {2.4.-.7}{4.-.6} < {3.-.7} : [2072]
...
{1.-.8}{2.4.7.8} < {2.-.4.6.7} < {2.3.5.-.8} < {2.-.6.8} < {4.5.7.8} < {4.6.7} < {5.-.7} : [8]
{1.-.8}{3.4.6.8} < {2.6.7} < {2.-.4.8} < {4.6.7} < {3.6.-.8} < {3.-.5.8} < {3.-.7} : [8]

Total number of 8-traces: 92
Total number of solutions: 51304
```

Figure 48: Analyze perfect traces of the linear permutation $(-5, -2, -7, 4, -8, 3, 6, -1)$.

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces -p 0
-o -5,-2,-7,4,-8,3,6,-1

1. 2. 3. 4. 5. 6. 7. 8. 9.
  1> <1 1> <1 1> 1> <1 1> <1
  :7 :5 :6 :8 :9 :2 :4 :3 :1
:: -5 -2 -7 +4 -8 +3 +6 -1  :: 1 cycles

summary

Reversal distance: 8

8-traces

{1.-.8}{2}{2.-.5.7.8}{2.4.-.7}{3.8}{2.4.7}{4.-.6}{3.-.7} (representative)
{1.-.8}{2}{2.-.5.7.8}{2.4.7}{3.8} < {2.4.-.7}{4.-.6} < {3.-.7} (normal form)
[112] (size)

{1.-.8}{2}{2.-.5.7.8}{3.8}{4}{3.4.7}{2.-.4}{2.-.6} (representative)
{1.-.8}{2}{2.-.5.7.8}{3.8}{4} < {3.4.7} < {2.-.4}{2.-.6} (normal form)
[1344] (size)

...

{1.-.8}{2.-.5.7.8}{2.4.-.7}{2.3.5.-.8}{2.-.6.8}{3.4.7.8}{3.-.7}{5.-.7} (representative)
{1.-.8}{2.-.5.7.8} < {2.4.-.7} < {2.3.5.-.8} < {2.-.6.8} < {3.4.7.8} < {3.-.7}{5.-.7} (normal form)
[16] (size)

Total number of 8-subtraces: 12
Total number of solutions: 11568
```

Figure 49: Analyze progressive perfect subtraces of $(-5, -2, -7, 4, -8, 3, 6, -1)$*, which is a linear permutation.*

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces -l F -h "3|0|0.7"
-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1 -w 39,42,44,125,14,30,32,391,18,47,100,176

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12.
  ... 01> <01 <01 02> <02 02> <02 <02 02> <02 01>
  ... :03 :12 :02 :11 :09 :05 :06 :07 :08 :10 :04
:: +01 +03 -02 -11 +05 -09 -10 +08 +06 -07 -04 +12  :: 3 cycles

summary

Reversal distance: 9

9-traces

{2.3}{3}{4.-.11}{5.-.10}{6}{6.-.8.10}{6.8}{6.-.9}{7.8} (representative)
{2.3}{3}{4.-.11}{5.-.10}{6}{6.-.8.10}{6.8} < {6.-.9}{7.8} (normal form)
[60480] (size)

{2.3}{3}{4.-.11}{6.8.-.10}{8.10}{5.-.10}{6}{7.-.10}{8.9} (representative)
{2.3}{3}{4.-.11}{5.-.10}{6}{6.8.-.10}{8.10} < {7.-.10}{8.9} (normal form)
[50904] (size)

{2.3}{3}{8.10}{5.8.-.10}{5.6.8.9}{5.7.-.9}{4.-.11}{6}{6.-.9} (representative)
{2.3}{3}{4.-.11}{5.8.-.10}{6}{8.10} < {5.6.8.9} < {5.7.-.9} < {6.-.9} (normal form)
[1008] (size)

{2.3}{3}{6.8.-.10}{5.6.8.10}{5.6.8.9}{5.-.8}{4.-.11}{6}{7.8} (representative)
{2.3}{3}{4.-.11}{6}{6.8.-.10} < {5.6.8.10} < {5.6.8.9} < {5.-.8}{7.8} (normal form)
[1512] (size)

Total number of 9-subtraces: 4
Total number of solutions: 113904
```

Figure 50: Apply the terminus-symmetry constraint for searching $(p, q, r)-$*induced subtraces with* $p = 3$ *(maximum number of external reversals),* $q = 0$ *(maximum number of terminus-asymmetric reversals) and* $r = 0.7$ *(threshold to the terminus-symmetry rate) of the circular permutation* $(1, 3, -2, -11, 5, -9, -10, 8, 6, -7, -4, 12)$ *with the respective marker lengths 39, 42, 44, 125, 14, 30, 32, 391, 18, 47, 100, and 176.*

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces
-s 3,11,13 -o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12. 13.
  01> ... 01> <01 03> <02 <03 <02 02> ... 01> <01 <01
  :11 ... :04 :13 :07 :09 :05 :06 :08 ... :12 :03 :01
:: -04 -03 +12 -11 -08 +10 +09 +07 -06 -05 +02 -01  :: 5 cycles

summary

Reversal distance: 8

Strata end points:
> 3 (between -3 and 12)
> 11 (between -5 and 2)
> 13 (after -1)

8-traces

{1.2}{2}{1.2.5.-.12}{7}{8.10}{12}{8.9}{1.-.4} (representative)
{1.2}{1.2.5.-.12}{2}{7}{8.10}{12} < {1.-.4}{8.9} (normal form)
[420] (size)

Total number of 8-subtraces: 1
Total number of solutions: 420
```

Figure 51: Analyze strata-induced subtraces with strata end points in (03.), (11.) and (13.), for the linear permutation $(-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$.

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces -l F -h "3|0|0.7" -p 2
-o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1 -w 39,42,44,125,14,30,32,391,18,47,100,176

  01. 02. 03. 04. 05. 06. 07. 08. 09. 10. 11. 12.
  ... 01> <01 <01 02> <02 02> <02 <02 02> <02 01>
  ... :03 :12 :02 :11 :09 :05 :06 :07 :08 :10 :04
:: +01 +03 -02 -11 +05 -09 -10 +08 +06 -07 -04 +12  :: 3 cycles

summary

Reversal distance: 9

9-traces

{2.3}{3}{4.-.11}{5.-.10}{6}{6.-.8.10}{6.8}{6.-.9}{7.8} (representative)
{2.3}{3}{4.-.11}{5.-.10}{6}{6.-.8.10}{6.8} < {6.-.9}{7.8} (normal form)
[28224] (size)

{2.3}{3}{4.-.11}{6.8.-.10}{8.10}{5.-.10}{6}{7.-.10}{8.9} (representative)
{2.3}{3}{4.-.11}{5.-.10}{6}{6.8.-.10}{8.10} < {7.-.10}{8.9} (normal form)
[25200] (size)

{2.3}{3}{6.8.-.10}{5.6.8.10}{5.6.8.9}{5.-.8}{4.-.11}{6}{7.8} (representative)
{2.3}{3}{4.-.11}{6}{6.8.-.10} < {5.6.8.10} < {5.6.8.9} < {5.-.8}{7.8} (normal form)
[1512] (size)

Total number of 9-subtraces: 3
Total number of solutions: 54936
```

Figure 52: Apply the progressive detection of common intervals, accepting at most two interval breaks, together with the terminus-symmetry constraint with at most 3 external and no terminus-asymmetric reversals for a threshold of 0.7 to the terminus-symmetry rate, for searching subtraces of the circular permutation $(1, 3, -2, -11, 5, -9, -10, 8, 6, -7, -4, 12)$ *with the respective marker lengths 39, 42, 44, 125, 14, 30, 32, 391, 18, 47, 100, and 176.*

```
>java -classpath baobab-luna.jar baobab.exec.trace.analyzeTraces
-s 3,11,13 -o -4,-3,12,-11,-8,10,9,7,-6,-5,2,-1 -p 0

