Segment-based distances and similarities in genomic sequences

Nadia Pisanti

April 2002
Thesis supervisors:
Prof. Fabrizio Luccio, Dr. Marie-France Sagot
Abstract

We address some problems arising in the analysis and representation of genomic data. The thesis is divided in two parts. Part I contains studies on large scale mutations of genomes with particular attention to gene duplications. We prove some results on genomic syntenic distance, and investigate some properties of families of paralogous genes. These latter can also form a ground for genome comparisons. Part II contains some studies on motif extraction which role in molecular biology is crucial for building genome maps and for many other applications at structures and similarities in biological sequences.
Beautiful things are either forbidden, or they are unhealthy, or they are immoral, or they are $NP$-complete!
Acknowledgments

First of all I want to thank my supervisors, Fabrizio Luccio (University of Pisa) and Marie-France Sagot (INRIA Rhône Alpes). The continuous support and trust of Fabrizio gave me the energy to explore this beautiful and relatively new field where I have been so lucky to have the opportunity to work with one of its leaders as Marie-France is. Without them this thesis would simply not exist.
It was a pleasure to work also with all the other co-authors and collaborators Anna Bernasconi, Valentina Ciriani, Paolo Ferragina, Antonio Frangioni, Roberto Grossi, Roberto Marangoni, Laurent Marsan, Alain Viari, Maxime Crochemore, Guglielmo Tamburrini, Raquel Tavares, Anne Vanet, Marina Zelwer, and Marcello Buiatti.
I feel grateful to Linda Pagli for all our nice chats about work and life and for her advise on many things. Also Roberto Grossi and Paolo Ferragina deserve my gratitude for their constant availability for questions and chats, as well as Valentina Ciriani with her immense patience.
I want to thank my room-mates at the Department in Pisa: Valentina, Paolo, Vincenzo, Giuseppe and Anna for being so patient with my noisy typing, and for creating such a nice working environment. It was very helpful to share joy and difficulties with them and with Marco Aldinucci, Laurent Marsan, Andrea Bracci, Francesca Scozzari and Alessandra Raffaetà and all the others (the list is too long).
I want to thank all the people that welcomed in the many places I had the opportunity to visit during my Ph.D: the group of theoretical computer science in Leiden (the Netherlands) led by G.Rozenberg where I wrote a big part of this thesis, the Gaspard Monge institute at Marne-la-Vallée (France) where everything started, the Laboratoire d’Algorithmique Combinatoire at the Institut Pasteur (France), the Atelier de Bioinformatique at Paris 6 (the best coffee out of Italy!), the Computer and Automation Research Institute of the Hungarian Academy of Science in Budapest, the University of Opava (Czech Republic), the Laboratoire de Biométrie et Biologie Evolutive of the University of Lyon 1 (France), and the INRIA Rhône Alpes in Grenoble (France).
I owe a lot to my friends and my family that make it so nice all the times I come back to my hometown, and make it so hard to leave again right after.

Finally, a special thought to Maurice, to whom my PhD is dedicated, with the wish that his thesis will follow soon.
Contents

Preface 1

1 Introduction 3
   1.1 Computational biology and bioinformatics ............... 3
   1.2 A huge amount of data: hunting the information .......... 4
   1.3 A brief description of some of the fields of computational biology . . 5
   1.4 Some notions of molecular biology ........................ 7
      1.4.1 DNA and proteins ................................... 7
      1.4.2 The structure of genomes ............................ 8
      1.4.3 Paralogous and orthologous genes .................... 9

I Some studies on segment-based evolution events 11

2 Introduction to part I 13
   2.1 Segment-based events ...................................... 13
   2.2 Sorting algorithms on genomes ............................... 16
   2.3 Gene duplication and its role in evolution ................. 18
   2.4 Overview of part I ....................................... 19

3 On Syntenic Distance 23
   3.1 Introduction ............................................... 23
   3.2 Definitions and previous results ........................... 25
   3.3 Syntenic diameters ....................................... 27
      3.3.1 Syntenic diameter ................................... 27
      3.3.2 Bidimensional syntenic diameter ...................... 31
   3.4 The multi-component case .................................. 32
      3.4.1 General observations ................................ 32
      3.4.2 Optimizing in a special case of two components by performing inter-component moves ......................... 33
      3.4.3 Finding an optimal components pairing is $\mathcal{NP}$-hard .... 34
   3.5 Nested synteny ............................................ 36
      3.5.1 General observations ................................. 36
3.5.2 Exact nested synteny ........................................... 37
3.5.3 Linear nested synteny ........................................... 41
3.5.4 General nested synteny ........................................... 41
3.6 An inclusion theorem for the general case ......................... 46
3.7 Further work ....................................................... 48