ABORTED: It is not possible to consider strata together with progressive detection of common intervals.
```

Figure 53: Strata (parameter `-s`*) and progressive detection of common intervals (parameter* `-p`*) are incompatible constraints when analyzing traces of a linear permutation. The program is aborted.*

Appendix B

Dealing with duplications

Summary

B.1 Repetition-free longest common subsequence 126

B.2 Hardness analysis 126

B.2.1 MAX 2,3-SAT 126

B.2.2 $L-$reducing MAX 2,3-SAT to RFLCS 127

B.3 Experiments 130

B.4 Final remarks 130

In this work we mostly talked about sorting by reversals without duplications. Although this model is valid and can be applied to the analysis of real cases, it has an important limitation. In practice, we often can not determine a one-to-one relation between the genes of the analyzed genomes. A genome can contain more than one copy of a gene and it may be difficult to decide which copy in one genome corresponds to a copy in the other [56]. All copies or duplications of a same gene can also be called a *family of genes*.

In the genome rearrangement domain, gene duplication is rarely considered as it usually makes the problem at hand harder. Sankoff [57] proposed the so called *exemplar model*, which consists in searching, for each family of duplicated genes, an exemplar representative in each genome. In biological terms, the exemplar gene may correspond to the original copy of the gene, which later originated all other copies. Following the parsimony principle, the choices of exemplars should be made so as to minimize the breakpoint (that is, the number of breakpoints) and/or the reversal distance between the two simpler versions of both genomes, composed only by the exemplar genes. An alternative to the exemplar model is the *multigene family model*, which consists in maximizing the number of paired genes among a family. Here the gene pairs should be chosen so as to minimize the breakpoint distance between the genomes. Both exemplar and multigene models were proven to lead to NP-hard problems [14, 20].

A study by Adi *et al.* [1] propose a similarity measure between two genomes, based on the length of a special kind of longest common subsequence [12, 24], that the authors called *repetition-free longest common subsequence* (RFLCS). We participated in the hardness analysis of computing this measure, as we will describe in this chapter.

B.1 Repetition-free longest common subsequence

To compare two genomes represented as sequences of symbols, Adi *et al.* [1] propose a similarity measure that takes into account the concept of exemplar genes. The measure is based on the length of a repetition-free longest common subsequence (LCS) between the two sequences. Only deletions of symbols from the original sequences are allowed to compute this special kind of LCS, and, furthermore, for each family with k duplicated genes in one sequence, at least $k-1$ of them must be deleted. The concept behind the exemplar model is captured by the repetition-free requirement in the sense that at most one representative of each family of duplicated genes in each sequence is taken into account. The length of a repetition-free LCS is a measure of the similarity between the sequences, so it can be used to compute a distance between two sequences (or genomes).

An *alphabet* is a finite set and we refer to each of its elements as a *symbol*. In the context of genome rearrangements, each symbol represents a gene, thus if one symbol occurs more than once in a sequence, this means that the corresponding gene is duplicated on the genome. All considered sequences are finite and over some alphabet usually implicit, as it may be considered to be the set of all symbols appearing in the involved sequences. For a sequence w, we use $|w|$ to denote the length of w. The problem we are interested in, denoted by RFLCS, consists of the following: given two sequences x and y, find a repetition-free LCS of x and y. We write RFLCS(x, y) when we refer to RFLCS for a generic instance consisting of a pair (x, y). We denote by opt(RFLCS(x, y)) the length of an optimal solution of RFLCS(x, y).

Bonizzoni *et al.* [15] considered some variants of the RFLCS. For instance, they considered the case where some symbols are required to appear in the sought LCS, and possibly more than once. They showed that these variants are APX-hard and that, in some cases, it is NP-complete just to decide whether an instance of the variants is feasible. This second complexity result makes these variants less tractable.

Adi *et al.* [1] presented some algorithmic results for the RFLCS. First, they showed some polynomial cases of RFLCS. If one of the considered sequences (or genomes) is free of duplications, for instance, the problem can be simply handled as LCS, which can be solved in polynomial time for a fixed number of sequences [12, 24]. Then the authors presented three approximation algorithms for RFLCS. They described some c-approximations for the case where each symbol appears at most c times in at least one of the sequences. They also proposed an integer linear programming formulation (IP) for RFLCS.

We participated in the hardness analysis of RFLCS, proving that RFLCS is APX-hard even when each symbol appears at most twice in both sequences.

B.2 Hardness analysis

We show that RFLCS(x, y) is APX-hard, by presenting an L-reduction [51] (a special kind of reduction which preserves APX-hardness) from a particular version of MAX 2-SAT, which is known to be APX-complete, to RFLCS.

B.2.1 MAX 2,3-SAT

Let V be a set of boolean variables. Denote by $\overline{v}$ the negation of a variable v and, generalizing this notation for a given set V, let $\overline{V} = \{\overline{v} : v \in V\}$. Recall a clause c over V is a set of literals, where a literal is an element from $V \cup \overline{V}$. We refer to a clause with k literals as a *k-clause*. An *assignment* for V is a function $a : V \to \{\textbf{true}, \textbf{false}\}$. A literal ℓ is **true** according to a if, for

$V = \{v_1, v_2, v_3\}$
$C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$

$c_1 = \{v_1, v_2\}$	$c_4 = \{v_1, v_3\}$	$c_7 = \{v_2, v_3\}$
$c_2 = \{\overline{v_1}, v_2\}$	$c_5 = \{v_1, \overline{v_3}\}$	$c_8 = \{\overline{v_2}, v_3\}$
$c_3 = \{\overline{v_1}, \overline{v_2}\}$	$c_6 = \{\overline{v_1}, \overline{v_3}\}$	$c_9 = \{\overline{v_2}, \overline{v_3}\}$

One optimal solution:

$a(v_1) =$ **true**, $a(v_2) =$ **false**, and $a(v_3) =$ **true**

Figure 54: An instance of MAX 2,3-SAT *with one optimal solution. Observe that all the clauses in C have two literals and, moreover, each literal in $V \cup \overline{V}$ appears in at most three clauses in C.*

some v in V, either $\ell = v$ and $a(v) =$ **true**, or $\ell = \overline{v}$ and $a(v) =$ **false**. A clause c is *satisfied* by an assignment a if at least one of its literals is **true** according to a.

For a set C of 2-clauses over V, where each literal in $V \cup \overline{V}$ may appear in at most three clauses in C, MAX 2,3-SAT(V, C) is the problem of finding an assignment for V that maximizes the number of satisfied clauses in C. This variant of MAX 2-SAT is APX-complete [3, 51]. We assume that, for any v in V, no clause is of the form $\{v, \overline{v}\}$. For an assignment a, we denote by val(MAX 2,3-SAT$(V, C), a)$ the number of clauses in C that are satisfied by a and let opt(MAX 2,3-SAT$(V, C)) = \max\{$val(MAX 2,3-SAT$(V, C), a) : a$ be an assignment for $V\}$. Figure 54 shows an instance of MAX 2,3-SAT with one corresponding optimal solution.

B.2.2 $L-$reducing MAX 2,3-SAT to RFLCS

Recall that opt(RFLCS$(x, y)) = \max\{|w| : w$ is a repetition-free subsequence of x and $y\}$. An L-*reduction* from MAX 2,3-SAT to RFLCS consists of a pair of polynomial-time computable functions (f, g) such that, for two fixed positive constants α and β, the following two conditions hold:

(C1) for every instance (V, C) of MAX 2,3-SAT, $f(V, C) = (x, y)$ is an instance of RFLCS, and opt(RFLCS$(x, y)) \leq \alpha\,$opt(MAX 2,3-SAT$(V, C))$;

(C2) for every instance (V, C) of MAX 2,3-SAT, and every repetition-free subsequence w of x and y, where $(x, y) = f(V, C)$, we have that $a = g(V, C, w)$ is an assignment for V, and opt(MAX 2,3-SAT$(V, C))$ − val(MAX 2,3-SAT$(V, C), a) \leq \beta\,($opt(RFLCS$(x, y)) - |w|)$.

Theorem 2 *Let x and y be two sequences over the same alphabet F. If, for each symbol f in F, the number of occurrences of f in x or y is bounded by two, then* RFLCS(x, y) *is APX-complete.*

Proof.
Adi *et al.* [1] proposed c-approximations for the case where each symbol appears at most c times in at least one of the sequences. For the particular case where each symbol appears at most 2 times in at least one of the sequences (when $m_a(x, y) \leq 2$), we have 2-approximations for RFLCS(x, y), so the case of RFLCS(x, y) addressed by this theorem is in APX. Next we show an L-reduction from MAX 2,3-SAT to RFLCS.

For an instance (V, C) of MAX 2,3-SAT, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of boolean variables and C is a set of 2-clauses over V, we describe an instance $(x, y) = f(V, C)$ of RFLCS. Let $\{c_1, c_2, \ldots, c_m\}$ be a set of distinct labels, one for each of the clauses in C. We abuse notation

$$V = \{v_1, v_2, v_3\}$$
$$C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$$

$$\begin{array}{lll} c_1 = \{v_1, v_2\} & c_4 = \{v_1, v_3\} & c_7 = \{v_2, v_3\} \\ c_2 = \{\overline{v_1}, v_2\} & c_5 = \{v_1, \overline{v_3}\} & c_8 = \{\overline{v_2}, v_3\} \\ c_3 = \{\overline{v_1}, \overline{v_2}\} & c_6 = \{\overline{v_1}, \overline{v_3}\} & c_9 = \{\overline{v_2}, \overline{v_3}\} \\ s(v_1) = c_1c_4c_5 & s(v_2) = c_1c_2c_7 & s(v_3) = c_4c_7c_8 \\ s(\overline{v_1}) = c_2c_3c_6 & s(\overline{v_2}) = c_3c_8c_9 & s(\overline{v_3}) = c_5c_6c_9 \end{array}$$

$$D = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}, d_{11}, d_{12}\}$$

$$x = \overbrace{\underline{c_1c_4c_5}}^{s(v_1)}\overbrace{c_2c_3c_6}^{s(\overline{v_1})}\underline{d_1d_2d_3d_4d_5d_6}\overbrace{c_1c_2c_7}^{s(v_2)}\overbrace{\underline{c_3c_8c_9}}^{s(\overline{v_2})}\underline{d_7d_8d_9d_{10}d_{11}d_{12}}\overbrace{c_4\underline{c_7}c_8}^{s(v_3)}\overbrace{c_5c_6c_9}^{s(\overline{v_3})}$$
$$y = \underbrace{c_2c_3c_6}_{s(\overline{v_1})}\underbrace{\underline{c_1c_4c_5}}_{s(v_1)}\underline{d_1d_2d_3d_4d_5d_6}\underbrace{\underline{c_3c_8c_9}}_{s(\overline{v_2})}\underbrace{c_1c_2c_7}_{s(v_2)}\underline{d_7d_8d_9d_{10}d_{11}d_{12}}\underbrace{c_5c_6c_9}_{s(\overline{v_3})}\underbrace{c_4\underline{c_7}c_8}_{s(v_3)}$$

Figure 55: L*−reducing* MAX 2,3-SAT(V,C) *to* RFLCS(x,y)*: an optimal solution for* RFLCS(x,y) *is given by the subsequence* $c_1c_4c_5d_1d_2d_3d_4d_5d_6c_3c_8c_9d_7d_8d_9d_{10}d_{11}d_{12}c_7$*, that corresponds to an optimal solution for* MAX 2,3-SAT(V,C)*, satisfying the clauses* c_1*,* c_3*,* c_4*,* c_5*,* c_7*,* c_8*, and* c_9 *with the assignment* $a(v_1) = $ **true***,* $a(v_2) = $ **false***, and* $a(v_3) = $ **true***.*

and use c_i to refer both to the label c_i and to the clause whose label is c_i. So in particular we denote also by C the set of labels $\{c_1, c_2, \ldots, c_m\}$.

For each literal ℓ, we denote by $s(\ell)$ a sequence composed by the (labels of the) clauses in which ℓ is present, taken in an arbitrary order. Thus, for each v in V and an assignment a for V, the sequence $s(v)$ contains the clauses of C that would be satisfied if $a(v) = $ **true** and the sequence $s(\overline{v})$ contains the clauses of C that would be satisfied if $a(v) = $ **false**. Observe that, since we do not have a clause of the form $\{v, \overline{v}\}$, then $s(v)$ and $s(\overline{v})$ have no common symbol. In addition, as each literal ℓ may appear in at most three clauses of C, we have that $|s(\ell)| \leq 3$.

We also use a new set of symbols $D = \{d_1, d_2, \ldots, d_k\}$, such that $k = 6(n-1)$ and $D \cap C = \emptyset$, and take the sequences x and y to be as follows (an example of this construction is available in Figure 55):

$x = s(v_1)s(\overline{v_1})d_1d_2d_3d_4d_5d_6s(v_2)s(\overline{v_2})d_7d_8d_9d_{10}d_{11}d_{12} \cdots d_{k-5}d_{k-4}d_{k-3}d_{k-2}d_{k-1}d_k s(v_n)s(\overline{v_n})$

and

$y = s(\overline{v_1})s(v_1)d_1d_2d_3d_4d_5d_6s(\overline{v_2})s(v_2)d_7d_8d_9d_{10}d_{11}d_{12} \cdots d_{k-5}d_{k-4}d_{k-3}d_{k-2}d_{k-1}d_k s(\overline{v_n})s(v_n)$.

The alphabet adopted is the set $C \cup D$. By definition, the sets C and D are disjoint and each symbol of D occurs once in both x and y. In addition, as each clause c in C has two literals, and, for each ℓ in $V \cup \overline{V}$, the correspondent sequence $s(\ell)$ appears once in either x or y, the number of occurrences of each symbol c in x is equal to two, and the same is valid for y.

Note that the described $(x,y) = f(V,C)$ can be computed in polynomial time. Since all clauses have two literals, then $n \leq 2m$, where $n = |V|$ and $m = |C|$. Also, each symbol of the adopted alphabet may appear at most once in a repetition-free subsequence of x and y, thus opt(RFLCS(x,y)) $\leq m+6(n-1) \leq 12m$. On the other hand, we can easily set an assignment a for V, such that val(MAX 2,3-SAT$(V,C), a) \geq m/2$: sequentially, for $i = 1, 2, \ldots, n$, define $C_i \subseteq C$ as $C_i = \{c \in C : c$ contains either v_i or $\overline{v_i}\}$ and $C = C \setminus C_i$; then make $a(v_i) = $ **true** if v_i is more common than $\overline{v_i}$ in the clauses of C_i, otherwise $a(v) = $ **false**. Note that the final C is empty and a satisfies at least $|C_i|/2$ clauses from C_i, for each i. Therefore, as $\cup C_i$ is equal to the initial C, the assignment a satisfies at least $m/2$ clauses from the initial C. So opt(MAX 2,3-SAT(V,C)) $\geq m/2$.

Putting the two together, we conclude that opt(RFLCS(x,y)) ≤ 24 opt(MAX 2,3-SAT(V,C)), and **(C1)** holds with $\alpha = 24$.

The next claim is used to prove that **(C2)** also holds. In particular, its proof includes the description of the function g.

Claim 1 *Let (V,C) be an instance of* MAX 2,3-SAT *and $(x,y) = f(V,C)$. There is a repetition-free subsequence w of x and y of length at least $p = q + |D|$ if and only if there is an assignment a for V that satisfies at least $q = p - |D|$ clauses of C.*

Proof. Let w be a repetition-free subsequence of x and y of length p. First we describe another repetition-free subsequence z of x and y of length at least p that contains all symbols in D. From z, we describe an assignment a that satisfies at least $|z| - |D| \geq p - |D|$ clauses of C.

The following procedure constructs z from w, so that z is a supersequence of $d_1 d_2 \cdots d_k$ and is at least as long as w. Sequentially, for $i = 1, 2, \ldots, n-1$, remove any symbol of w that comes from the alignment of a symbol of $s(v_i)s(\overline{v_i})$ in x (of $s(\overline{v_i})s(v_i)$ in y) and a symbol of $s(\overline{v_{i+1}})s(v_{i+1})d_{6(i+1)-5} \cdots d_k s(\overline{v_n})s(v_n)$ in y (of $s(v_{i+1})s(\overline{v_{i+1}})d_{6(i+1)-5} \cdots d_k s(v_n)s(\overline{v_n})$ in x, respectively), and then add $d_{6i-5}d_{6i-4}d_{6i-3}d_{6i-2}d_{6i-1}d_{6i}$ (which are the symbols from D that are between $s(v_i)$ and $s(v_{i+1})$ in x or y). Call z the resulting subsequence after all these substitutions. At the end, add to z the symbols from D that are not already present in it. Observe that $|s(v_i)s(\overline{v_i})| \leq 6$, thus at each step we replace at most six symbols by exactly six new symbols from D (that do not occur in w). Hence z is also a repetition-free subsequence of x and y, with $|z| \geq |w|$.

We now proceed to describe the assignment a. Since z contains all symbols from D, the other portions of z are subsequences of $s(v_i)s(\overline{v_i})$ in x and $s(\overline{v_i})s(v_i)$ in y, for each $i = 1, 2, \ldots, n$. Moreover, because all symbols in $s(v_i)$ differ from those in $s(\overline{v_i})$, the subsequence z does not align simultaneously symbols from both $s(v_i)$ and $s(\overline{v_i})$. So we define an assignment a as $a(v_i) =$ **true** if z aligns a symbol from $s(v_i)$, otherwise $a(v_i) =$ **false**. Set $g(V,C,w) = a$. Observe that assignment a satisfies at least $q = |z| - |D| \geq p - |D|$ clauses.

For the other direction, consider an assignment a for V that satisfies q clauses of C. Let w be a repetition-free subsequence of x and y obtained as follows. For $i = 1, 2, \ldots, n$, add to w the symbols that correspond to the clauses in $s(v_i)$ if $a(v_i) =$ **true**, otherwise add the symbols that correspond to the clauses in $s(\overline{v_i})$. After all the additions, eliminate repetitions and add to w, at the corresponding positions, all symbols from D. Then w is a repetition-free subsequence of x and y such that $|w| = q + |D|$. □

We now go back to the proof of Theorem 2. First note that the assignment $a = g(V,C,w)$ described in the proof of Lemma 1 can be obtained in polynomial time. So g is a polynomially computable function.

Due to Lemma 1, for each optimal solution for RFLCS(x,y) there is a corresponding solution for MAX 2,3-SAT(V,C) that satisfies at least opt(RFLCS(x,y)) $- |D|$ clauses from C. Analogously, an optimal solution for MAX 2,3-SAT(V,C) corresponds to a solution for RFLCS(x,y) of length at least opt(MAX 2,3-SAT(V,C)) $+ |D|$. Thus, we have that opt(RFLCS(x,y)) $=$ opt(MAX 2,3-SAT(V,C)) $+ |D|$.

Also according to Lemma 1, there is a repetition-free subsequence w of x and y of length p if and only if there is an assignment a for V that satisfies at least $p - |D|$ clauses of C. Let $g(V,C,w) = a$ and note that a can be obtained in polynomial time from V, C, and w. Also, we have that val(MAX 2,3-SAT$(V,C), a$) $\geq |w| - |D|$. So, opt(MAX 2,3-SAT(V,C)) $-$

val(MAX 2,3-SAT(V, C, a)) $\leq$ opt(RFLCS(x, y)) $- |D| - (|w| - |D|) =$ opt(RFLCS(x, y)) $- |w|$, and **(C2)** holds with $\beta = 1$. □

B.3 Experiments

Adi *et al.* [1] described three c-approximations for the case where each symbol appears at most c times in at least one of the sequences. The authors presented also an integer linear programming formulation (IP) for RFLCS and showed some computational results obtained for RFLCS, considering the three approximation algorithms and the use of the proposed formulation to the IP solver for finding optimal solutions of the tested instances.

The experimental results obtained by Adi *et al.* [1] indicate that the performance of the approximation algorithms is quite satisfactory for the instance sizes tested, that have length at most 512. However, the authors consider that it would be nice to test the performance on larger instances. The limitation is that for larger instances, especially when the sequences have many repetitions (small alphabet), the solution of the approximation algorithms can be computed very fast, but often the optimal value can not be computed. The authors are working on the branch and cut algorithm to solve larger instances, and expect to be able to find out whether there is a constant approximation algorithm for RFLCS.

B.4 Final remarks

We participated in a study by Adi *et al.* [1], that proposes a similarity measure between two genomes with gene duplications, based on the length of a special kind of longest common subsequence [12, 24]. The authors called this measure repetition-free longest common subsequence (RFLCS).

Adi *et al.* [1] presented some algorithmic results for the RFLCS. In particular, they showed some polynomial cases and three approximation algorithms for RFLCS. They described some c-approximations for the case where each symbol appears at most c times in at least one of the sequences. They also proposed an integer linear programming formulation (IP) for RFLCS. We participated in the hardness analysis of RFLCS, proving that RFLCS is APX-hard even when each symbol appears at most twice in both sequences.

Adi *et al.* [1] showed some computational results obtained for RFLCS, considering the three approximation algorithms and the use of the proposed formulation to the IP solver for finding optimal solutions of the tested instances. The experimental results indicate that the performance of the approximation algorithms is quite satisfactory for the instance sizes tested (that have length at most 512).

For the complete study of Adi *et al.* [1], including computational results and proofs, see `http://www.ime.usp.br/~cris/publ/rflcs.pdf`.

Appendix C

Extended abstract in French

Summary

C.1 Introduction **131**
C.2 Permutations, intervalles et inversions **132**
C.3 Le tri par inversions et son espace de solutions **133**
C.3.1 Traces 133
C.3.2 Un algorithme pour énumérer directement les traces 135
C.4 Contraintes biologiques **136**
C.5 Applications **139**
C.5.1 Analyse des chromosomes sexuels humains X et Y 139
C.5.2 Analyse des bactéries *Rickettsia* 140
C.6 baobabLuna **140**
C.7 Conclusions et limitations **140**
C.8 Perspectives **141**

C.1 Introduction

Notre travail concerne l'algorithmique des réarrangements de génomes. On se concentre principalement sur le problème de comparer deux génomes avec l'objectif de déterminer des séquences d'inversions pour transformer un génome dans un autre.

Les génomes subissent des constantes mutations au cours de l'évolution. Ces mutations peuvent être de petite échelle, comme les polymorphismes d'un seule nucléotide ou *single nucleotide polymorphisms* (SNPs), ou de large échelle, comme les inversions, insertions, délétions, transpositions, fusions et fissions de chromosomes. Les inversions sont des événements de large échelle observés très fréquemment, spécialement dans l'évolution des prokaryotes [13]. Chez les eukaryotes, les inversions ont aussi été observées et actuellement il existe des théories qui attribuent aux inversions un rôle majeur dans l'évolution des chromosomes sexuels X et Y chez les mammifères et aussi chez des autres organismes [40, 55, 60].

Parmi les problèmes les plus étudiés dans l'approche algorithmique de la comparaison de deux génomes différents sont celui de calculer la distance de réarrangement, c'est-à-dire, le nombre minimum d'événements nécessaires pour transformer un génome dans un autre, et celui de déterminer une séquence optimale d'événements qui transforme un génome dans un autre. Quand les événements sont restreints aux inversions et les duplications de gènes ne sont pas acceptées, il existe des algorithmes polynomiaux pour résoudre ces deux problèmes [32, 33], qui sont alors nommés la distance d'inversions et le tri par inversions. Néanmoins, le nombre de séquences optimales différentes est très grand, et il faut alors considérer d'autres critères pour pouvoir réaliser une analyse plus précise.

Une stratégie possible est celle de chercher les séquences qui respectent certaines contraintes biologiques, comme par exemple les intervalles communs, qui sont les ensembles de gènes colocalisés dans les génomes analysés - une séquence d'inversions qui ne sépare pas les intervalles communs doit être plus réaliste qu'une séquence qui sépare. Une autre approche est celle de générer l'univers de toutes les séquences optimales, grace à un algorithme proposé par Siepel [59]. Comme cet ensemble peut être trop grand pour être interpreté, un modèle pour regrouper des sous-ensembles de séquences optimales dans des classes d'équivalence a été proposé par Bergeron *et al.* [9], ce qui permet de réduire la taille de l'ensemble à traiter. Mais le problème de trouver les classes sans énumérer toutes les solutions optimales restait ouvert, alors un de nos résultats les plus importants est l'algorithme qui donne une réponse à ce problème, c'est-à-dire, un algorithme qui génère une séquence optimale par classe d'équivalence et qui donne aussi le nombre de séquences par classe, sans énumérer toutes les séquences [18]. Mais, bien que le nombre de classes soit beaucoup plus petit que le nombre de séquences, il peut être encore trop grand.

On propose alors l'utilisation de différentes contraintes biologiques, comme les intervalles communs (détectés initialement et progressivement), pour réduire le nombre de classes, et on montre comment utiliser ces méthodes pour analyser des cas réels d'évolution. En particulier, on analyse le scénario évolutif de la bactérie *Rickettsia* et des chromosomes sexuels X et Y chez l'être humain. Par rapport aux résultats des études précédentes, qui se sont basées sur une seule séquence optimale, on obtient une meilleure caractérisation de ces scénarios évolutifs.

Tous les algorithmes développés sont implémentés en java, integrés à BAOBABLUNA [16], un logiciel qui contient des outils pour manipuler des génomes et des inversions. Le téléchargement et le tutoriel de BAOBABLUNA sont disponibles en ligne.

C.2 PERMUTATIONS, INTERVALLES ET INVERSIONS

Les génomes étudiés sont représentés par la liste de marqueurs homologues (généralement des gènes ou des blocs des gènes contigus) qu'ils partagent. Ces marqueurs homologues sont représentés par les nombres entiers $1, 2, \dots, n$, avec un signe de plus ou de moins pour indiquer sur quel brin de l'ADN ils sont. L'ordre et l'orientation des marqueurs dans un génome est représenté par une *permutation signée* $\pi = (\pi_1, \pi_2, \dots, \pi_{n-1}, \pi_n)$ de taille n sur $\{-n, \dots, -1, 1, \dots, n\}$, de façon que, pour chaque valeur i de 1 a n, ou bien i ou bien $-i$ soit obligatoirement représenté, mais pas les deux. La *permutation identité* $(1, 2, 3, \dots, n)$ est dénoté par $\mathcal{I}_n$.

Un sous-ensemble d'entiers positifs $\rho \subseteq \{1, \dots, n\}$ est dit d'être un *intervalle* d'une permutation π si il existe $i, j \in \{1, \dots, n\}$, $1 \leq i \leq j \leq n$, de façon que $\rho = \{|\pi_i|, |\pi_{i+1}|, \dots, |\pi_{j-1}|, |\pi_j|\}$. Étant donnée une permutation π et un intervalle ρ de π, on peut appliquer une *inversion* sur l'intervalle ρ en π, c'est-à-dire, l'operation qui inverse l'ordre et les signes des éléments de ρ. Cette operation est dénotée par $\pi \circ \rho$. Si $\pi = (\pi_1, \pi_2, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_{j-1}, \pi_j, \pi_{j+1}, \dots, \pi_{n-1}, \pi_n)$ and $\rho = \{|\pi_i|, |\pi_{i+1}|, \dots, |\pi_{j-1}|, |\pi_j|\}$,

$$\pi \circ \rho = (\pi_1, \pi_2, \dots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \dots, -\pi_{i+1}, -\pi_i, \pi_{j+1}, \dots, \pi_{n-1}, \pi_n).$$

Par exemple, si on considère la permutation $\pi = (-3, 2, 1, -4)$ et l'intervalle $\rho = \{1, 2, 4\}$, on a $\pi \circ \rho = (-3, 4, -1, -2)$. Pour cette raison, un intervalle ρ peut aussi être utilisé pour dénoter une inversion.

Une séquence d'inversions $\rho_1\rho_2\ldots\rho_d$ est valide pour une permutation π si ρ_1 est un intervalle de π, ρ_2 est un intervalle de $\pi \circ \rho_1$, ρ_3 est un intervalle de $(\pi \circ \rho_1) \circ \rho_2$, et ainsi de suite. Si $\rho_1\rho_2\ldots\rho_d$ est une séquence valide d'inversions pour une permutation π, alors $\pi \circ \rho_1\rho_2\ldots\rho_d$ dénote l'application consécutive des inversions ρ_1, ρ_2, $\ldots\rho_d$ dans l'ordre qu'elles apparaissent. On dit qu'une séquence d'inversions $\rho_1\ldots\rho_d$ *trie* une permutation π dans une permutation π_T si $\pi \circ \rho_1\ldots\rho_d = \pi_T$. La longueur des plus courtes séquences d'inversions qui trient une permutation π dans π_T est appelée la *distance d'inversions* de π et π_T, dénotée par $d(\pi, \pi_T)$. De plus, les plus courtes séquences d'inversions qui trient π dans π_T sont appelées des *séquences optimales.* Par exemple, si la permutation π est $(-3, 2, 1, -4)$ et la permutation $\pi_T = \mathcal{I}_4$, on a $d(\pi, \pi_T) = 4$ et une séquence optimale est $\{1, 2, 4\}\{1, 3, 4\}\{2, 3, 4\}\{3\}$.

Dans le contexte de notre travail, sans perte de généralité, souvent on omet la permutation finale π_T. Dans ce cas, π_T correspond à la permutation identité $\mathcal{I}_n = (1, 2, 3, \ldots, n)$, ou n est la taille de la permutation initiale π, et la notation $d(\pi)$ est équivalente à $d(\pi, \mathcal{I}_n)$.

C.3 Le tri par inversions et son espace de solutions

Comme on a mentionné avant, étant donnée une permutation π, le problème de calculer $d(\pi)$ et celui de trouver une séquence optimale d'inversions pour trier π peuvent être solutionnés en temps polynomiel. L'approche classique pour analyser ces deux problèmes a été développée par Hannenhalli and Pevzner [8, 32, 33, 53] et s'est basée sur une structure appelée le *graph de points de cassure* (pour plus de détails, consulter [53]). Plusieurs études postérieures proposent d'autres algorithmes qui donnent aussi une séquence optimale pour trier un génome par inversions [4, 11, 26, 31, 63], mais le nombre de séquences optimales peut être très grand. Par exemple, pour la permutation $(-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$, le nombre de solutions différentes est 8278540, alors connaître une seule séquence de cet univers est insuffisant pour étudier le scénario évolutif des génomes correspondants.

Pour pouvoir réaliser une analyse plus précise, il faut considérer d'autres critères. Une stratégie possible est celle de chercher les séquences qui respectent certaines contraintes biologiques, comme par exemple les intervalles communs, qui sont les ensembles de gènes co-localisés dans les génomes analysés - une séquence d'inversions qui ne sépare pas les intervalles communs doit être plus réaliste qu'une séquence qui sépare [26]. Le problème est qu'on ne peut pas garantir qu'il existe une séquence qui respecte les contraintes choisies, alors on peut avoir un résultat nul.

Une autre approche, proposée par Siepel [59], est une méthode qui permet la construction de l'espace de toutes les solutions du problème du tri par inversions, c'est-à-dire, l'énumération de toutes les séquences optimales. Si on utilise cette méthode sur la permutation $\pi = (-3, 2, 1, -4)$, par exemple, on trouve 28 séquences optimales pour trier π, listées dans le Tableau 25).

Même si elle permet l'exploration de l'espace de toutes les séquences optimales, l'intérêt de cette approche est limité, parce que le nombre de séquences est en général tellement grand qu'on ne peut pas les calculer. Dans le Tableau 26 on peut voir, par des différentes exemples, comment l'augmentation de la distance d'inversion peut provoquer l'augmentation rapide du nombre de séquences optimales.

C.3.1 Traces

Bergeron *et al.* [9] ont observé que plusieurs séquences sont équivalentes et peuvent être regroupées dans des classes d'équivalence. D'une manière intuitive, toutes les séquences optimales

01.	$\{1\}\{1,2,3\}\{2\}\{4\}$	**11.**	$\{1,2,3\}\{4\}\{1\}\{2\}$	**21.**	$\{4\}\{1,2,3\}\{1\}\{2\}$
02.	$\{1\}\{1,2,3\}\{4\}\{2\}$	**12.**	$\{1,2,3\}\{4\}\{2\}\{1\}$	**22.**	$\{4\}\{1,2,3\}\{2\}\{1\}$
03.	$\{1\}\{2\}\{1,2,3\}\{4\}$	**13.**	$\{2\}\{1\}\{1,2,3\}\{4\}$	**23.**	$\{4\}\{2\}\{1\}\{1,2,3\}$
04.	$\{1\}\{2\}\{4\}\{1,2,3\}$	**14.**	$\{2\}\{1\}\{4\}\{1,2,3\}$	**24.**	$\{4\}\{2\}\{1,2,3\}\{1\}$
05.	$\{1\}\{4\}\{1,2,3\}\{2\}$	**15.**	$\{2\}\{1,2,3\}\{1\}\{4\}$	**25.**	$\{1,2,4\}\{1,3,4\}\{2,3,4\}\{3\}$
06.	$\{1\}\{4\}\{2\}\{1,2,3\}$	**16.**	$\{2\}\{1,2,3\}\{4\}\{1\}$	**26.**	$\{1,2,4\}\{1,3,4\}\{3\}\{2,3,4\}$
07.	$\{1,2,3\}\{1\}\{2\}\{4\}$	**17.**	$\{2\}\{4\}\{1\}\{1,2,3\}$	**27.**	$\{1,2,4\}\{3\}\{1,3,4\}\{2,3,4\}$
08.	$\{1,2,3\}\{1\}\{4\}\{2\}$	**18.**	$\{2\}\{4\}\{1,2,3\}\{1\}$	**28.**	$\{3\}\{1,2,4\}\{1,3,4\}\{2,3,4\}$
09.	$\{1,2,3\}\{2\}\{1\}\{4\}$	**19.**	$\{4\}\{1\}\{1,2,3\}\{2\}$		
10.	$\{1,2,3\}\{2\}\{4\}\{1\}$	**20.**	$\{4\}\{1\}\{2\}\{1,2,3\}$		

Table 25: Les 28 séquences d'inversion optimales qui trient la permutation $(-3,2,1,-4)$.

Permutation (π)	$\mathbf{N}_M$	$d(\pi)$	$\mathbf{N}_S$
$\pi_A = (-3,2,1,-4)$	4	4	28
$\pi_B = (-4,1,-3,6,-7,-5,2)$	7	6	204
$\pi_C = (-6,5,7,-1,-4,3,2)$	7	6	496
$\pi_D = (-4,-3,12,-11,-8,10,9,7,-6,-5,2,-1)$	12	8	31 752
$\pi_E = (-4,3,12,-11,-8,10,9,7,-6,-5,2,-1)$	12	9	407 232
$\pi_F = (-12,11,-10,6,13,-5,2,7,8,-9,3,4,1)$	13	10	8 278 540
$\pi_G = (-12,11,-10,-1,16,-4,-3,15,-14,9,-8,-7,-2,-13,5,-6)$	16	12	505 634 256
$\pi_H = (-12,11,-10,6,-5,13,2,7,8,-9,14,-15,3,4,-16,1)$	16	13	40 313 272 766

Table 26: Exemples de permutations, leurs tailles, leurs distances d'inversion et leurs nombre de séquences optimales.

dans la même classe sont composées par les mêmes inversions, mais apliquées dans un ordre différent. Le 28 séquences optimales qui trient la permutation $\pi_A = (-3,2,1,-4)$, par exemple, peuvent être regroupées dans deux classes d'équivalence, une avec 24 et l'autre avec 4 séquences (Tableau 27).

Classe 1:

01.	$\{1\}\{1,2,3\}\{2\}\{4\}$	**09.**	$\{1,2,3\}\{2\}\{1\}\{4\}$	**17.**	$\{2\}\{4\}\{1\}\{1,2,3\}$
02.	$\{1\}\{1,2,3\}\{4\}\{2\}$	**10.**	$\{1,2,3\}\{2\}\{4\}\{1\}$	**18.**	$\{2\}\{4\}\{1,2,3\}\{1\}$
03.	$\{1\}\{2\}\{1,2,3\}\{4\}$	**11.**	$\{1,2,3\}\{4\}\{1\}\{2\}$	**19.**	$\{4\}\{1\}\{1,2,3\}\{2\}$
04.	$\{1\}\{2\}\{4\}\{1,2,3\}$	**12.**	$\{1,2,3\}\{4\}\{2\}\{1\}$	**20.**	$\{4\}\{1\}\{2\}\{1,2,3\}$
05.	$\{1\}\{4\}\{1,2,3\}\{2\}$	**13.**	$\{2\}\{1\}\{1,2,3\}\{4\}$	**21.**	$\{4\}\{1,2,3\}\{1\}\{2\}$
06.	$\{1\}\{4\}\{2\}\{1,2,3\}$	**14.**	$\{2\}\{1\}\{4\}\{1,2,3\}$	**22.**	$\{4\}\{1,2,3\}\{2\}\{1\}$
07.	$\{1,2,3\}\{1\}\{2\}\{4\}$	**15.**	$\{2\}\{1,2,3\}\{1\}\{4\}$	**23.**	$\{4\}\{2\}\{1\}\{1,2,3\}$
08.	$\{1,2,3\}\{1\}\{4\}\{2\}$	**16.**	$\{2\}\{1,2,3\}\{4\}\{1\}$	**24.**	$\{4\}\{2\}\{1,2,3\}\{1\}$

Classe 2:

01.	$\{1,2,4\}\{1,3,4\}\{2,3,4\}\{3\}$	**03.**	$\{1,2,4\}\{3\}\{1,3,4\}\{2,3,4\}$
02.	$\{1,2,4\}\{1,3,4\}\{3\}\{2,3,4\}$	**04.**	$\{3\}\{1,2,4\}\{1,3,4\}\{2,3,4\}$

Table 27: Les deux classes d'équivalence qui contiennent les séquences optimales pour trier $\pi_A = (-3,2,1,-4)$.

Pour formaliser la relation d'équivalence des ces classes, on a besoin d'introduire le concept de *commutation*. On dit que deux intervalles (ou inversions) *se chevauchent* si ils (elles) s'intersectent mais aucun(e) ne contient l'autre. Par exemple, dans la permutation $(-3,2,1,-4)$, les intervalles $\{2,3\}$ et $\{1,2,4\}$ se chevauchent, mais $\{2,3\}$ et $\{1,2,3\}$ ne se chevauchent pas. Soient $s = \rho_1\rho_2\dots\rho_{i-1}\rho_i\rho_{i+1}\rho_{i+2}\dots\rho_d$ une séquence valide d'inversions pour une permutation π, et ρ_i et ρ_{i+1} deux inversions qui ne se chevauchent pas et qui apparaissent consécutivement en s. Comme ρ_i et ρ_{i+1} ne se chevauchent pas, alors ρ_{i+1} est un intervalle de $\pi \circ \rho_1\rho_2\dots\rho_{i-1}$ et ρ_i est un intervalle de $\pi \circ \rho_1\rho_2\dots\rho_{i-1}\rho_{i+1}$, c'est-à-dire, la séquence $s' = \rho_1\rho_2\dots\rho_{i-1}\rho_{i+1}\rho_i\rho_{i+2}\dots\rho_d$, obtenue par le remplacement de $\rho_i\rho_{i+1}$ par $\rho_{i+1}\rho_i$ dans s, est aussi une séquence valide d'inversions pour π. Cette operation d'inverser, dans une séquence, les positions de deux inversions consécutives ρ_i and ρ_{i+1} qui ne se chevauchent pas est appelée *commutation* de ρ_i et ρ_{i+1}.

Deux séquences sont dites d'être *équivalentes* si une peut être obtenue de l'autre par une séquence de commutations d'inversions qui ne se chevauchent pas. Une classe de séquences optimales équivalentes sous cette relation est appelée une *trace*. Il est facile de voir que toutes les séquences dans la même trace ont le même nombre d'inversions. Soit $s = \rho_1\rho_2 \dots \rho_i$ une séquence d'inversions, on dit que s est une $i-$*séquence* optimale pour une permutation π si $d(\pi \circ s, \pi_T) = d(\pi, \pi_T) - i$. On dénote alors par $i-$*trace* une trace de $i-$séquences.

Bergeron *et al.* [9] ont montré que l'ensemble de toutes les séquences optimales est une union de traces. Comme le nombre de traces est beaucoup plus petit que le nombre de séquences, représenter les séquences par les traces est une approche plus puissante qui l'énumération de toutes les séquences.

Forme normale d'une trace

Une séquence s dans une trace T est dite d'être dans la *forme normale* si elle peut être décomposée dans des sous-chaînes [17] $s = u_1 < \cdots < u_m$[18] de façon que:

- chaque deux inversions d'une sous-chaîne u_i ne se chevauchent pas;
- pour chaque inversion ρ d'une sous-chaîne u_i $(i > 1)$, il y a au moins une inversion θ de la sous-chaîne u_{i-1} de façon que ρ et θ se chevauchent;
- chaque sous-chaîne u_i est triée dans l'ordre lexicographique ascendant.

Un théorème de Cartier et Foata [23] (cité par [9]) énonce que, pour chaque trace, il y a un seule élément qui est dans la forme normale. En conséquence, on représente une trace par son élément dans la forme normale. Les deux traces qui contiennent les séquences optimales pour trier la permutation $\pi_A = (-3, 2, 1, -4)$, listées dans le Tableau 27, par exemple, peuvent être représentées par ses formes normales $\{1\}\{1, 2, 3\}\{2\}\{4\}$ et $\{1, 2, 4\}\{3\} < \{1, 3, 4\} < \{2, 3, 4\}$.

C.3.2 Un algorithme pour énumérer directement les traces

Bergeron *et al.* [9] n'ont pas donnés des pistes algorithmiques pour l'énumération directe des traces, sans énumérer toutes les séquences. De plus, l'énumération d'un élément de chaque trace sans énumérer toutes les séquences a été ennoncé comme un problème ouvert. Un de nos résultats les plus importants est une solution pour ce problème, c'est-à-dire, un algorithme qui énumère les formes normales de toutes les traces pour une permutation donnée, et qui compte le nombre de séquences dans chaque trace, sans énumérer toutes les séquences.