4 PaTre: a tool for building paralogy trees 49
4.1 Introduction ....................................................... 49
4.2 Preliminaries ..................................................... 50
   4.2.1 The problem: from paralogous genes to paralogy trees .... 50
   4.2.2 The transformation “distance” ................................ 51
   4.2.3 The lightest spanning arborescence problem ............... 53
4.3 Our proposal: the PaTre tool .................................... 53
4.4 Experimental results based on synthetic data ..................... 55
   4.4.1 The simulator .................................................. 55
   4.4.2 Validating the simulator ..................................... 56
   4.4.3 Results on synthetic data ..................................... 57
4.5 Preliminary experimental results on real data .................... 57
   4.5.1 Application to histone family in *Saccharomyces cerevisiae* 58
   4.5.2 Some studies on a family of *Mycoplasma pneumoniae* .... 58
   4.5.3 The case of a higher organism: the plant *Arabidopsis thaliana* 59
4.6 Further work ..................................................... 62

5 Clustering families of paralogous genes by using random graphs 65
5.1 Introduction ..................................................... 65
5.2 Generalities of the problem ..................................... 66
   5.2.1 The problem .................................................. 66
5.3 Some previous work .............................................. 67
5.4 The graph theory approach ...................................... 68
   5.4.1 The Levenshtein distance .................................... 68
   5.4.2 The graph $\mathcal{G}_f$ ....................................... 69
   5.4.3 Comparing with random graphs ............................... 72
5.5 Searching for a good random graph model ......................... 74
   5.5.1 The model $\mathcal{G}(n,m)$ .................................... 74
   5.5.2 Ultrametrics and molecular evolution ........................ 77
   5.5.3 The model $\mathcal{G}(n,m,f)$ .................................... 78
   5.5.4 The model $\mathcal{G}(n,m,\alpha + 1)$ ........................... 81
5.6 Applications to a bacterium .................................... 83
   5.6.1 Description of the organism ................................. 83
   5.6.2 Comparing histograms ....................................... 83
   5.6.3 Back to biology ............................................. 89
5.7 Further work ..................................................... 93
**II Motif extraction**

6 *Introduction to part II*

6.1 Motif extraction ........................................ 97
6.2 Some previous work ........................................ 98
6.3 Preliminaries: suffix trees .................................. 100
6.4 Overview of part II ........................................ 101

7 *Tiling motifs*

7.1 Preliminary definitions ....................................... 104
  7.1.1 Motifs .................................................. 104
  7.1.2 Tiling motifs ........................................... 105
7.2 Bound on the number of tiling motifs ......................... 106
7.3 Finding tiling motifs ........................................ 107
7.4 Some properties of tiling motifs ............................... 111
  7.4.1 Representing motifs on the suffix tree of s .................. 111
  7.4.2 Some properties of maximal and redundant motifs .............. 112
7.5 Characterization of solid parts of tiling motifs .............. 114
7.6 Hints for a new algorithm: the operator $\otimes_q$ .............. 116
7.7 Further work ............................................. 118

8 *Improving the extraction of structured motifs* .............. 119

8.1 Introduction ................................................ 119
8.2 Structured motif extraction .................................. 120
8.3 Using maximal extensibility of factors to speed up the computation .. 122
  8.3.1 The idea ............................................... 122
  8.3.2 Storing the extensibility information ..................... 123
  8.3.3 Fetching and using extensibility information ............... 125
  8.3.4 Minimizing the amount of ghost nodes ..................... 127
  8.3.5 The case of structured motifs ................................ 128
  8.3.6 A further optimization .................................. 130
8.4 Experimental results ......................................... 132
  8.4.1 Preliminary results ..................................... 132
  8.4.2 A first implementation: results .......................... 133
8.5 Conclusions and further work .................................. 134

9 *Conclusions* .................................................. 137

9.1 Concluding remarks .......................................... 137
9.2 Future work ................................................ 138
  9.2.1 Syntenic distance ......................................... 138
  9.2.2 PaTrE: a tool for building paralogy trees .................. 139
  9.2.3 Clustering gene families by using random graphs ............ 139
  9.2.4 Tiling motifs ........................................... 140
9.2.5 Improving structured motif extraction . . . . . . . . . . . . 140

Bibliography 143

Index 153
List of Figures

1  Structure of the thesis.................................  2

1.1  Number of database entries in EMBL until October 2001........  4
1.2  Number of nucleotide stored in the EMBL database until October 2001.  5
1.3  Number of database entries in SwissProt. Note that the size is plotted per release and not per year.........................  8