L'idée de cet algorithme est derivée de la notion suivante. Une k-trace T' est un *préfixe* d'une i-trace T $(k \leq i)$ si, et seulement si, chaque k-séquence de T' est un préfixe d'au moins une i-séquence de T. De plus, le nombre de séquences dans une $i-$trace est la somme des nombres de séquences dans ses $(i-1)-$préfixes (Figure 56).

On propose alors un algorithme qui construit toutes les i-traces simultanément d'une façon incrémental, sans générer toutes les séquences. Sans coût additionnel, l'algorithme calcule aussi le

[17] Les "sous-chaînes" sont tous les sous-ensembles contigus d'une séquence.

[18] Dans la notation original, la forme normale est $s = u_1| \dots |u_m$, mais on prefère utiliser le symbol '<' à la place de '|'.

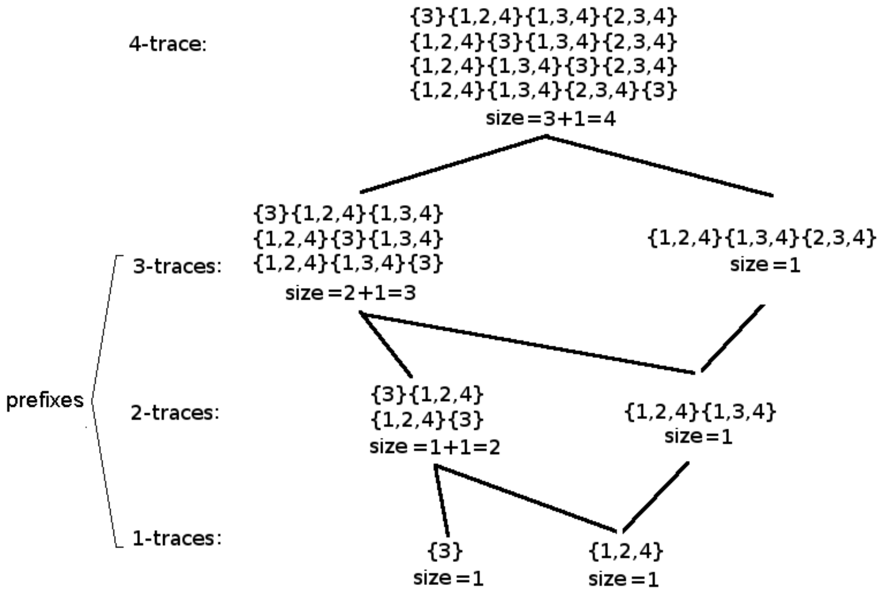


Figure 56: La decomposition d'une 4-trace dans ses préfixes.

nombre de séquences dans chaque i-trace. Étant donnée une permutation π, la méthode, illustrée dans la Figure 57, est la suivante. Dans chaque iteration i, pour chaque $(i-1)-$trace T construite dans l'iteration précédente on applique les inversions de la séquence de la forme normale de T sur π, pour obtenir une permutation intermédiaire π_T. On fait alors tourner l'algorithme de Siepel sur π_T et chaque une des inversions résultantes ρ est ajoutée á la $(i-1)-$trace T pour construir une nouvelle $i-$trace T_ρ, avec le même nombre d'éléments de T. Si il existe déjà une trace T' équivalente a T_ρ construite précédemment, on additionne le nombre d'éléments de T_ρ à T'. Sinon, on ajoute T_ρ à l'ensemble de $i-$traces déjà construites. La complexité de cet algorithme est $O(Nn^{k_{max}+4})$ [18], où k_{max} correspond à la largeur maximum d'une trace finale (la largeur d'une trace est donné par la taille du plus grand sous-ensemble d'inversions de la trace, dans lequel toutes les inversions commutent).

Implémentation et performance

L'implémentation de notre algorithme est intégrée au package BAOBABLUNA, qui est décrit dans la Section C.6. On a tourné plusieurs tests différents sur des permutations artificielles pour évaluer la performance de cet algorithme. Une partie des résultats est montrée dans le Tableau 28. Ces nombres nous donnent une bonne idée de la quantité d'information qui est manipulée, exprimée en nombre de séquences et de traces.

C.4 CONTRAINTES BIOLOGIQUES

Comme on a pu voir, bien que le nombre de classes soit beaucoup plus petit que le nombre de séquences, il peut être encore trop grand pour être interprété. De plus, pour les distances

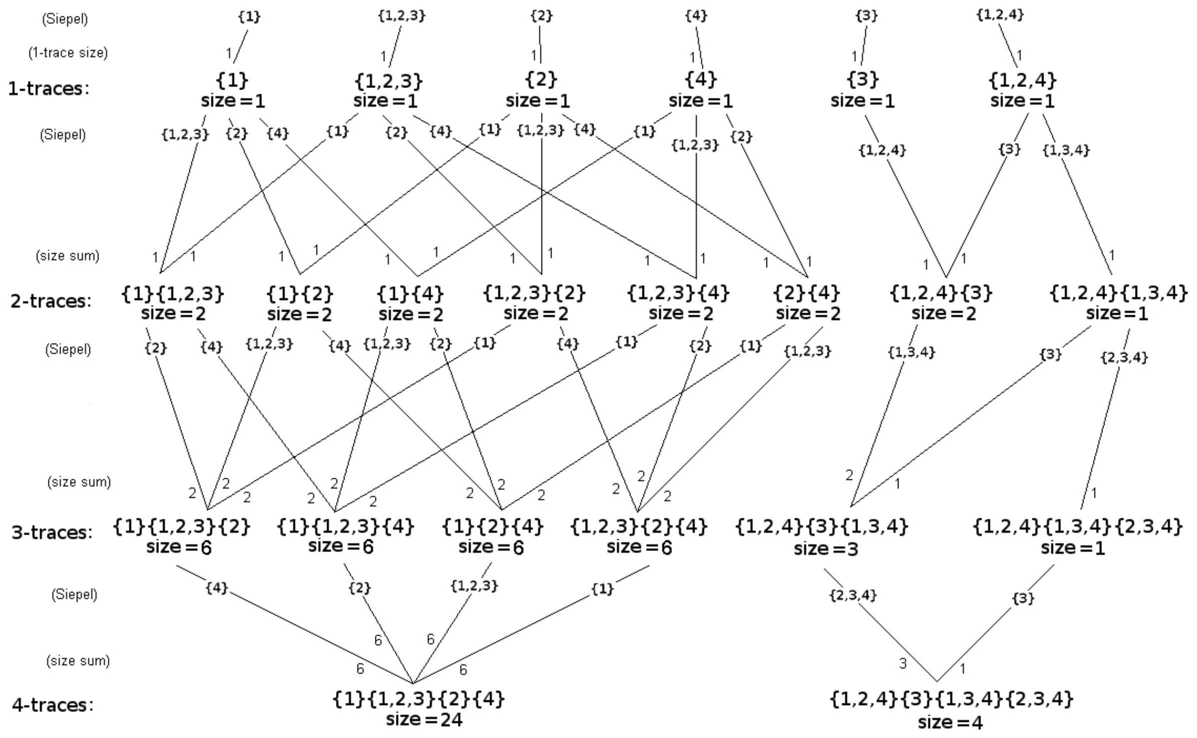


Figure 57: La construction de toutes les traces pour la permutation $(-3, 2, 1, -4)$.

PERMUT.	**N$_s$**	**N$_t$**	**Enum. sol.**	**Enum. + traces**	**Traces**
π_F n =12 d=10	8 278 540	2 151	$\simeq$ 13.5 min	$\simeq$ 30.1 min	$\simeq$ 27 sec
π_G n =16 d=12	505 634 256	21 902	$\simeq$ 16 h	$\simeq$ 43.5 h	$\simeq$ 7.25 min

Table 28: Résultats des expérimentations (1). Les colonnes de la gauche vers la droite contiennent: 1- la permutation, son nombre d'éléments et distance d'inversion; 2- le nombre de séquences; 3- le nombre de traces; 4- le temps d'exécution de l'algorithm qui énumère toutes les séquences; 5- le temps d'exécution de l'algorithm qui calcule toutes les traces par l'énumèration de toutes les séquences; 6- le temps d'exécution de l'algorithme qui énumère directement les traces. Les deux permutations analysées sont $\pi_F = (-12, 11, -10, 6, 13, -5, 2, 7, 8, -9, 3, 4, 1)$ *et* $\pi_G = (-12, 11, -10, -1, 16, -4, -3, 15, -14, 9, -8, -7, -2, -13, 5, -6)$. *Tous les algorithmes font partie du package* BAOBABLUNA.

d'inversion de $\simeq 20$ ou plus grandes, notre programme ne peut pas calculer les traces. Pour réduire le nombre de traces à générer et aussi pour attribuer une signification biologique aux traces, on a proposé l'énumération directe des traces qui respectent certaines contraintes.

À coté des deux permutations signées π et π_T, ce méthode requiert aussi une liste de q contraintes compatibles $C = (C_1, C_2, \ldots, C_q)$. On cherche les traces dont les séquences sont en accord avec les contraintes données. Toutefois, fréquemment seulement un sous-ensemble des séquences d'une trace sont en accord avec les contraintes en C et ce sous-ensemble est appelé sous-trace $C-$induite. La construction des traces reste inchangée, mais, en conséquence de la sélection des inversions, en fait on construit directement les sous-traces $C-$induites, dont les séquences sont en accord avec toutes les contraintes en C. Le résultat d'appliquer cette méthode, pour une liste C de contraintes et une permutation données, est l'ensemble des sous-traces C-induites non nulles, avec le nombre de séquences dans chaque sous-trace. Comme il n'y a pas de garantie de l'existence d'une trace non nulle, on peut occasionnellement avoir un résultat nul.

On a analysé qualitativement comment les contraintes peuvent affecter la chronologie des inversions et on a montré que certaines contraintes amènent à une approche symétrique (quand les sous-traces dont les séquences trient π dans π_T peuvent être obtenues des sous-traces dont les séquences trient π_T dans π) et d'autres amènent à une approche asymétrique. Pour chaque séquence d'inversions $s = \rho_1\rho_2 \dots \rho_{d-1}\rho_d$, on définie l'inverse de s comme $inv(s) = \rho_d\rho_{d-1} \dots \rho_2\rho_1$. De plus, étant donnée une trace T (respectivement, sous-trace t), on définie l'inverse de T (respectivement, l'inverse de t) comme $inv(T) = \{ \ inv(s) \mid s \in T \ \}$ (respectivement, $inv(t) = \{ \ inv(s) \mid s \in t \ \}$). Si s est une séquence d'inversions qui trie π dans π_T, alors $inv(s)$ est une séquence qui trie π_T dans π. Une liste C de contraintes est dite d'être ***symétrique*** quand la séquence s est en accord avec C à la condition que $inv(s)$ soit en accord avec C. Sinon, C est dite d'être ***asymétrique***. Si C est symétrique et t est une sous-trace $C-$induite pour trier π dans π_T, alors $inv(t)$ est une sous-trace $C-$induite pour trier π_T dans π.

On a considéré plusieurs contraintes biologiques différentes. Une de ces contraintes est la liste d'intervalles communs, qui sont les ensembles de gènes co-localisés dans les génomes analysés - une séquence d'inversions qui ne sépare pas les intervalles communs doit être plus réaliste qu'une séquence qui sépare [26]. Les intervalles communs entre deux permutations π et π_T sont les intervalles de π qui sont aussi intervalles de π_T.

On a utilisé les intervalles communs dans deux approches différentes. La première cherche les traces dont les séquences d'inversions ne cassent pas les intervalles communs détectés entre les deux permutations initialles, ce qu'on appele intervalles communs initialement détectés (les intervalles communs initialement détectés sont l'objet de plusieurs études sur le tri par inversions [6, 7, 10, 26]). Dans notre étude on a montré que cette approche est symétrique.

La deuxième approche, qui a été proposé par la première fois dans notre travail, cherche les sous-traces dont les séquences d'inversions ne cassent pas les intervalles communs détectés entre toutes les permutations intermédiaires, obtenues après l'application de 0 à $d-1$ inversions, et la permutation finale [17]. Cette approche est appellée détection progressive des intervalles communs et est asymétrique.

D'autres contraintes ont été definies pour étudier les problèmes pratiques auxquels on s'est intéressé. En particulier, on a pu caractériser les inversions par rapport au terminus de la réplication des chromosomes circulaires (cette méthode a été utilisée dans l'analyse de l'évolution de la bactérie *Rickettsia*), et appliquer une contrainte pour analyser directement le processus de stratification dans l'évolution des chromosomes sexuels humains X et Y [18, 41].

Les algorithmes qui considèrent les intervalles communs détectés initialement ou progressivement et celui qui considère la stratification des chromosomes XY ont la même complexité que l'algorithme original, qui énumère toutes les traces. En plus, plusieurs expérimentations ont montré que tous les variants qui prennent les contraintes biologiques en considération tournent plus rapidement que l'algorithme original.

L'analyse qualitative des contraintes a montré qu'une partie amène à des approches symétriques (quand le résultat de l'analyse des séquences qui trient le premier génome dans le deuxième peut être obtenu de l'analyse des séquences qui trient le deuxième génome dans le premier) et une partie amène à des approches asymétriques. On a argumenté que, quand la contrainte est asymétrique, elle peut être apliquée que quand le rapport ancêtre-descendant entre les deux génomes analysés est connu et, dans ce cas, l'analyse a une direction bien definie. Ci-dessous on résume les characteristiques de chacune des contraintes qu'on a considéré.

1. *Détection initial d'intervalles communs sans cassures*: symétrique, peut être appliqué à des chromosomes linéaires et circulaires.
2. *Détection initial d'intervalles communs avec un nombre limité de cassures*: asymétrique (analyse doit être faite du descendant vers l'ancêtre), peut être appliqué à des chromosomes linéaires et circulaires.
3. *Détection progressive d'intervalles communs*: asymétrique (analyse doit être faite du descendant vers l'ancêtre), peut être appliqué à des chromosomes linéaires et circulaires.
4. *Terminus-symétrie*: symétrique, peut être appliqué à des chromosomes circulaires uniquement.
5. *Strata chez les chromosomes sexuels XY*: asymétrique par concept (l'analyse doit être faite du chromosome X vers le Y), peut être appliqué à des chromosomes linéaires uniquement.

C.5 Applications

Avec l'énumération des traces qui respectent des contraintes biologiques, on a analysé le scénario évolutif de la bactérie *Rickettsia* et des chromosomes sexuels X et Y chez l'être humain. Par rapport aux résultats des études précédentes, qui se sont basées sur une seule séquence optimale [13, 55], on a obtenu une meilleure caractérisation de ces scénarios évolutifs.

C.5.1 Analyse des chromosomes sexuels humains X et Y

Dans l'analyse des chromosomes sexuels X et Y chez l'homme, Ross *et al.* [55] ont proposé une stratification particulière du chromosome X, soutenue par une seule séquence optimale d'inversions, obtenue grâce au logiciel GRIMM [64]. Néanmoins, les auteurs n'ont pas verifié l'existence d'autres séquences qui produisent la même stratification, ou des séquences qui produisent d'autres stratifications. Avec l'utilisation d'une stratification donnée comme contrainte pour filtrer toutes les traces de séquences optimales, on a pu vérifier la distribution des séquences optimales selon différentes stratifications du chromosome X. En particulier, on a pu vérifier que toutes les séquences qui génèrent la stratification proposé par Ross et al sont dans la même sous-trace, donc, si on suppose que cette stratification est précise, alors les inversions données par [55] ont été identifiées correctement.

Néanmoins, les permutations qui représentent les chromosomes X et Y, proposées par Ross *et al.* [55] couvrent seulement les premiers 11.2Mbps du X, et même pour cette partie réduite du chromosome il y en a des limites de stratification alternatives. De plus, si on étend les permutations pour couvrir une partie plus importante des chromosomes, le nombre de possibilités pour placer les limites des strates augmente. En effet, seulement la limite entre la region pseudo-autosomal (PAR) et la strate 5 et la limite entre les strates 5 et 4 sont bien établies. Les limites des autres strates sont encore controversées et même le nombre exacte de strates est discuté. On a participé dans un travail collaboratif avec Lemaitre *et al.* [41], qui a présenté des arguments supplémentaires pour expliquer le processus de stratification. En particulier, notre méthode pour chercher des sous-traces qui respectent la formation des strates a été utilisée pour analyser les permutations étendues et vérifier différentes hypothèses pour le placement de la limite entre les strates 4 et 3.

C.5.2 Analyse des bactéries *Rickettsia*

Le scénario évolutif de six espèces de la bactérie *Rickettsia* a été reconstruit par Blanc *et al.* [13] et, en particulier, une séquence arbitraire a été proposé pour trier l'ancêtre $R2$ dans *Rickettsia felis*. Cette séquence est composé par cinq inversions externes et quatre inversions symétriques au terminus de la réplication. On a pu analyser l'univers de toutes les séquences qui trient ces génomes avec notre méthode. La recherche de séquences avec un maximum de 3 inversions externes et aucune inversion asymétrique, combinée avec la détection progressive d'intervalles communs (affaiblie pour permettre deux cassures d'intervalles) a trouvé trois sous-traces dont les séquences ont six inversions symétriques, ce qui donne une meilleure characterisation du scénario évolutif de ces génomes, par rapport au travail précédent de Blanc *et al.* [13].

C.6 BAOBABLUNA

Tous les algorithmes qu'on a développés sont implémentés en java, intégrés à BAOBABLUNA [16], un logiciel qui contient des outils pour manipuler des génomes et des inversions. Le téléchargement et le tutoriel de BAOBABLUNA sont disponibles en ligne à l'adresse `http://pbil.univ-lyon1.fr/software/luna/`

Le logiciel BAOBABLUNA contient une structure qui est capable de compresser efficacement les traces pour les garder dans un ensemble trié. La comparaison de la performance de cette structure par rapport à une implementation standard d'un ensemble trié a montré qu'on a pu économiser beaucoup de mémoire sans perdre en temps d'exécution.

C.7 CONCLUSIONS ET LIMITATIONS

Un de résultats les plus importants de notre travail est un algorithme qui génère direcetement les traces de séquences d'inversions optimales qui trient un génome dans un autre et qui donne aussi le nombre de séquences par trace, sans énumérer toutes les séquences [18]. L'algorithme qui génère directement les traces représente une amélioration importante par rapport à l'énumeration de toutes les séquences optimales. Néanmoins, pour les permutations dont la distance d'inversion est supérieure à une certaine valeur, l'univers de traces est encore trop grand pour être interprété et fréquemment il ne peut même pas être calculé, malgré l'optimization de l'utilisation de la mémoire qu'on a implementé dans BAOBABLUNA. En effet, on n'est pas encore capable de calculer les traces pour les permutations dont les distances d'inversion sont égales ou supérieures à 20.

On a proposé alors l'utilisation de différentes contraintes biologiques pour réduire la taille de l'ensemble à traiter. Différentes contraintes ont été proposées, comme les intervalles communs (détectés initialement et progressivement), la stratification des chromosomes sexuels X et Y et la symétrie des inversions par rapport au terminus de la réplication chez les bactéries. On a utilisé ces méthodes pour analyser des cas réels d'évolution. En particulier, on a analysé le scénario évolutif de la bactérie *Rickettsia* et des chromosomes sexuels X et Y chez l'être humain. Par rapport aux résultats des études précédentes, qui se sont basées sur une seule séquence optimale, on obtient une meilleure caractérisation de ces scénarios évolutifs.

L'utilisation de contraintes biologiques est alors une bonne stratégie pour réduire l'ensemble à traiter par la sélection de séquences d'inversions qui sont biologiquement plus significatives. Toutefois, il n'y a pas de garantie de l'existence d'une séquence qui respecte les contraintes données, alors cette approche peut amener à un résultat nul, ce qui n'est pas désirable. Affaiblir les contraintes pour pouvoir obtenir un résultat non-nul est généralement possible, mais cette procédure peut demander plusieurs essais des paramètres d'affaiblissement à un coût de calcul important.

La symétrie par rapport au terminus de la réplication, qui est une des contraintes le plus prometteuses, a aussi des limitations. Le choix de valuers pour ses paramètres n'est pas trivial et demande plusieurs essais. De plus, on adopte certaines simplifications, comme ignorer le space entre les marqueurs et le déplacement du terminus de la réplication par chaque inversion asymétrique. Il faut considérer ces simplications quand on interprète le résultat des analyses.

Tous les algorithmes développés sont implémentés en java, integrés à BAOBABLUNA [16], un logiciel qui contient des outils pour manipuler des génomes et des inversions. Le téléchargement et le tutoriel de BAOBABLUNA sont disponibles en ligne.

C.8 PERSPECTIVES

En ce qui concerne le programme pour analyser les traces, d'autres contraintes peuvent être introduites. La difficulté ici est de mieux connaître le processus évolutif d'organismes relativement proches et d'isoler les proprietés qui peuvent être utilisées pour filtrer les inversions à chaque pas de la construction de traces.

Trouver d'autres applications réels pour utiliser la méthode avec les contraintes qui sont déjà implementés est aussi un objectif à poursuivre. En particulier, on croit que la détection progressive d'intervalles communs et les inversions symétriques par rapport au terminus de la réplication chez les chromosomes circulaires peuvent révéler des détails intéressants sur le processus de réarrangement de ces organismes. Comme on peut traiter que des génomes relativement proches, la difficulté est de trouver des données adéquates. Un bon candidat est la bactérie *Mycbacterium*, dont l'ancêtre a été reconstruit récemment [30].

De plus, la méthode qui cherche les inversions symétriques peut être ameliorée par l'application d'une limitation sur le déplacement du terminus de la réplication. L'avantage de cette approche est que, à la suite d'une inversion qui déplace le terminus, elle favorise la sélection des inversions qui compensent le déplacement précédent, alors l'ordre des inversions peut être plus fortement restreint.

La méthode qui cherche les sous-traces qui respectent la formation des strates chez les chromosomes sexuels peut être utilisée pour analyser d'autres espèces. On a très peu de données disponibles pour l'instant, mais on espère qu'on aura de plus en plus de séquences de chromosomes sexuels de différentes espèces dans le futur proche.

Appendix D

Article submitted to *Genome Biology and Evolution*

We developed the study *Footprints of inversions at present and past pseudoautosomal boundaries in human sex chromosomes* in collaboration with Lemaitre *et al.* [41]. This work concerns the evolution of the human sexual chromosomes X and Y and was recently submitted to the journal *Genome Biology and Evolution*. The submitted version is attached after the references.

References