3.1  Tree representation of a nested instance of the syteny problem...  38

4.1  A script that transforms a sequence $S$ into a sequence $T$, and its description as in [VDR99]..............................  52
4.2  Example of a simulator output. The labels assigned to the nodes show the order of generation of the corresponding genes........  56
4.3  Two paralogy trees for the histone family of *Saccharomyces cerevisiae*. The labels on the edges represent the transformation distances...  58
4.4  The paralogy tree output by PaTre for the 8 genes of the heat shock protein family of *Mycoplasma pneumoniae*..................  59
4.5  The paralogy tree output by PaTre for the SHAGGY/GSK-3 family in *Arabidopsis thaliana*..............................................  61
4.6  The second best solution for the SHAGGY/GSK-3 family in *Arabidopsis thaliana*. ....................................................  62

5.1  Distance distributions in *Mycoplasma genitalium*; myge is the distribution of the organism, snyge is the one obtained from the shuffling within genes, and xmyge is the one from the shuffling of the complete genome. .......................................  70
5.2  Distance distributions in *Mycoplasma pneumoniae*; mypn is the distribution of the organism, smypn is the one obtained from the shuffling within genes, and xmypn is the one from the shuffling of the complete genome.......................................................  71
5.3  Number of edges in Levenshtein graphs for all possible values of the threshold from 0.00 to 1.00 time $D$. The curve myge refers to the pure biological data, while snyge refers to the shuffled sequences........  72


5.4 Degree distributions for a random graph of the model $\mathcal{G}(468, m=133)$ (curve gnm) and for a shuffled *Mycoplasma genitalium* (curve xmyge) with a threshold $0.74$. Both the graphs have 133 edges. The curve bino shows the binomial distribution of parameters $n-1 = 467$ and $p=133/N$. The histograms represent, for all $k$, the number of nodes of degree $k$. ............................................ 75

5.5 Degree distributions for a random graph of the model $\mathcal{G}(468, m = 77323)$ (curve gnm) and for a shuffled *Mycoplasma genitalium* (curve xmyge) with a threshold $\Gamma = 0.81$. They both have 77323 edges. The curve bino shows the binomial distribution of parameters $n-1 = 467$ and $p = 77323/N$. ............................................ 76

5.6 Degree distributions. The curve xmyge refers to the graph built for the shuffled sequences (1287 edges), while $f_2$ and $f_3$ are the distributions of the corresponding graphs in $\mathcal{G}(n, m, 2)$ and in $\mathcal{G}(n, m, 3)$, respectively. ............................................ 80

5.7 Degree distributions. The curve xmyge refers to the graph built for the shuffled sequences (77323 edges), while $f_2$ and $f_3$ are the distributions of graphs in $\mathcal{G}(n, m, 2)$ and in $\mathcal{G}(n, m, 3)$. ................................. 81

5.8 Degree distributions. The curve xmyge refers to the graph built for the shuffled sequences (101431 edges), while $f_2$ and $f_3$ are the distributions of graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$. ................................. 82

5.9 Path distributions. The curve xmyge refers to the graph built for the shuffled sequences (1287 edges), while $f_2$ and $f_3$ are the distributions of, respectively, the corresponding graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$. ................................. 83

5.10 Path distributions. The curve xmyge refers to the graph built for the shuffled sequences (77323 edges), while $f_2$ and $f_3$ are the distributions of the corresponding graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$. ................................. 84

5.11 Path distributions. The curve xmyge refers to the graph built for the shuffled sequences (101431 edges), while $f_2$ and $f_3$ are the distributions of the graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$. ................................. 85

5.12 Degree and (length of) path distributions (respectively at the top and at the bottom). The curve xmyge refers to the graph built for the shuffled sequences (1287 edges), while $f_2$ and $f_1.5$ are the distributions of, respectively, the corresponding graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 1+0.5)$. ................................. 86

5.13 Degree and (length of) path distributions (respectively at the top and at the bottom). The curve xmyge refers to the graph built for the shuffled sequences (77323 edges), while $f_2$ and $f_1.5$ are the distributions of, respectively, the corresponding graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 1+0.5)$. ................................. 87
5.14 Degree and (length of) path distributions (respectively at the top and at the bottom). The curve xmyge refers to the graph built for the shuffled sequences (101431 edges), while f2 and f1.5 are the distributions of, respectively, the corresponding graphs in \( G(n, m, 2) \) and \( G(n, m, 1 + 0.5) \). \hfill 88