[1] Adi, S., Braga, M. D. V., Fernandes, C., Ferreira, C., Martinez, F., Sagot, M.-F., Stefanes, M., Tjandraatmadja, C. and Wakabayashi, Y., "Repetition-free longest common subsequence", *Discrete Applied Mathematics*, submitted, 2009 (a preliminary version appeared in Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS), *Eletronic Notes in Discrete Mathematics*, Vol. 30, Pages 243–248, 2008).

[2] Andersson S. G. E. and Kurland C. G., "Reductive evolution of resident genomes", *Trends in Microbiology*, Vol. 6, Number 7, 263–268, 1998.

[3] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M., *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer, 1999.

[4] Bader, D. A., Moret, B. M. E. and Yan, M., "A linear-time algorithm for computing inversion distances between signed permutations with an experimental study", *J. Comput. Biol.* 8, 5 (2001), 483–491.

[5] Vineet Bafna, V., Pevzner, P. A., "Sorting by Transpositions", *SIAM Journal on Discrete Mathematics*, vol. 11, issue 2, 224–240, 1998.

[6] Berard S., Bergeron A. and Chauve C., "Conserved structures in evolution scenarios", RCG 2004, *Lecture Notes in Bioinformatics*, vol. 3388, 1–15, 2005.

[7] Berard S., Bergeron A., Chauve C. and Paul C., "Perfect sorting by reversals is not always difficult", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, Vol. 4, No. 1, 4–16, 2007.

[8] Bergeron A., "A very elementary presentation of the Hannenhalli-Pevzner theory", *Discrete Applied Mathematics*, vol. 146, 134–145, 2005.

[9] Bergeron A., Chauve C., Hartmann T. and St-Onge K., "On the properties of sequences of reversals that sort a signed permutation", *JOBIM 2002*, 99–108, 2002.

[10] Bergeron A., Heber S. and Stoye J., "Common intervals and sorting by reversals: a marriage of necessity", *Bioinformatics*, 18 (Suppl. 2): S54–63, 2002.

[11] Bergeron A., Mixtacki J. and Stoye J., "The inversion distance problem", *Mathematics of evolution and phylogeny* (O. Gascuel Ed.) Oxford University Press, 2005.

[12] Bergroth, L., Hakonen, H. and Raita T., "A Survey of Longest Common Subsequence Algorithms", SPIRE 00: pages 39-48, 2000.

[13] Blanc G., Ogata H., Robert C., Audic S., Suhre K., Vestris G., Claverie J.-M. and Raoult D., "Reductive genome evolution from the mother of Rickettsia", *PLoS Genetics*, volume 3, p. 103-114, 2007.

[14] Blin, G., Fertin, G. and Chauve, C., "The breakpoint distance for signed sequences", *Texts in Algorithms*, vol. 3, pages 3-16, CompBioNets 2004.

[15] Bonizzoni P., Della Vedova G., Dondi R., Fertin G. and Vialette S., "Exemplar Longest Common Subsequence", *ACM/IEEE Trans. Computational Biology and Bioinformatics*, Vol. 4, No. 4, pages 535–543, 2007.

[16] Braga M. D. V., "`baobabLuna`: the solution space of sorting by reversals", submitted to

Bioinformatics, 2009.

[17] Braga M. D. V., Gautier C. and Sagot M.-F., "An asymmetric approach to preserve common intervals while sorting by reversals", submitted to *Algorithms for Molecular Biology*, 2009.

[18] Braga M. D. V., Sagot M.-F., Scornavacca C. and Tannier E., "Exploring the solution space of sorting by reversals with experiments and an application to evolution", *Transactions on Computational Biology and Bioinformatics*, volume 5, number 3, 348–356, 2008 (A preliminary version appeared in ISBRA 2007, *Lecture Notes in Bioinformatics* vol. 4463, 293–304).

[19] Brightwell G. and Winkler P., "Counting linear extensions is #P-complete", *STOC '91: Proceedings of the twenty-third annual ACM Symposium on Theory of Computing*, ACM Press, 1991.

[20] Bryant, D., "The complexity of calculating exemplar distances", 2000.

[21] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley and Sons Ltd, Chichester, UK, 1996.

[22] Caprara, A., "Sorting by reversals is difficult", *RECOMB*, 75–83, 1997.

[23] Cartier, P. and Foata D., "Problèmes combinatoires de commutations et réarrangements", *Lecture Notes in Math*, vol. 85, Springer, Berlin, 1969.

[24] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., *Introduction to algorithms*, MIT Press, 2. edition, 2001.

[25] Diekert V. and Rozenberg G. (eds), *The book of traces*, World Scientific, 1995.

[26] Diekmann Y., Sagot M.F. and Tannier E., "Evolution under reversals: parsimony and conservation of common intervals", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, No. 2, 301–309 (A preliminary version appeared in COCOON 2005, *Lecture Notes in Computer Science*, vol. 3595, 42–51).

[27] Eisen J. A., Heidelberg J. F., White O. and Salzberg S. L., "Evidence for symmetric chromosomal inversions around the replication origin in bacteria", *Genome Biology*, vol. 1, No. 6, 2000.

[28] Fulkerson D. R., "Note on Dilworth's decomposition theorem for partially ordered sets", Proc. Amer. Math. Soc. 7, 701–702, 1956.

[29] Goffeau A. *et al.*, "Life with 6000 genes", Science 274. doi:10.1126/science.274.5287.546, 1996.

[30] Gomez-Valero, L., Rocha, E. P. C., Latorre, A. and Silva, F. J., "Reconstructing the ancestor of *Mycobacyerium leprae*: the dynamics of gene loss and genome reduction", *Genome Research*, vol. 17, 1178–1185, 2007.

[31] Han Y., "Improving the Efficiency of Sorting by Reversals", Proceedings of The 2006 International Conference on Bioinformatics and Computational Biology, CSREA Press, Las Vegas, Nevada, USA, 2006.

[32] Hannenhalli S. and Pevzner P., "Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)", *Journal of the ACM*, 46:1–27, 1999.

[33] Hannenhalli, S. and Pevzner, P., "Transforming men into mice (polynomial algorithm for genomic distance problem)", In Proceedings of the IEEE 36th Annual Symposium on Foun-

dations of Computer Science, pages 581-592, 1995.

[34] Heber, S. and Stoye, J., "Finding all common intervals of k permutations", in Combinatorial Pattern Matching, 12th Annual Symposium, *Lecture Notes in Computer Science*, vol. 2089, 207–218, 2001.

[35] IJdo J. W., Baldini A., Ward D. C., Reeders S. T., Wells R. A., "Origin of human chromosome 2: an ancestral telomere-telomere fusion", Proc Natl Acad Sci U S A, 88(20):9051-5, 1991.

[36] Iwase, M., Satta, Y., Hirai, Y., Hirai, H., Imai, H., and Takahata, N., "The amelogenin loci span an ancient pseudoautosomal boundary in diverse mammalian species", *PNAS*, vol. 100, no. 9, 5258–5263, 2003.

[37] Kaplan, H., Shamir, R., and Tarjan, R. E., "Faster and simpler algorithm for sorting signed permutations by reversals", *SIAM J. Comput*, vol. 29, 880-892, 1999.

[38] Kececioglu, J. and Sankoff, D., "Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement", *Algorithmica*, 13:1/2, 180–210, 1995.

[39] Knuth, D., "The Art of Computer Programming, Volume 3: Sorting and Searching", Second Edition, Addison-Wesley, 1998.

[40] Lahn B. T. and Page D. C., "Four evolutionary strata on the human X chromosome", *Science*, vol. 286, 964–967, 1999.

[41] Lemaitre C., Braga M. D. V., Gautier C., Sagot M.-F., Tannier E. and Marais G. A. B., "Footprints of inversions at present and past pseudoautosomal boundaries in human sex chromosomes", submitted to *Genome Biology and Evolution*, 2009[19].

[42] Mackiewicz, P., Mackiewicz, D., Kowalczuk, M. and Cebrat S.,"Flip-flop around the origin and terminus of replication in prokaryotic genomes", *Genome Biology*, vol. 2, No. 12, 2001.

[43] Mazowita, M., Haque, L. and Sankoff, D., "Stability of rearrangement measures in the comparison of genome sequences", *Journal of Computational Biology*, vol. 13, 554–566, 2006.

[44] McLysaght, A., Seoighe, C. and Wolfe K. H., "High frequency of inversions during eukaryote gene order evolution", in *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families*, D. Sankoff and J. H. Nadeau (Eds.), 47–55, 2000.

[45] Moret, B.M.E., Wyman, S., Bader, D.A., Warnow, T., and Yan, M., "A new implementation and detailed study of breakpoint analysis", *Proc. 6th Pacific Symp. on Biocomputing* (PSB 2001), Hawaii, World Scientific Pub., 583–594, 2001.

[46] Nakabachi A., Yamashita A., Toh H., Ishikawa H., Dunbar H., Moran N., Hattori M., "The 160-kilobase genome of the bacterial endosymbiont Carsonella", *Science*, 314 (5797): 267, 2006.

[47] Ogata H., Renesto P., Audic S., Robert C., Blanc G., Fournier P.-E., Parinello H., Claverie J.-M. and Raoult D., "Genome sequence of Rickettsia felis identifies the first putative conjugative plasmid in an obligate intracellular parasite", *PLoS Biology*, volume 3, p. 1-12, 2005.

[48] Ogata H., La Scola B., Audic S., Renesto P., Blanc G., Robert C., Fournier P.-E., Claverie J.-M. and Raoult D., "Genome sequence of Rickettsia bellii illuminates the role of amoebae in

[19]The work of Lemaitre *et al.* is attached at the end of this manuscript

gene exchange between intracellular pathogens", *PLoS Genetics*, volume 2, p. 733-744, 2006.

[49] Ohno, S., *Evolution by gene duplication*, Springer-Verlag, New York, 1970.

[50] Ohno, S., *Sex chromosomes and sex-linked genes*, Springer, Berlin, 1967.

[51] Papadimitriou, C. H. and Yannakakis, M., "Optimization, approximation and complexity classes", *Journal of Computer and System Sciences*, 43:425–440, 1991.

[52] Parfrey, L. W., Lahr, D. J. G., Katz, L. A., "The Dynamic Nature of Eukaryotic Genomes", *Molecular Biology and Evolution*, 25 (4): 787, 2008.

[53] Pevzner P., *Computational Molecular Biology - An Algorithmic Approach*, The MIT Press, 2000.

[54] Pradella S., Hans A., Spröer C., Reichenbach H., Gerth K., Beyer S., "Characterisation, genome size and genetic manipulation of the myxobacterium Sorangium cellulosum So ce56". *Arch Microbiol*, 178 (6): 484-92, 2002.

[55] Ross M. T. *et al.*, "The DNA sequence of the human X chromosome", *Nature*, vol. 434, p. 325–337, 2005.

[56] Sankoff, D., "Gene and genome duplication", *Current Opinion in Genetics and Development*, vol. 11, p. 681–684, 2001.

[57] Sankoff, D., "Genome Rearrangement with gene families", *Bioinformatics*, vol. 15, no. 11, pages 909-917, 1999.

[58] Schneiker S. *et al.*, "Complete genome sequence of the myxobacterium Sorangium cellulosum", *Nature Biotechnology*, 25, 1281–1289, 2007.

[59] Siepel A., "An algorithm to enumerate sorting reversals for signed permutations", *J Comput Biol*, 10:575–597, 2003.

[60] Skaletsky H. *et al.*, "The male-specific region of the human Y chromosome is a mosaic of discrete sequence classes", *Nature*, vol. 423, 825–837, 2003.

[61] Steiner G., "An algorithm to generate the ideals of a partial order", *Operations Research Letters*, 5(6):317–320, 1986.

[62] Steiner G., "Polynomial algorithms to count linear extensions in certain posets", *Congressus Numerantium*, 75, 71–90, 1990

[63] Tannier E., Bergeron A. and Sagot M.-F., "Advances on Sorting by Reversals", *Discrete Applied Mathematics*, vol. 155, no. 6-7, 881–888, 2007 (a preliminary version appeared in CPM 2004, *Lecture Notes in Computer Science*, vol. 3595, 42–51).

[64] Tesler, G., "GRIMM: genome rearrangements web server", *Bioinformatics*, vol. 18, no. 3, 492–493, 2002.

[65] Weaver R. F., *Molecular Biology*, Mc Graw Hill, second edition, 2002.

[66] Zheng, C., Lenert, A. and Sankoff, D., "Reversal distance for partially ordered genomes", Bioinformatics 21, i502-i508, 2005.

Research Article

Footprints of inversions at present and past pseudoautosomal boundaries in human sex chromosomes

Claire Lemaitre*[1], Marilia DV Braga* [1], Christian Gautier[1], Marie-France Sagot[1], Eric Tannier¶[1] and Gabriel AB Marais¶[1]

[1] *Université de Lyon; Université Lyon 1; Centre National de la Recherche Scientifique; Institut National de Recherche en Informatique et en Automatique; UMR5558; Laboratoire de Biométrie et Biologie évolutive; Villeurbanne, F-69622 cedex, France.*

* Equal contribution

¶ Corresponding authors:

- G. Marais - marais@biomserv.univ-lyon1.fr - tel: +33 4 72 43 29 09, fax: +33 4 72 43 13 88
- E. Tannier – Eric.Tannier@inria.fr - tel: +33 4 26 23 44 74, fax: +33 4 72 43 13 88

Running head: Inversions and evolution of sex chromosomes

Keywords: inversion, duplication, recombination, sex chromosomes, evolutionary strata

List of nonstandard abbreviations:
PAR = pseudoautosomal region, XAR = X-added region, NRY = non-recombining Y, d_S = rate of synonymous substitutions, d_N = rate of non-synonymous substitutions, NHEJ = non-homologous end joining

Abstract

Inversions are known to occur repeatedly in genome evolution but their evolutionary significance remains obscure. It has been suggested that five large inversions might have affected the human Y chromosome and reduced the pseudoautosomal region (PAR) in five steps. This could explain the five evolutionary strata (chromosomal domains with different levels of X-Y divergence) that were observed in the human X chromosome. Clear evidence for these inversions is however missing. Here we looked for such evidence by focusing on a region (the X Added Region - called XAR - that includes the PAR and the most recent strata 3, 4 and 5) that has conserved gene order between the human X and its chicken homolog. This means that detected rearrangements between X and Y in this region should have occurred in Y. We first estimated and analysed the whole set of parsimonious scenarios of Y inversions given the gene orders in XAR and in its Y homolog. Comparing these scenarios to scenarios for simulated sequences (with and without formation of strata) suggests that the most recent strata 4 and 5 in humans have been formed by inversions on the Y chromosome. By comparing DNA sequences from X and Y, we then found clear evidence of two Y inversions associated with duplications that coincide with the boundaries of strata 4 and 5. Divergence between the duplicates is in agreement with the timing of strata 4 and 5 formation. These duplicates show a complex pattern of gene conversion that resembles the pattern previously found for *AMELX* and *AMELY*, a locus of stratum 3. This suggests that the *AMEL* locus - despite *AMELY* being unbroken- was possibly involved in a Y inversion that formed stratum 3. However, no clear evidence supporting the formation of stratum 3 by a Y inversion was found probably because this stratum is too old for such inversions to be detectable. Our results on XAR strongly support the view that in humans, the most recent strata have arisen by inversions on the Y and suggest that inversions have played a major role in the differentiation of our sex chromosomes.

350 words

Introduction

Chromosomal inversions are very common in animal, fungal and plant genomes (Murphy *et al.* 2005, Yogeeswaran *et al.* 2005, Fischer *et al.* 2006, Ranz *et al.* 2007, Bhutkar *et al.* 2008). However, the forces that establish inversions and their evolutionary significance remain poorly understood. An important characteristic of inversions is that recombination is suppressed at the inverted regions in chromosomal heterozygotes (Navarro *et al.* 1997, Andolfatto *et al.* 2001). This makes inversions particularly prone to accumulating mutations involved in local adaptation. Based on this ground, it has been suggested that inversions could have a significant role in speciation (Kirkpatrick & Barton 2006). Another possible case of the evolutionary importance of inversions is the evolution of sex determination and sex chromosomes.

Well-differentiated sex chromosomes such as the human XY chromosomes do not recombine except in small regions (called pseudoautosomal regions, PARs). Theory predicts that recombination between newly formed sex chromosomes should be suppressed at male determining genes so that they are genetically linked to the Y and no neutral or hermaphroditic recombinants are formed (Nei 1969, Charlesworth & Charlesworth 1978). It is also predicted that later in the evolution of the sex chromosomes, the accumulation of antagonistic genes (beneficial for males and deleterious for females) should gradually suppress recombination between X and Y making the Y chromosome a fully or almost fully non-recombining chromosome (Charlesworth *et al.* 2005).

In a pioneer work, Lahn & Page (1999) found that synonymous divergence between X-Y homologous gene pairs correlated with gene position on the X chromosome in a stair-like shape. They took this as evidence that human XY were originally recombining autosomes that gradually stopped recombining, forming ''evolutionary strata'' (i.e. groups of genes which, because they do not recombine anymore, start diverging at the same time). Since gene order for those homologous gene pairs was found to be completely different on the X and on the Y chromosomes, they suggested that large Y inversions might have caused the evolutionary strata. The *XG* gene that spans the current pseudoautosomal boundary on the human X but is truncated on the human Y fits well with this idea.

Iwase and colleagues (2003) looked at Amelogenine, a gene that they believed to be located on an ancient pseudoautosomal boundary (strata 3/4) because the X-Y divergence drops from 30% to 10% within this gene however this gene was thought to be not involved in an Y inversion since its Y copy (*AMELY*) is not truncated. The sequencing of the euchromatic part of the human Y made the picture even more blurred instead of clarifying it. Skaletsky *et al.* (2003) re-analysed XY gene pairs as in Lahn & Page (1999) with more data but they did not find four well defined strata as in Lahn & Page (1999). The limits between strata seemed to overlap, especially those of the most recent strata (3 and 4). Following this work, doubts were raised about the Y inversions model put forward by Lahn & Page (Charlesworth *et al.* 2005). Chromosomal rearrangements are known to accumulate at a faster rate in regions of reduced recombination. The rearrangements between X and Y could well have post-dated the formation of the strata (and not pre-dated it as expected in the Lahn & Page model).

In the paper reporting the complete sequencing of the human X, Amelogenine was dismissed as evidence against the Y inversion model (Ross *et al.* 2005). *AMELX* and *AMELY* can do gene conversion, which could explain the Amelogenine peculiar X-Y divergence pattern (Marais & Galtier 2003, Ross *et al.* 2005). Using the GRIMM software, Ross and colleagues reported the first attempt to reconstruct the X-Y chromosomal rearrangements and found a scenario consistent with strata 4 and 5, a new stratum that they defined. Inferring a scenario of inversions is known to be a very challenging task when analysing a sequence with relatively few markers and many inversions as in the case of XY chromosomes. GRIMM uses an algorithm known to infer efficiently the minimum number of inversions from one sequence to another. However, GRIMM does not include any framework to find and analyse all the optimal scenarios. It just gives one arbitrarily drawn optimal scenario, which considerably weakens the conclusions of Ross *et al.* (2005). Thus, evidence for the model of Y inversions in humans remains dubious.

The gradual loss of recombination that formed the strata had a profound impact on the human Y degeneration. The strata have different levels of degeneration (the most recent ones being the least degenerate) and the level of dosage compensation, which is known to be a response to Y degeneration, is correlated with the strata (the most recent ones having less genes showing dosage compensation) (Carrel & Willard 2005). The gradual loss of recombination between our sex chromosomes has enhanced some processes of degeneration (genetic hitchhiking) over others (Muller's ratchet) compared to what would have happen if recombination had been stopped once (Bachtrog 2008). Evolutionary strata are not a bizarre feature of our sex chromosomes. They have been found in other organisms such as rodents (Sandstedt & Tucker 2004), birds (Handley *et al.* 2004, Nam & Ellegren 2008) and plants (Nicolas *et al.* 2005, Bergero *et al.* 2007) and seem to be a general phenomenon in heteromorphic sex chromosomes. Despite the importance of the strata for the biology and evolution of the sex chromosomes in general, we still know very little on how they are formed.

Here our goal was to test whether the reconstruction of the human X-Y chromosomal rearrangements fits with the currently defined evolutionary strata in our species. We focused on XAR (=X-Added Region, it is located on the X p-arm and comprises PAR and strata 5, 4 and 3 that show about 5, 10 and 30% of X-Y divergence respectively) because gene order is conserved between human and chicken (human XAR matches with chicken chromosome 1q with almost no rearrangements, see Ross *et al.* 2005). Detected rearrangements between X and Y in that region should have occurred in Y. We first used the same 12 markers as in Ross *et al*. (2005) and evaluated the number of possible scenarios of Y inversions given the gene orders in XAR and in its Y homolog using a method that we developed (Braga *et al.* 2008). We found that there are many scenarios in which Y inversions coincide with strata boundaries. Using simulations with randomly distributed inversions on Y, we found that it is unlikely that this pattern has emerged just by chance. Another set of simulations with Y inversions occurring only among strata indicates that recent strata have arisen by inversions on the Y.

If Y inversions have formed the strata, by finding and analysing the Y regions homologous to the strata boundaries on the X, we should be able to find footprints of those inversions. To do that we used a method designed to detect and analyse genomic regions with breakpoints (Lemaitre *et al.* 2008) and we found clear evidence of inversions associated with duplications at the PAR/stratum 5 and the strata 5/4 boundaries. Analysis of the divergence between duplicates for both boundaries allowed us to date the inversions and is in agreement with stratum 5 being more recent than stratum 4. This strongly suggests that in humans, recent strata have arisen by inversions on the Y and give support to the idea Lahn & Page first put forward that recombination between X and Y stopped because of inversions on the Y. A pair of duplicates associated with an inversion shows evidence for gene conversion and this suggests that gene conversion in *AMEL* is in fact consistent with *AMEL* spanning the strata 3/4 boundary. However, we could not find either clear footprints of inversions for stratum 3 or any optimal scenarios consistent with the formation of stratum 3 by a single Y inversion (including all the available markers for this stratum and not just two as in Ross *et al.* 2005) but this may simply be because stratum 3 is too old,footprints of inversions have been erased and X-Y are too rearranged. We discuss the case of ancient human strata.

Materials and Methods

Identification of new markers

Already known markers are from Skaletsky *et al.* (2003) and Ross *et al.* (2005). New markers were identified from an alignment of the human X-Y chromosomes (from NCBI version 36, hg18) using BlastZ (Schwartz *et al.* 2003). Local similarities found with BlastZ were concatenated if they had the same order and orientation and if they were less than 30 Kb apart on both chromosomes. Concatenates smaller than 30 Kb and those located in known ampliconic regions and in the peri-centromeric regions were discarded. We thus obtained three new markers (see Table 1).

Analysis of optimal scenarios for X-Y rearrangements

To analyse the formation of strata 4 and 5, we ran the Braga *et al.* (2008) program on the following sequences: X =(1,2,3,4,5,6,7,8,9,10,11,12) and Y =(-12,11,-2,-1,-10,-9,8,-5,7,6,-4,3) with numbers corresponding to markers in Ross *et al.* (2005) and minus indicating a change in orientation. To analyse the formation of stratum 3, we ran the same program on the following sequences: X =(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23) and Y = (19,14,-15,12,-2,-1,13,22,20,-23,16,-11,-10,9,-6,8,7,-5,4,-3,21,17,-18). We also did the analysis without marker 23 (see Discussion).

Simulation of inversions

Simulations were conducted to create a sequence Y from a sequence X with a given number of inversions *d*. A simulated Y sequence was made in *d* steps. At step *i*, two positions were selected at random, an inversion between these two positions was produced. The inversion was accepted only if going back from Y_i (Y after *i* inversions) to Y_0 (Y without inversion = X) implied *i* inversions (parsimony criterion). We stopped the process after *d* parsimonious inversions. We repeated this 1000 times and obtained the free inversions simulations. The set of strata-constrained simulations was obtained by adding another constraint in the simulation process. The first inversion had to form stratum 3 (markers 11, 12). Then, two inversions had to take place in the following order: one inversion including all stratum 4 markers (with the possibility of including some markers of stratum 3) and a second inversion including all stratum 5 markers (with the possibility of including some markers of strata 4 and 3). The other inversions occurred in a non-determined order within the already formed strata. All inversions had to fulfil the parsimony criterion (see above). Different values of *d* were tested and gave very similar results. Results shown in the paper (Table 2, Figure 2) are with *d* = 8 (results for other values of *d* are shown in the supplementary data)

Analysis of breakpoints

We used the method by Lemaitre *et al.* (2008). Briefly, this method works on breakpoint regions. A breakpoint is defined by two adjacent markers on one sequence (here X) that are not adjacent on the homologous sequence (here Y). By aligning an X region with a breakpoint with its two corresponding Y regions, it is possible to locate precisely the breakpoint and to analyse it. Alignments were performed with BlastZ (Schwartz *et al.* 2003) on repeat-free (using RepeatMasker) sequences. The minimum breakpoint interval is obtained by looking at the distribution of hits using a partitioning algorithm.

Levels of divergence between duplicates

Duplicates found with the method described in the previous section were aligned on the entire Y chromosome and no other copies were found. The level of divergence between duplicates was obtained using exact pairwise alignment tools (the Water and Matcher programs from the EMBOSS tools suite, see Rice *et al.* 2000).

For the analysis of *XG*, we ran PAML on coding sequences to get d_N and d_S values (Yang 1997, 2007 and PAML on the web: http://coot.embl.de/pal2nal/, Suyama *et al.* 2006). The percentage of similarity for total (coding + non-coding) DNA was obtained by aligning sequences using BlastZ with the chaining option (Schwartz *et al.* 2003). Only blocks of more than 70 % of similarity were kept. To compute the % of similarity, we summed all the identical sites and divided by the size of the X sequence.

Results and Discussion

Comparing scenarios for X-Y chromosomal rearrangements and evolutionary strata 4 and 5

Ross *et al.* (2005) presented a scenario of inversions between the human X and Y chromosomes consistent with the evolutionary strata 3, 4 and 5. They obtained it using GRIMM (Tesler 2002), a software that uses marker order in two sequences to propose a scenario of inversions minimising the total number of inversion events and applied it to 12 pairs of X-Y markers spanning a small part of stratum 3, the whole strata 4, 5 and PAR (and totalling 11 Mb of sequences). In the proposed scenario (shown in Figure 1), there are two large inversions that coincide with strata 4 and 5 and could have formed them, and five small inversions that are included in the strata and do not affect strata boundaries. They also found an inversion consistent with stratum 3 but because only a small part of this stratum was investigated, no conclusion could be drawn. This showed for the first time since Lahn and Page (1999) had proposed their model for the evolution of strata that it was possible to find a scenario of inversions consistent with the human strata, at least the most recent ones (i.e. strata 4 and 5). However, there is an ongoing debate about how relevant is the scenario proposed by GRIMM and other similar programs. GRIMM uses the Hannenhalli-Pevzner (1995) algorithm and we know that it is an accurate way of getting the minimum number of inversions between two sequences with different marker orders. The scenario proposed by GRIMM however is only one possible optimal scenario (i.e. with the minimum number of inversions). There may be many other such optimal scenarios and GRIMM does not offer the possibility of identifying and analysing these scenarios. This of course weakens the conclusion on the evolution of human strata from the GRIMM results. The proposed scenario is consistent with strata formation but there may be equally good scenarios not consistent with strata formation. To solve this problem in general, it was suggested to enumerate all the possible optimal scenarios (Siepel 2003). The result of this showed that the number of optimal scenarios is often huge. Such an enumeration method is however too demanding in terms of computational time. A mathematical formalism was proposed to fasten this process (Bergeron *et al.* 2002). Briefly, the idea is to group optimal scenarios into classes of equivalence. All scenarios in any one of the classes are composed by the same inversions but in different order. Finding an efficient algorithm to enumerate all classes of equivalence remained however an open problem for a few years, until Braga *et al.* (2008) designed and implemented one, which is efficient when the number of rearranged markers is not too large.

We applied this method to the same 12 markers used by Ross *et al.* (2005) to evaluate how their conclusion was affected by analysing all the optimal scenarios (see Table 1 and Figure 1 for more information about these 12 markers). We found six classes of equivalence (i.e. groups of optimal scenarios with the same inversions but in different order). The solution proposed by GRIMM is in one of them but there are 5 other classes with different inversions for a total of 31752 optimal scenarios. We then counted the number of optimal scenarios consistent with strata 3, 4 and 5 with strata boundaries as defined in Lahn & Page (1999), Skaletsky *et al.* (2003) and corrected by Ross *et al.* (2005). We expected that in these scenarios, there were three ordered inversions affecting, first markers in stratum 3 (markers 11, 12), then markers in stratum 4 (markers 3 to 10), and finally markers in stratum 5 (markers 1, 2). A given inversion forming a stratum could comprise additional markers from the previous stratum (for instance, markers 11 and 12 could be involved in a large inversion forming stratum 4 as in Figure 1). The remaining inversions were small-scale ones occurring within already formed strata. With these criteria, we found that only one class of equivalence – the one including the GRIMM scenario – agrees with the currently defined strata 3, 4 and 5 (see Table 2). Inside this class, 420 scenarios were found consistent with these strata. We also looked at scenarios consistent with 3 strata with different boundaries than the ones proposed in Ross *et al.* (2005), with 2 strata, with one stratum only and without strata (see Table 2). In each case, we found a number of optimal scenarios consistent with the tested strata structure depending on the classes (see Table 2). When pooling results for all classes, we found more optimal scenarios for the currently defined 3 strata (420) than for the alternative 3 strata (120) but there are more scenarios for 2 strata (2520) and 1 stratum (2520) and even more for no stratum at all (26172) than for the currently defined 3 strata.

We then performed simulations with random inversions in order to help us interpret the data. We did a first round of simulations with free inversions (see Methods) among 12 markers with the number of inversions being equal to the observed number for the human X-Y data of Ross *et al.* (2005). In the second round of simulations, the idea was to model the formation of strata by inversions. We simulated inversions that were constrained by 3 strata with boundaries mimicking the currently defined strata 3, 4 and 5 (see Methods). In the process of simulations, inversions had to be ordered (starting with the oldest strata markers and finishing with the youngest) and had to respect strata boundaries. Small inversions could occur in already formed strata only. We called this set of simulations the “strata-constrained inversions”. We then ran the method by Braga *et al.* (2008) on the simulated sequences (comparing each time the initial sequence and one sequence with simulated inversions). This generated distributions of parameters such as total number of optimal scenarios, total number of classes of equivalence and others. The mean values of these parameters are shown in Table 3. We compared the distributions of free versus strata-constrained simulations by using non-parametric statistics. For all of them, the distributions for free and strata-constrained simulations are significantly different (see Table 3). The most relevant parameter is probably the number of optimal scenarios consistent with the 3 strata over the total number of optimal scenarios (#strata_scen/#tot_scen). We found that this parameter is significantly different for free and strata-constrained simulations ($p < 10^{-16}$). Figure 2 shows the distributions of this parameter for the two sets of simulations. The observed value for the human XY data (from Table 2) is a clear outlier in the “free” distribution. We get a p-value of 0.009 from this comparison, which means that it is very unlikely that the inversions between X and Y (in the region studied by Ross *et al.* (2005)) occurred freely along the Y sequence without any constraints. The comparison of the observed value of #strata_scen/#tot_scen for the human XY data with the distribution of the same parameter for the strata-constrained simulations resulted in a non-significant p-value, which means that the process used to simulate this set may be the same for the human sex chromosomes. Although the two sets of simulations that we generated are extreme cases and we did not investigate intermediary cases, these results suggest that the hypothesis that evolutionary strata 4 and 5 (no conclusion can be drawn for stratum 3, see above) have been formed by Y inversions is a likely hypothesis.

Looking at chromosomal breakpoints near strata 4 and 5 boundaries

A strong evidence for the model of Y inversions put forward by Lahn & Page would be to find a region spanning a strata boundary on the X chromosome matching with two broken bits on the Y chromosome. At the pseudoautosomal boundary, the Y copy of the *XG* gene is truncated. If stratum 5 had been formed by a Y inversion, we should be able to find the missing bit of *XG* at the end of the putative inversion(s) having formed stratum 5 on the Y chromosome (upstream of marker 1 on the Y sequence on Figure 1). The same rationale can be applied to stratum 4. To check this, we used a method to precisely detect and analyse chromosomal breakpoints in sequences. This method starts with a BlastZ (Schwartz *et al.* 2003) comparison between two sequences where there is a breakpoint (Lemaitre *et al.* 2008). It maps all the hits on the sequences and uses the distribution of hits to find the minimum interval where the breakpoint is (see Methods). It gives a precise picture of the similarities at breakpoints.

Figure 3A shows the results for PAR/stratum 5. We found similarities between X and Y at the end of the PAR region (from 2.67 to 2.71 Mb on both X and Y sequences). This includes the part of the *XG* gene found both in X and Y that defines the human pseudoautosomal boundary. Interestingly, we found also similarities between PAR and a sequence around position 13 Mb of the Y chromosome that happens to be one of the ends of stratum 5 on that chromosome. These similarities extend over 40 Kb and mirror the similarities found in the PAR, which clearly indicates an inverted duplication. Instead of finding only the missing bit of the *XG* region when looking at the end of stratum 5 in the Y chromosome, we thus found an inverted duplication of the almost entire *XG* region. Analysis of the strata 4/5 boundary gave very similar results (see Figure 3B). Again the dotplot indicates duplications with similarities between the X region from 3.66 to 3.74 Mb (end of stratum 5) and both Y regions from 7.06 to 7.18 Mb (end of stratum 5) and from 19.72 to 19.8 Mb (beginning of stratum 4). Duplicated sequences flanking stratum 4 suggest again a single inversion event spanning the whole stratum 4. In Figure 3C, we show the duplicates and their orientations on the X and Y sequences for PAR/stratum 5 and stratum 5/stratum 4.