5.15 \( \chi^2 \) of degree histograms for all the thresholds (differences from degree distribution of the Levenshtein graph built from *Mycoplasma genitalium*). The curve xmyge refers to the shuffled genome, f1 is the one corresponding to random graphs of the basic model \( G(468, m) \), f2 corresponds to graphs in \( G(468, m, 2) \), and finally f1.5 corresponds to graphs of \( G(468, m, 1 + 0.5) \). \hfill 89

6.1 The suffix tree for the string \( s = abbabc \). \hfill 100

6.2 The generalized suffix tree for the strings \( s_1 = abbabc \) and \( s_2 = cbabb \). The two strings have endmarkers \$1 \) and \$2 \) respectively. \hfill 101

7.1 The suffix tree for the sequence \( s = baabbbaa \). Bigger unfilled nodes are those of the compact suffix tree \( T \), and the smaller black ones are those that are the nodes of \( U \) only, and that actually do not branch. Nodes of the compact suffix tree that have crosses on them, are those from which edges labeled with ‘$’ should leave (and are omitted). The number labeling each node \( v \) is \( n_v \). \hfill 111

7.2 Details of the suffix tree for a sequence where fragment \( f' \) always follows fragment \( f \). \hfill 115

8.1 Area \( T_{(GCCCA,1,d)} \) where the second box has to be searched once the first one is \( GCCCA \). Nodes \( x \) and \( y \) are the occurrences of \( GCCCA \). The subtrees rooted at nodes that are \( d \) levels below \( x \) and \( y \) and that are their descendants compose \( T_{(GCCCA,1,d)} \). \hfill 130

8.2 Time costs before and after the improvement presented in this chapter. The horizontal axis shows the size each one of the 20 input sequences, and the vertical axis is the time cost expressed in minutes. \hfill 134
Preface

This thesis aims to address some problems that belong to the most recent phase of analysis and representation of genomic data that we introduce in chapter 1. Figure 1 shows the structure of the text which is divided into two parts. Part I contains some studies on large scale mutations of genomes, and part II contains some investigations in the field of motif extraction whose role is crucial for building genome maps, and many other applications at structures and similarities in biological sequences. Each part contains its own specific introduction (respectively chapter 2 and chapter 6), which briefly describes the state of the art of the field, and which gives further motivations for each addressed problem. Moreover, these two chapters also give some possible necessary preliminary definitions specific of their part of the thesis.

Chapters 3, 4, 5, 7, and 8 are self-contained for readers that know the research field (otherwise, reading chapter 1 and the introduction of the specific part is advised). Each chapter begins with its own introduction where possible preliminaries and motivations specific to that chapter are given. All these chapters end with a section that describes possible further work. Finally, a conclusion (chapter 9) common to both part I and part II ends the thesis.
Figure 1: Structure of the thesis.
Chapter 1

Introduction

1.1 Computational biology and bioinformatics

Donald Knuth once wrote that biology easily has 500 years of exciting problems to work on. Indeed, although a lot has been done since the recent discovery of the structure of DNA, one can say that the understanding of the mechanisms underlying molecular biology and genetics is just at the beginning. Before 1900, biological science was mainly a descriptive discipline. Only with Mendel’s experiments on peas (Mendel was a physicist), mathematics entered biology ([Bui00]). Once again the interaction between different disciplines, enabled the discovery of the double helix by Watson and Crick, that gave birth to molecular biology. Since then, a true revolution made molecular biology crucially interesting to both natural and medical science. In the last twenty years, we have witnessed the new phase of genome sequencing, where the DNA sequence of a gene (or an entire genomes) can be scanned and then transcribed into electronic form. This has brought about an impressive growth of the amount of biological data available in the form of sequences (figure 1.1 shows the number of entries in one of the main DNA sequence databases). In order to manage and study this data, new tools are required. To this purpose, both the maintenance and the study of bio-sequences directly involve computer science and its applications: managing data bases of such sizes, analyzing huge amount of data, and inferring properties of these sequences, clearly requires suitable and efficient methods. The activity which involves the design of combinatorial problems motivated by molecular biology and the search for their algorithmic solutions is nowadays called computational biology. The more practical counterpart of this is mostly referred to as bioinformatics.
1.2 A huge amount of data: hunting the information