Inversions are often found associated with duplications (Casals & Navarro 2007, Kehrer-Sawatzki & Cooper 2008). This association has been interpreted as the result of non-allelic homologous recombination between duplicates. Indeed, there are well-documented cases of such mechanism. In humans for instance, there is a polymorphic inversion on chromosome Xq28 that includes the *FLNA* and *EMD* loci and that is flanked by inverted duplicates. It was shown that these inverted duplicates are present in all placental mammals and that there is a recurrent inversion of the segment between these duplicates in several mammalian lineages (Caceres *et al.* 2007). However, some inversions come from another mechanism (Casals & Navarro 2007, Kehrer-Sawatzki & Cooper 2008). Human and chimp genomes differ by several chromosomal rearrangements. One of them is a pericentric inversion in the chromosome 10 of chimpanzee in the region around the *SLCO1B3* gene. A comparison between humans and chimps revealed that this pericentric inversion produced a duplication found only in chimps, which suggests that the inversion generated the duplication and not the contrary (Kehrer-Sawatzki *et al.* 2005). More recently, a multigenome comparison in drosophila revealed that 60% of the inversions produced duplications and were formed by a mechanism called isochromatid model with staggered single-strand breaks (Ranz *et al.* 2007). In this mechanism, two pairs of staggered single-strand breaks result in long 5'-overhangs, which can then be filled in by DNA synthesis. When followed by a repair pathway called nonhomologous end joining (NHEJ), this results in an inversion flanked by inverted duplications of the sequences between the paired single-strand breaks (Ranz *et al.* 2007).

Both models could explain our results. A Y-specific duplication could have occurred and then an inversion between the duplicates in agreement with the non-allelic recombination between duplicates model. The Y inversion itself could have produced the duplicates as in the isochromatid model with staggered single-strand breaks. We tend to favour the latter because it is more parcimonious (one event – the inversion – in the isochromatid model with staggered single-strand breaks instead of two events – a duplication and an inversion – in the non-allelic recombination between duplicates model). In Figure 3D, we show a scenario of the formation of strata 4 and 5 consistent with the isochromatid model with staggered single-strand breaks but in any case, our results clearly point towards a single inversion event spanning the whole stratum 5 and another similar event for stratum 4. An important point is that orientation of the duplicates is fully compatible with two large inversions giving rise to stratum 4 first and then to stratum 5 (see Figure 3D). Indeed, if we invert stratum 5 back in place on the Y chromosome, the duplicates for stratum 4 are in inverted orientation. We also compared the X and Y duplicated sequences found at PAR/stratum 5 and strata 4/5 (see Methods). The level of divergence between duplicated segments is 30% for PAR/stratum 5 and 50% for strata 4/5, which is fully consistent with the fact that the formation of stratum 5 is more recent than the formation of stratum 4. However, the divergence is clearly higher than the reported divergence for stratum 5 (5%) and stratum 4 (10-15%) (see Skaletsky *et al.* 2003, Iwase *et al.* 2003, Ross *et al.* 2005). This may be because these estimates of divergence and ours have been obtained by different methods. In previous work, the estimates mainly come from the comparison of synonymous sites of coding regions (d_S values). Our estimates have been obtained on X and Y regions that include non-homologous sequences (i.e. DNA repeats and other inserted/deleted sequences), which decreases the quality of the global alignment and increases divergence. Nevertheless, our results strongly suggest that there are footprints of inversions at the recent strata boundaries and that these inversions have produced the strata.

Evidence for gene conversion between *XG* copies

A more careful analysis of the divergence pattern between the pair of duplicates (namely the *XG* gene) associated with the stratum 5 inversion gave unexpected results. *XG* has three copies: the X copy (*XG-X*) and two Y copies: the one in the PAR (*XG-Y1*) and the entire copy in the non-recombining Y hereafter called NRY (*XG-Y2*). *XG-Y1* and a part of *XG-X* are in the PAR and are nearly identical. We extracted from ENSEMBL v49 the sequences of *XG-X* and *XG-Y2*. We first computed the d_N and d_S using PAML (Yang 1997, 2007) for the exons of these copies (see Figure 4). Surprisingly, we found very different results for the part common to *XG-X*, *XG-Y1* and *XG-Y2* (hereafter called *XG-5'*) and for the other part (found only in *XG-X* and *XG-Y2*, hereafter called *XG-3'*). *XG-5'* has a lower d_S value (0.060) than *XG-3'* (0.091) and *XG-5'* has a much lower d_N/d_S ratio (0.019) than *XG-3'* (1.01). The results for *XG-3'* are in agreement with *XG-Y2* being a pseudogene (d_N/d_S ratio of 1) but the lower d_S value and the much lower d_N/d_S ratio for *XG-5'* suggests genetic exchanges from the functional *XG-X* to the non-functional *XG-Y2*. This is striking since *XG-Y2* is in the NRY and is not expected to recombine. The analysis of total DNA (including exons and introns) confirmed the results for the exons only (see Figure 4). We found a higher % of similarity for *XG-5'* (83.26%) than for *XG-3'* (67.32%) when comparing *XG-X* and *XG-Y2*. Genetic exchanges from *XG-X* to *XG-Y2* are not expected. This could have happen by X-Y gene conversion involving *XG-X* and *XG-Y2* or by Y-Y gene conversion involving *XG-Y1* and *XG-Y2*. We know that Y-Y gene conversion exists in humans since evidence of strong gene conversion has been found among Y genes of the same multigene families (Rozen *et al.* 2003, Bhowmick *et al.* 2007). The drop in divergence between *XG-X* and *XG-Y2* seems to lie in a region of about 8 Kb where both copies share more than 95% of identity, which is consistent with a single very recent event of gene conversion.

These results have important implications for *AMEL*. The boundary between strata 3 and 4 was thought to lie within *AMEL* because of a drop of divergence from 30 to 10% within this gene (Iwase *et al.* 2003). But *AMELY* is not truncated and this was considered as evidence that there was no inversion affecting *AMEL*. *AMEL* was clearly a problem for the Lahn & Page model of strata formation by Y inversion until it was found that *AMELX* and *AMELY* showed evidence for gene conversion, which could explain the peculiar divergence pattern of *AMEL* (Marais & Galtier 2003, Ross *et al.* 2005). The strata 3/4 boundary has been put between *KAL1X* and *TBL1X* by Ross *et al.* (2005) but this boundary is still debated. Our results on *XG* suggest that the strata 3/4 boundary could be in *AMEL*. An inversion could have formed stratum 3 and produced two copies of *AMEL* on the Y chromosome with one copy being complete. The same kind of configuration as for XG could have existed with a part of *AMELX* and a truncated *AMELY* in the PAR and an entire *AMELY* in the NRY with possibility of gene conversion between these copies. This would have produced the divergence pattern that we now observed when comparing *AMELX* and *AMELY*, which resembles that of *XG-X* and *XG-Y2*. We looked for a truncated *AMELY* by BlastZ search (*AMELY* against the whole Y chromosome) and found no other hit than *AMELY* itself. However, the formation of stratum 3 is an ancient event and the truncated *AMELY*, which was made a pseudogene at that time, may well be no longer recognisable or may have been deleted.

Discussing the case of the ancient human strata

Following Ross *et al.* (2005), we focused mainly on strata 4 and 5 in the previous sections. In the first section, we had some markers from stratum 3 but they have been mainly used to delimit stratum 4, and stratum 3 was not analysed entirely. We only included two stratum 3 markers as in Ross *et al.* (2005). In this section, we address the question whether stratum 3 was formed by a single inversion on the Y chromosome. We used all the markers available in the literature for stratum 3 plus 3 new markers that we found when looking for similarities between the X and Y sequences. In total, we had 23 markers (stratum 5: 2, stratum 4: 7+ 1 new, stratum 3: 11 + 2 new) that with the PAR covered the first 45 Mb of the X (see Table 1). We ran our method on the set of 23 markers. We could still find optimal scenarios consistent with the formation of strata 4 and 5 by Y inversions, which shows that the conclusions obtained with 12 markers remain unchanged by adding more markers. We could not however find any scenario consistent with the formation of stratum 3 by a Y inversion (using strata 3/4 boundary as in Ross *et al*. 2005).

There are several explanations for that. One is that the strata 3/4 boundary is not well defined and this may affect the results. When we put the strata 3/4 boundary at *AMELX* (by removing the *AMEL* marker and putting the boundary at its place), the results remain unchanged. The second possible explanation is that stratum 3 was formed by a mechanism different from Y inversions. Consistent with this idea, when we analysed stratum 3 with our method (Lemaitre *et al.* 2008) but we could find no clear evidence (no duplications) of large inversions as we found for strata 5 and 4. In *Silene latifolia* - a dioecious plant with recently evolved (< 10 mya) sex chromosomes - 3 strata have already evolved (Nicolas *et al.* 2005, Bergero *et al.* 2007). Maps for the X and Y chromosomes are being developed and preliminary data suggest that stratum 2 may have been formed by a large paracentric inversion but for strata 1 and 3 the mechanism is less clear (Bergero *et al.* 2008). Stratum 3 may have been formed by translocation in some populations but not all, which suggests that other type of chromosomal rearrangements than inversion could form strata. However, the only translocation that has affected stratum 3 seems to be the translocation of the entire XAR (see Ross *et al*. 2005) and it is not clear why this would have stopped recombination at some part of XAR (e.g. stratum 3) and not others (strata 3, 4 and the current PAR). Moreover, absence of footprints of an inversion spanning stratum 3 may be simply due to too high level of divergence between X and Y sequences for this stratum.

A third explanation is that stratum 3 is not just one stratum but several. It is interesting to notice that more ancient strata get larger and larger. Stratum 1 alone covers the whole q arm of the X chromosome (Lahn & Page 1999, Skaletsky *et al.* 2003). This of course is surprising and may be simply due to the difficulty in identifying distinct strata when these strata are ancient. Stratum 3 may well include several strata that were formed successively in a short period of time and that are no longer distinguishable simply because the levels of divergence between the X and Y sequences in these strata are very similar. Another line of evidence supporting this hypothesis is that a similar number of strata has been observed in recent and ancient heteromorphic sex chromosomes. Ancient sex chromosomes such as that of humans and chicken have five and three strata respectively (Ross *et al*. 2005, Nam & Ellegren 2008) and recently evolved but already differentiated XY such as that of *S. latifolia* also have three strata (Nicolas *et al.* 2005, Bergero *et al.* 2007, 2008). This suggests that strata can accumulate at a fast rate and that in old sex chromosomes old strata may not be distinguishable.

Interestingly, it is possible to find an optimal scenario consistent with the markers in stratum 3 up to *CASK*, which suggests that stratum 3 could comprise a first stratum from the strata 3/4 boundary (either between *KAL1X* and *TBL1X* or at *AMELX*) to *CASK* and another one including *UTX*. We know that heterozygotes for an inversion have recombination suppressed at the inversion. Inhibition of the molecular mechanism of recombination or selection against gametes with chromosomal rearrangements could explain this suppression. The suppression is complete for pericentric inversions but for paracentric inversions, it is stronger at the inversion breakpoints (Navarro *et al*. 1997, Andolfatto *et al*. 2001). For sufficiently large paracentric inversions, recombination is not suppressed in the middle of the inversion. Indeed, for such inversions, the probability of two crossovers occurring within the inversions (and avoiding chromosomal rearrangements) can be high. All this seems to be also true for inversions affecting sex chromosomes in *Drosophila americana* (McAllyster 2003, Evans *et al.* 2007) and the black muntjac (Zhou *et al*. 2008). Recombination between sex chromosomes will be suppressed efficiently with relatively small inversions only, which reinforces the idea that the ancient strata that we have defined based on X-Y divergence such as stratum 3 are in fact a mosaic of smaller strata that are no longer distinguishable.

Concluding remarks

Our results strongly suggest that strata 4 and 5 – the most recent strata in humans - were formed by large Y inversions, which give support to the model proposed by Lahn & Page that X-Y recombination has been suppressed by Y inversions. For stratum 3 – an older stratum – we could not find evidence for a large Y inversions. This may be due to a wrong definition of the strata 3/4 boundary, to the formation of stratum 3 by a mechanism different from Y inversions or more likely to the existence of several strata (maybe 2) within stratum 3 or (see previous section). This illustrates the difficulty of working on an ancient stratum; stratum 3 was formed before the radiation of the principal placental mammalian orders (Lahn & Page 1999). In such an ancient stratum, high levels of X-Y divergence, paucity of Y markers, Y deletions in great number and a multitude of X-Y rearrangements make reliable inferences difficult. The task is even more difficult for strata 1 and 2. These strata were formed at the very early stages of the XY evolution probably before the placental/marsupial split (Lahn & Page 1999, for the age of the human XY see Rens *et al.* 2007, Veyrunes *et al.* 2008, Portzebowsky *et al.* 2008). These strata are extremely differentiated with only a handful of markers still detectable on both sex chromosomes and with a massive gene loss on the Y chromosome (stratum 1: X=588 genes, Y=3 genes, stratum 2: X=151, Y=2 genes, from ENSEMBL v49 data). Moreover, the X region with strata 1,2 and its homolog in chicken are rearranged (see Ross *et al.* 2005) and the absence of outgroup makes almost impossible the reconstruction of the X-Y rearrangements for these strata. We probably need to look at more recently evolved sex chromosomes systems such as *S. latifolia* XY to obtain more data on the formation of strata, especially the first ones.

Acknowledgements

We thank Laurent Duret, Mark Kirkpatrick, Eduardo Rocha, and David Sankoff for helpful comments on this ms. This work was partially supported by the ANR (REGLIS project NT05-3_45205 and GENOMICRO project ANR-05-JCJC-0139-01). MVDB is supported by the Programme AlBan, the European Union Programme of High Level Scholarships for Latin America, scholarship no. E05D053131BR.

Supplementary Material

1) Classes of equivalence found for the X-Y chromosomes (12 markers analysis)

2) Simulations with different numbers of inversion (*d*) values and subsequent analysis

Literature cited

Andolfatto P, Depaulis F, Navarro A. Inversion polymorphisms and nucleotide variability in Drosophila. Genet Res. 2001 Feb;77(1):1-8.

Bachtrog D. The temporal dynamics of processes underlying Y chromosome degeneration. Genetics. 2008 Jul;179(3):1513-25.

Bergero R, Charlesworth D, Filatov DA, Moore RC. Defining regions and rearrangements of the Silene latifolia Y chromosome. Genetics. 2008 Apr;178(4):2045-53.

Bergero R, Forrest A, Kamau E, Charlesworth D. (2007) Evolutionary strata on the X chromosomes of the dioecious plant Silene latifolia: evidence from new sex-linked genes. Genetics. 75: 1945-54.

Bergeron A., Chauve C., Hartmann T., St-Onge K., “On the properties of sequences of reversals that sort a signed permutation”. JOBIM 2002, 99-108.

Bhowmick BK, Satta Y, Takahata N. The origin and evolution of human ampliconic gene families and ampliconic structure. Genome Res. 2007 Apr;17(4):441-50.

Bhutkar A, Schaeffer SW, Russo SM, Xu M, Smith TF, Gelbart WM. Chromosomal rearrangement inferred from comparisons of twelve Drosophila genomes. Genetics. 2008 Jul 13.

Braga M. D. V, Sagot M.-F., Scornavacca C. and Tannier E. (2008) Exploring the solution space of sorting by reversals with experiments and an application to evolution, Transactions on Computational Biology and Bioinformatics in press.

Cáceres M; National Institutes of Health Intramural Sequencing Center Comparative Sequencing Program, Sullivan RT, Thomas JW. A recurrent inversion on the eutherian X chromosome. Proc Natl Acad Sci U S A. 2007 Nov 20;104(47):18571-6.

Carrel L and Willard HF. X-inactivation profile reveals extensive variability in X-linked gene expression in females. Nature 2005 March 17 ; 434 400-404.

Casals F, Navarro A. Chromosomal evolution: inversions: the chicken or the egg? Heredity. 2007 Nov;99(5):479-80.

Charlesworth D, Charlesworth B, Marais G. Steps in the evolution of heteromorphic sex chromosomes. Heredity 2005 95(2):118-28.

Charlesworth D, Charlesworth B. A Model for the Evolution of Dioecy and Gynodioecy. The American Naturalist, 1978 112 : 975-997.

Evans AL, Mena PA, McAllister BF. Positive selection near an inversion breakpoint on the neo-X chromosome of Drosophila americana. Genetics. 2007 Nov;177(3):1303-19.

Fischer G, Rocha EP, Brunet F, Vergassola M, Dujon B. Highly variable rates of genome rearrangements between hemiascomycetous yeast lineages. PLoS Genet. 2006 Mar;2(3):e32.

Handley LJ, Ceplitis H, Ellegren H. (2004) Evolutionary strata on the chicken Z chromosome: implications for sex chromosome evolution. Genetics. 167: 367-76.

Hannenhalli, S. and Pevzner, P. A. Transforming men into mice (polynomial algorithm for genomic distance problem), InProceedings of the IEEE 36th Annual Symposium on Foundations of Computer Science, pages 581-592, 1995.

Iwase M, Satta Y, Hirai Y, Hirai H, Imai H, Takahata N. The amelogenin loci span an ancient pseudoautosomal boundary in diverse mammalian species. Proc Natl Acad Sci U S A. 2003 Apr 29;100(9):5258-63.

Kehrer-Sawatzki H, Cooper DN. Molecular mechanisms of chromosomal rearrangement during primate evolution. Chromosome Res. 2008;16(1):41-56.

Kehrer-Sawatzki H, Sandig CA, Goidts V, Hameister H. Breakpoint analysis of the pericentric inversion between chimpanzee chromosome 10 and the homologous chromosome 12 in humans. Cytogenet Genome Res. 2005;108(1-3):91-7.

Kirkpatrick M, Barton N. Chromosome inversions, local adaptation and speciation. Genetics. 2006 May;173(1):419-34.

Lahn BT and Page DC. (1999) Four evolutionary strata on the human X chromosome. Science 286: 964-967

Lemaitre C, Tannier E, Gautier C, Sagot M-F. (2008) Precise detection of rearrangement breakpoints in mammalian chromosomes. BMC Bioinformatics 9: 286.

McAllister BF. Sequence differentiation associated with an inversion on the neo-X chromosome of Drosophila americana. Genetics. 2003 Nov;165(3):1317-28.

Marais G and Galtier N. Sex chromosomes: how X-Y recombination stops. Curr Biol. 2003 Aug 19;13(16):R641-3

Murphy WJ, Larkin DM, Everts-van der Wind A, Bourque G, Tesler G, Auvil L, Beever JE, Chowdhary BP, Galibert F, Gatzke L, Hitte C, Meyers SN, Milan D, Ostrander EA, Pape G, Parker HG, Raudsepp T, Rogatcheva MB, Schook LB, Skow LC, Welge M, Womack JE, O'brien SJ, Pevzner PA, Lewin HA. Dynamics of mammalian chromosome evolution inferred from multispecies comparative maps. Science. 2005 Jul 22;309(5734):613-7.

Nam K, Ellegren H. Scrambled Eggs: The Chicken (Gallus gallus) Z Chromsome Contains at Least Three Non-linear evolutionary Strata. Genetics. 2008 Sep 14. [Epub ahead of print]

Navarro A, Betrán E, Barbadilla A, Ruiz A. Recombination and gene flux caused by gene conversion and crossing over in inversion heterokaryotypes. Genetics. 1997 Jun;146(2):695-709.

Nei M. Linkage modifications and sex difference in recombination. Genetics. 1969 Nov;63(3):681-99.

Nicolas M, Marais G, Hykelova V, Janousek B, Laporte V, Vyskot B, Mouchiroud D, Negrutiu I, Charlesworth D, Monéger F. (2005) A gradual process of recombination restriction in the evolutionary history of the sex chromosomes in dioecious plants. PLoS Biol. 3: e4.

Potrzebowski L, Vinckenbosch N, Marques AC, Chalmel F, Jégou B, Kaessmann H. Chromosomal gene movements reflect the recent origin and biology of therian sex chromosomes. PLoS Biol. 2008 Apr 1;6(4):e80.

Ranz JM, Maurin D, Chan YS, von Grotthuss M, Hillier LW, Roote J, Ashburner M, Bergman CM. Principles of genome evolution in the Drosophila melanogaster species group. PLoS Biol. 2007 Jun;5(6):e152.

Rens W, O'Brien PC, Grützner F, Clarke O, Graphodatskaya D, Tsend-Ayush E, Trifonov VA, Skelton H, Wallis MC, Johnston S, Veyrunes F, Graves JA, Ferguson-Smith MA. The multiple sex chromosomes of platypus and echidna are not completely identical and several share homology with the avian Z. Genome Biol. 2007;8(11):R243.

Rice,P. Longden,I. and Bleasby,A. EMBOSS: The European Molecular Biology Open Software Suite. Trends in Genetics vol. 16, (6) p. 276--277, 2000.

Ross MT *et al.* (2005) The DNA sequence of the human X chromosome. Nature 434: 325-337.

Rozen S, Skaletsky H, Marszalek JD, Minx PJ, Cordum HS, Waterston RH, Wilson RK, Page DC Abundant gene conversion between arms of palindromes in human and ape Y chromosomes. Nature. 2003 Jun 19;423(6942):873-6.

Sandstedt SA and Tucker PK. (2004) Evolutionary strata on the mouse X chromosome correspond to strata on the human X chromosome. Genome Res. 14: 267-72.

Schwartz, S., Kent, W.J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R.C., Haussler, D., Miller, W. Human-mouse alignments with BLASTZ. Genome Res, vol. 13, p. 103--107, 2003.

Siepel A. An algorithm to enumerate sorting reversals for signed permutations. J

Comput Biol 10:575-597, 2003.

Skaletsky H *et al.* (2003) The male-specific region of the human Y chromosome is a mosaic of discrete sequence classes. Nature 423: 825-837.

Suyama M, Torrents D, and Bork P (2006). PAL2NAL: robust conversion of protein sequence alignments into the corresponding codon alignments. Nucleic Acids Res. 34, W609-W612.

Tesler G. GRIMM: genome rearrangements web server. Bioinformatics. 2002 Mar;18(3):492-3.

Veyrunes F, Waters PD, Miethke P, Rens W, McMillan D, Alsop AE, Grützner F, Deakin JE, Whittington CM, Schatzkamer K, Kremitzki CL, Graves T, Ferguson-Smith MA, Warren W, Marshall Graves JA. Bird-like sex chromosomes of platypus imply recent origin of mammal sex chromosomes. Genome Res. 2008 Jun;18(6):965-73.

Yang Z. PAML: a program package for phylogenetic analysis by maximum likelihood. Comput Appl Biosci. 1997 Oct;13(5):555-6.

Yang Z. PAML 4: phylogenetic analysis by maximum likelihood. Mol Biol Evol. 2007 Aug;24(8):1586-91. Epub 2007 May 4.

Yogeeswaran K, Frary A, York TL, Amenta A, Lesser AH, Nasrallah JB, Tanksley SD, Nasrallah ME. Comparative genome analyses of Arabidopsis spp.: inferring chromosomal rearrangement events in the evolutionary history of A. thaliana. Genome Res. 2005 Apr;15(4):505-15.Zhou Q, Wang J, Huang L, Nie W, Wang J, Liu Y, Zhao X, Yang F, Wang W. Neo-sex chromosomes in the black muntjac recapitulate incipient evolution of mammalian sex chromosomes. Genome Biol. 2008;9(6):R98

Figure legends

Figure 1: **A possible scenario for the X-Y rearrangements and the evolution of recent human strata** (adapted from Ross *et al.* 2005). This scenario has been obtained by GRIMM using 12 markers covering PAR – stratum 5 – stratum 4 and beginning of stratum 3. PAR = pseudoautosomal region. Strata definitions are from Ross *et al.* (2005). See list of markers in Table 1. Inversions that coincide with strata and that could have formed them are indicated in red. Other inversions are in brown.

Figure 2: **Distributions of the number of optimal scenarios consistent with the human strata 4 and 5 over the total number of optimal scenarios (# strata_scen/# total_scen) for free and strata-constrained simulated sequences**. Free simulations have been obtained by random inversions (black boxes). Strata-constrained simulations have been obtained by simulating formation of strata by inversions (using currently defined strata 3, 4 and 5 for humans) with additional small inversions occurring within strata after their formation (white boxes). See main text (Results and Discussion and Methods) for more details. # strata_scen/# total_scen values for simulated sequences have been obtained using the Braga *et al.* (2008) program. The value observed for the true X-Y sequences is indicated by a red arrow.

Figure 3: **Analysis of breakpoints at strata and PAR boundaries in humans**. A) Dotplot for the PAR/stratum 5 boundary. This shows the similarities between the X region at the PAR/stratum 5 boundary with two “broken” regions on the Y. It clearly shows that the X region is duplicated on the Y (with one duplicate being inverted). Total length of the region = 45 Kb. B) Dotplot for the strata 4/5 boundary. This shows the similarities between the X region at the strata 4/5 boundary with two “broken” regions on the Y. It clearly shows that the X region is duplicated on the Y (both Y duplicates are in inverted orientation compared to the X homologous region). Total length of the region = 110 Kb. See main text (Results and Discussion and Methods) for more details. C) Picture showing the location and orientation of the duplications on the chromosomes X and Y. On the Y chromosome, stratum 5 is flanked by duplication of the PAR/stratum 5 region of the X (shown in red), which indicates a large inversion spanning the entire stratum 5. Duplicates of the strata 4/5 region of the X are found at the ends of stratum 5 and stratum 4 (shown in green). This defines a large inversion spanning the whole stratum 4. Importantly, duplicates are in an orientation consistent with two large inversions that have formed stratum 4 first and then stratum 5. D) Sketch showing the scenario with two inversions for the formation of strata 4 and 5 under the isochromatid model with staggered single-strand breaks (see text). The first inversion reduces the size of the PAR and forms stratum 4 with two inverted duplicates flanking the inversion. The second inversion reduces further the size of the PAR and forms stratum 5 with two inverted duplicates flanking the inversion. Note that duplicates associated with the formation of stratum 4 are no longer inverted because one of them is involved in the inversion that has formed stratum 5. Lines with arrows indicate inversions. In C) and D): Blocks of similarities are indicated by blue boxes and shadows. Black lines indicate large stretches of non-homologous sequences. Sizes of boxes and lines are not in scale.

Figure 4: **Divergence patterns among *XG* copies**. The three copies of *XG* are shown: the X copy (*XG-X*) and two Y copies: the one in the PAR (*XG-Y1*) and the entire copy in the NRY (*XG-Y2*). *XG-X* and *XG-Y2* have been compared. Two regions were defined: *XG-5’* (1-29500) and *XG-3’* (29500-63854). d_S and d_N were estimated using PAML and the % of similarity for total DNA was obtained using BlastZ (see Material and Methods). Grey arrows indicate possible events of gene conversion.

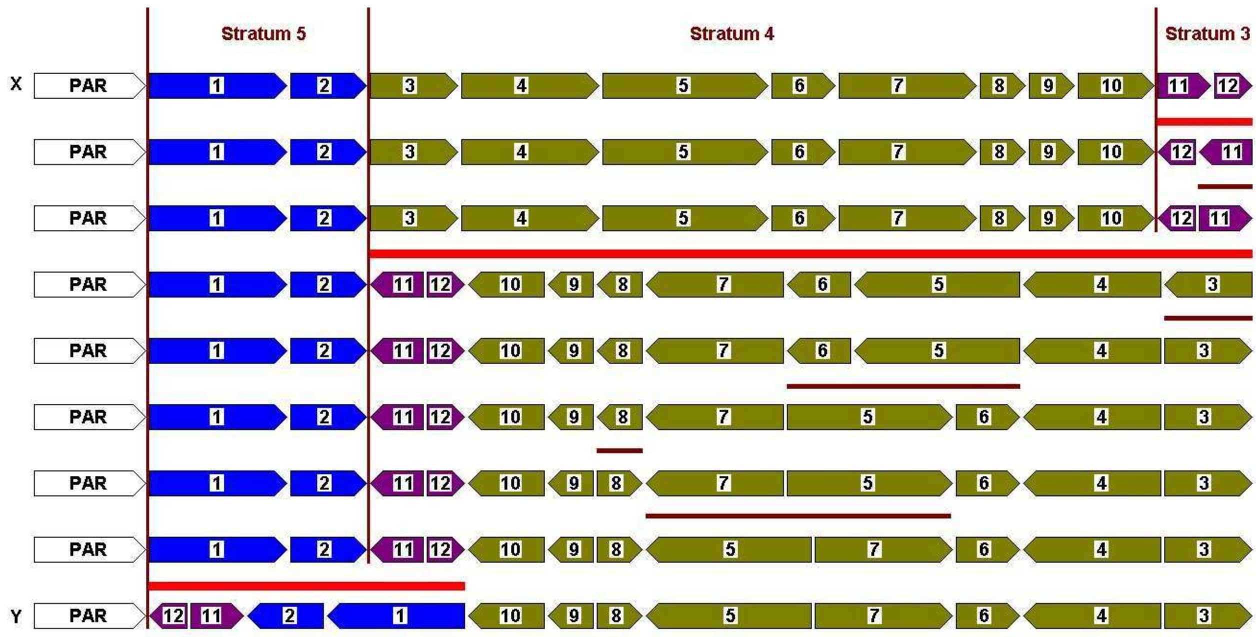


Figure 1

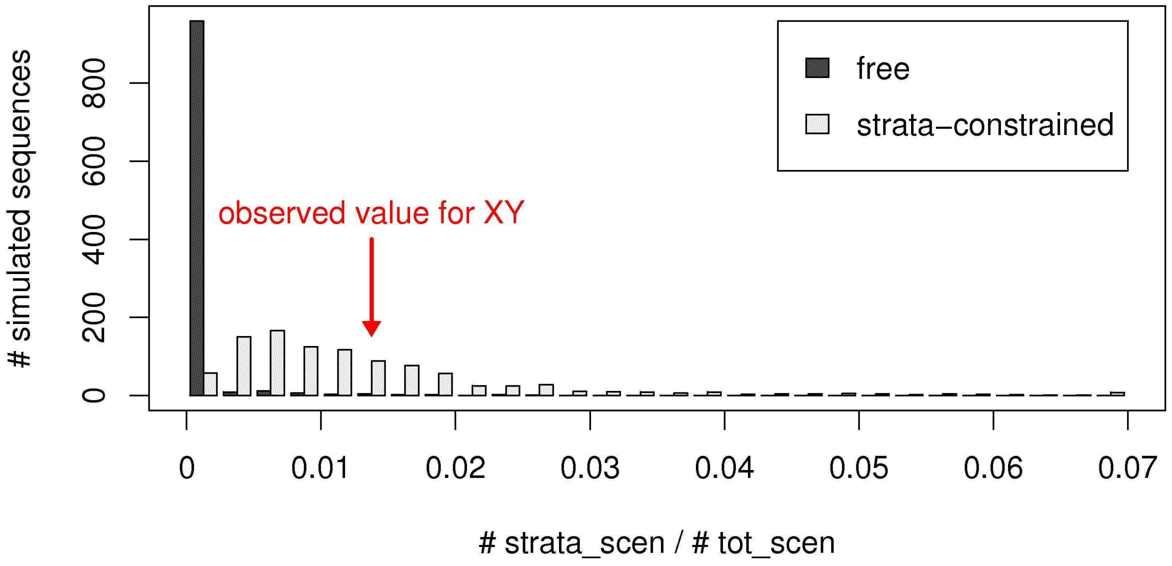

free
strata−constrained
observed value for XY
# simulated sequences
0
200
400
600
800
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
# strata_scen / # tot_scen

Figure 2

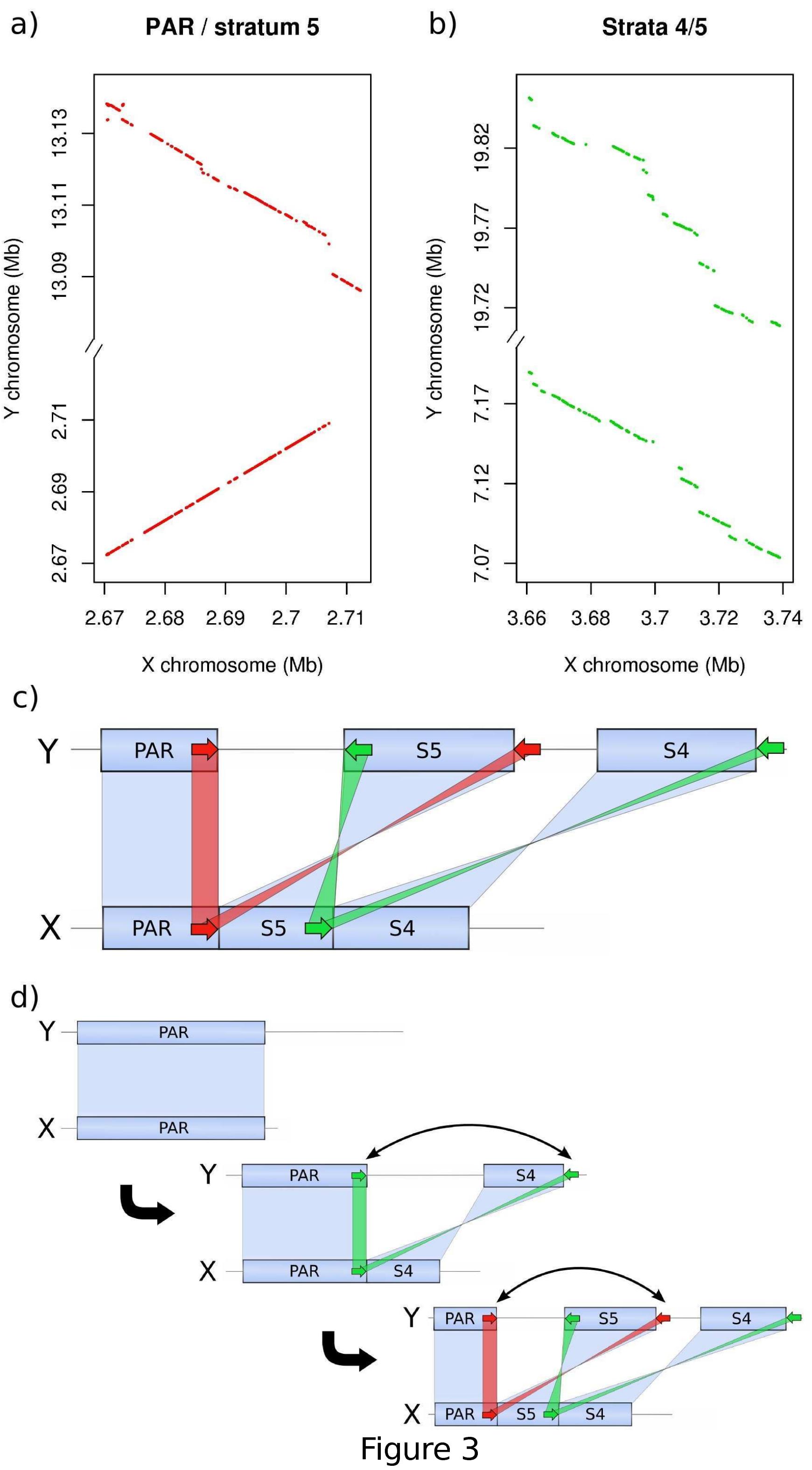
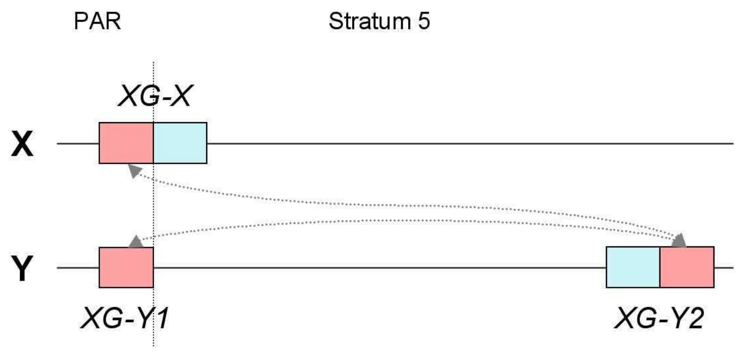

a)
PAR / stratum 5
Y chromosome (Mb)
13.13
13.11
13.09
2.71
2.69
2.67
2.67
2.68
2.69
2.7
2.71
X chromosome (Mb)
b)
Strata 4/5
Y chromosome (Mb)
19.82
19.77
19.72
7.17
7.12
7.07
3.66
3.68
3.7
3.72
3.74
X chromosome (Mb)
c)
Y
PAR
S5
S4
X
PAR
S5
S4
d)
Y
PAR
X
PAR
Y
PAR
S4
X
PAR
S4
Y
PAR
S5
S4
X
PAR
S5
S4

Figure 3



	XG-5'	***XG-3'***
d_S	0.060	0.091
d_N/d_S	0.019	1.01
% similarity total DNA	83.26	67.32

Figure 4

Tables

<u>Table 1</u>: **List of markers on X and Y chromosomes in humans in this study.**

Marker name[a]	**X position** [b]	**Y position** [b]	**X order**[c]	**Y order**[c]	**Strata**	**Refs**[d]
PAR	0-2709520	0-2709520	0	0	PAR	Lahn & Page (1999)
GYG,ARSD*,ARSE*, ARSF**,ADLICAN*** (1)	2672359-3346731	12492110-13139179	1	-6	5	*Lahn, Page (1999) **Skaletsky *et al.* (2003)
PRK (2)	3345018-3848954	7068601-7506089	2	-5	5	Lahn, Page 1999
Anonymous	3662755-3909738	19743211-19851471	3	-20	4	This ms
Anonymous (3)	4110549-4490406	17583377-18076812	4	19	4	Ross *et al.* (2005)
Anonymous (4)	4602689-5384111	16706206-17570219	5	-18	4	Ross *et al.* (2005)
around NLGN4 (5)	5384848-6313029	14981290-15805945	6	-15	4	Skaletsky *et al.* (2003)
Anonymous (6)	6594680-6624810	16664668-16691008	7	17	4	Ross *et al.* (2005)
around STS (7)	6625496-7448677	15807027-16376778	8	16	4	Lahn, Page
Anonymous (8)	7449397-7646086	14794314-14971778	9	14	4	Ross *et al.* (2005)
around VC (9)	7731889-7952770	14681748-14772569	10	-13	4	Skaletsky *et al.* (2003)
around KAL1 (10)	8388775-8678660	14456224-14455780	11	-12	4	Lahn, Page (1999)
TBL1 (11)	9367582-9694004	6818075-7040054	12	4	3 (or 4) [e]	Skaletsky *et al.* (2003)
APXL	9803943-9836714	13139984-13177590	13	7	3 (or 4) [e]	Skaletsky *et al.* (2003)
Anonymous	9925052-10026743	2935524-6736276	14	2	3 (or 4) [e]	This ms
AMEL (12)	11221454-11228802	6756180-6804332	15	-3	3	Lahn, Page

						(1999)
TMSB4	12893995-12914689	14259652-14336452	16	11	3	Lahn, Page (1999)
TXNLG	16713573-16773411	20187740-20234258	17	22	3	Skaletsky *et al.* (2003)
EIF1A	20052557-20069887	21146999-21164428	18	-23	3	Lahn, Page (1999)
ZF	24071318-24144376	2855296-2922379	19	1	3	Lahn, Page (1999)
MAP3 / TAB3	30755480-30819301	13771944-13828537	20	9	3	This ms
BCoR	39795364-39917376	20076630-20184596	21	21	3	Skaletsky *et al.* (2003)
CRSP2P-CASK	40392502-41667660	13240309-13592325	22	8	3	Lahn, Page (1999)
UT	44617701-44856791	13869035-14101947	23	-10	3	Lahn, Page (1999)

[a] in brackets are indicated marker numbers in Ross et al. (2005).

[b] positions on X and Y from NCBI version 36, hg18.

[c] order on the X and Y chromosomes of each marker is indicated.

[d] reference mentioning the markers for the first time is shown. The 3 new markers that we found are mentioned.

[e] see section on gene conversion between *XG* copies

Table 2: **Analysis of all the optimal scenarios for X-Y rearrangements.**

Classes of equivalence	**Curr. 3 strata**	**Alt. 3 strata**	**2 strata**	**1 stratum**	**No strata**	**Total**
1	420	0	2520	0	7140	10080
2	0	0	0	1260	8820	10080
3	0	0	0	1260	8820	10080
4	0	120	0	0	216	336
5	0	0	0	0	336	336
6	0	0	0	0	840	840
Total	420	120	2520	2520	26172	31752

The 12 markers from Ross *et al.* (2005) have been used (see also Table 1). Classes of equivalence group scenarios with the same inversions but in different orders (Braga *et al.* 2008). Curr. 3 strata are the strata defined by Ross *et al.* (2005). Alt. 3 strata are alternative strata with the following definition: stratum 5 = {1,2,3}, stratum 4 = {4,…,10} and stratum 3 = {11,12}. 2 strata: {1,2} and {3,…,12}. One stratum: {1,…,12}. Sub-totals and grand total are indicated. All the classes of equivalence are described in the supplementary data.

Table 3: **Comparison of free and strata constrained simulations and true X-Y sequences.**

	#tot_ scen	**#strat_ scen**	**#strat_scen / #tot_scen**	**#tot_ class**	**#strat_class**	**#strat_class/ #tot_class**
Observed XY	31752	420	0.0132	6	1	0.167
Free simulations	36100	12	0.0004	173	0.323	0.0019
Strata-constrained simulations	41700	406	0.0135	191	6.91	0.0544
p-values free vs. strata-constrained	$< 10^{-5}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-3}$	$< 10^{-16}$	$< 10^{-16}$

Values are median of the distribution of the different parameters. Parameters are # tot_scen = total number of optimal scenarios, # strata_scen = number of scenarios consistent with the 3 currently strata 3,4,5 , # tot_class = total number of classes of equivalence, # strata_class = number of classes of equivalence consistent with the 3 currently strata 3,4,5. Statistical tests are non-parametric tests (Wilcoxon) for comparing medians of two distributions. Observed values for XY are from Table 2.

Supplementary material

1) <u>Classes of equivalence found for X-Y chromosomes (12 markers analysis)</u>

Classes number	**Definition**
1	{11,12}{3,...,12}{11}{8}{5,6}{3}{1,2,11,12}{5,7}
2	{1,...,12}{11}{1,2,3}{4,...,10}{8}{5,6}{1,2,4,...,10}{5,7}
3	{1,...,12}{1,...,11}{3,...,11}{8}{5,6}{3}{1,2,11}{5,7}
4	{11,12}{8}{5,6}{4,...,10,12}{5,7}{1,2,3,12}{1,...,11}{1,2,4,...,10}
5	{1,...,11}{8}{5,6}{1,2,3,12}{5,7}{4,...,10,12}{11,12}{1,2,4,...,10}
6	{3,...,11}{4,...,10}{8}{5,6}{3,12}{5,7}{4,...,10,12}{1,2,11,12}

Classes of equivalence group optimal scenarios with the same inversions in different order. Table shows the 6 classes found for the X-Y data with 12 markers. In this table, groups of genes in brackets indicate breakpoints where inversions occur.

PS: the marker numbers are as in Ross *et al.* (2005)

2) Simulations with different numbers of inversion (*d*) values and subsequent analysis

Distance 4 :

	# tot_scen	#strata_scen	#strata_scen / #tot_scen	#tot_class	#strata_class	#strata_class / #tot_class
Free simulations	14	0.003	0.0003	1.44	0.002	0.001
Strata-constrained simulations	12	1.89	0.22	1.58	1	0.71
p-values	$< 10^{-7}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-7}$	$< 10^{-16}$	$< 10^{-16}$

1000 simulated sequences for free simulations and 585 for strata-constrained.

Distance 6 :

	# tot_scen	#strata_scen	#strata_scen / #tot_scen	#tot_class	#strata_class	#strata_class / #tot_class
Free simulations	400	0.007	0.0002	8.5	0.11	0.001
Strata-constrained simulations	450	16.6	0.05	9.1	1.31	0.22
p-values	$< 10^{-4}$	$< 10^{-16}$	$< 10^{-16}$	0.012	$< 10^{-16}$	$< 10^{-16}$

1000 simulated sequences for each type of simulations.

Distance 10 :

	#tot_scen	#strata_scen	#strata_scen/ #tot_scen	#tot_class	#strata_class	#strata_class / #tot_class
Free simulations	7062279	2337	0.0003	9262	12	0.001
Strata-constrained simulations	7750511	25907	0.004	9685	119	0.014
p-values	0.033	$< 10^{-16}$	$< 10^{-16}$	0.28	$< 10^{-16}$	$< 10^{-16}$

400 simulated sequences for each type of simulations.

L'ESPACE DE SOLUTIONS DU TRI PAR INVERSIONS ET SON UTILISATION DANS L'ANALYSE DE RÉARRANGEMENTS DE GÉNOMES

Resumé: Le calcul de la distance d'inversion et la recherche des séquences optimales d'inversions pour transformer un génome dans un autre quand les duplications de gènes ne sont pas acceptées sont des outils algorithmiques très utiles pour l'analyse de scénarios d'évolution réels. Néanmoins, le nombre de séquences optimales différentes est très grand. Avec un modèle proposé antérieurement pour regrouper des sous-ensembles de séquences optimales dans des classes d'équivalence, on a développé un algorithme qui génère une séquence optimale par classe d'équivalence, sans énumérer toutes les séquences, ce qui permet de réduire la taille de l'ensemble à traiter. On propose aussi l'utilisation de différentes contraintes biologiques, comme les intervalles communs détectés initialement et progressivement, pour réduire le nombre de classes, et on montre comment utiliser ces methodes pour analyser des cas réels d'évolution. En particulier, on analyse le scénario évolutif de la bactérie *Rickettsia* et des chromosomes sexuels X et Y chez l'être humain. Par rapport aux résultats des études précédentes, qui se sont basées sur une seule séquence optimale, on obtient une meilleure caractérisation de ces scénarios évolutifs. Tous les algorithmes qu'on a développés sont implémentés en java, integrés à BAOBABLUNA, un logiciel qui contient des outils pour manipuler des génomes et des inversions. Le téléchargement et le tutoriel de BAOBABLUNA sont disponibles en ligne.

Mots-clés: Évolution ; réarrangements de génomes ; algorithmes ; tri par inversions

EXPLORING THE SOLUTION SPACE OF SORTING BY REVERSALS WHEN ANALYZING GENOME REARRANGEMENTS

Abstract: Calculating the reversal distance and searching for optimal sequences of reversals to transform a genome into another when gene duplications are not allowed are useful algorithmic tools to analyse real evolutionary scenarios. However, the number of sorting sequences is usually huge. Using a model previously proposed to group the sorting sequences into classes of equivalence, we developed an algorithm to direct generate the classes without enumerating all sequences, reducing thus the size of the set to be handled. We then propose the use of different biological constraints, such as the common intervals initially and progressively detected, to reduce the universe of sequences and classes, and show how to apply these methods to analyze real cases in evolution. In particular, we analyzed the evolution of the *Rickettsia* bacterium, and of the sexual chromosomes X and Y in human. We obtain a better characterization of the evolutionary scenarios of these genomes, with respect to the results of previous studies, that were based on a single sorting sequence. All the algorithms developed in this work are implemented, integrated to BAOBABLUNA, a java framework to deal with genomes and reversals. Download and tutorial for BAOBABLUNA are available on-line.

Keywords: Evolution ; genome rearrangements ; algorithms ; sorting by reversals

Discipline: Bioinformatique

Intitulé et adresse du laboratoire: Laboratoire de Biométrie et Biologie Évolutive
UMR CNRS 5558 - Bâtiment Gregor Mendel - Université Claude Bernard LYON I
43, boulevard du 11 novembre 1918 - 69622 Villeurbanne cedex - FRANCE