The main European DNA sequence database, maintained by the EMBL [SBdB+01] (European Molecular Biology Laboratory), as well as the other genomic databases, has doubled its size every year in the last two decades. Figure 1.1 gives the number of database entries since 1982, and figure 1.2 shows the corresponding amount in gigas of nucleotides (nucleotides are the chemical elements composing a DNA sequence). We have now reached a point in which the bottle-neck in research concerns the analysis of the data rather than its acquisition (nevertheless, such data acquisition does not seem to stop growing). Transforming this huge amount of data into scientific knowledge is the new challenge of molecular biologists. This challenge has recently entered a new phase. In fact, the availability of more and more complete genome sequences asks for (and offers) new ways of analysing the data. Pairwise sequence comparisons does not suffice to this purpose, and new larger scale representations and analyses are becoming increasingly important. The comparison of genomes maps is among these ([OCS+96, TKL97]): one of the first public “issues” of the complete human genome sequence has been given by means of a comparison with the one of mice. Such a view is also useful in order to guess possible large scale mutation events since the most recent common ancestor organism. The study
of large scale mutations rather than point level mutations (see chapter 2) is another approach that recently grew in importance. In order to build genome maps, it is necessary to detect specific sites that are repeated in the sequence or common to other sequences. To this purpose, it is necessary to conceive efficient tools that perform specific types of text mining.

1.3 A brief description of some of the fields of computational biology

Especially at the beginning of computational biology, some of the main results obtained in the field were adaptations of previously existing results from other areas of computer science. Examples of these are the search for repetitions and sequence comparison that share basic ideas with text compression and string matching theory. Next to this, there are entire subjects within computational biology that contain completely new combinatorial problems suggested by molecular biology [Wat95, SM97, Gus97, Pev00, Sag00]. We briefly mention here a few areas of computational biology that we will — directly or indirectly — meet in this thesis.

One of the first fields explored at the beginning of the sequencing phase con-
cerns pairwise sequence alignment. It includes several variants of techniques that use dynamic programming to measure the similarity between sequences. It is one of the most studied research areas in computational biology so far [SK83], and it led to crucial tools for all kinds of studies based on sequence comparisons. This has enabled the creation of softwares for sequences alignment, among which the most known are BLAST (freely available at http://www.ncbi.nlm.nih.gov/BLAST/) and FASTA (http://www.ebi.ac.uk/fasta33/), that are extensively used by molecular biologists.

The area of genome rearrangement, includes problems concerning genome evolution and comparison, such as computing distances among genomes and counting large scale mutations that are assumed to have caused divergence between genomes. We will go into further details on genome rearrangement in section 2.1, as the whole part II concerns this field.

A field of computational biology that has opened new issues in one of the most discussed problems of theoretical biology is phylogeny reconstruction. This includes all methods that try to infer information about speciation and evolutionary events, and to build trees that represent such relationships. Basically, there are two kinds of methods that differ from the type of data from which one tries to infer the phylogeny: character data and distance data. In the former, information about organisms (or genes that represent them) is given in terms of suitably chosen characters that may assume two or more values each. In this case, the speciation are associated with changes in these values. The latter starts with a notion of distance computed pairwise on the organisms (between sequences representing them or based on representations alternative to sequences), and then the phylogeny is inferred from the matrix that contains all these distances. From the computational complexity point of view, the resulting problem is either solvable in polynomial time or difficult, depending on the number and type of values the characters may take in the first case, and on metric properties of the distance adopted in the second case. Although interesting and useful investigations and hypotheses can be done, this problem is will always remain open from the biologist point of view, because obviously there is no unquestionable way to verify the obtained results.

The areas mentioned so far are among those that most involve theoretical computer science and mathematics, requiring algorithms and combinatorics as well as statistics and probability theory. As already mentioned, we only listed those to which we will refer in the next chapters. Within bioinformatics, there are also many other problems for other specific applications. Listing them all is beyond the purposes of this thesis.
1.4 Some notions of molecular biology

In this section, we give some basic notions of molecular biology that will be useful specifically for understanding the problems addressed in this thesis. Since in molecular biology exceptions to general rules are very frequent, and since we will be concerned with only some aspects and steps of biological processes, what follows is necessarily incomplete, and sometimes not completely faithful to the truth as far as special cases (which we ignore) are concerned. In fact, writing a complete and correct summary of molecular biology in a few pages is again beyond the purposes of this thesis (and probably impossible). For more detailed information on anything within molecular biology, one may consult [WHR+87].

1.4.1 DNA and proteins

Abstracting from chemical structures, biological sequences consist of possibly very long words on a certain alphabet. For DNA sequences, the alphabet is $\Sigma = \{A, C, G, T\}$, where the four letters denote the four nucleotides that compose DNA molecules. The A stands for Adenine, C for Cytosine, G for Guanine and T for Thymine. These are also called DNA bases. In the case of protein sequences, the alphabet contains twenty letters representing the amino acids of which proteins are made in almost all organisms. Since such sequences can be very long, and it is often necessary to work with many of them at once, computer scientists’ framework of asymptotic complexity is here well motivated. Figure 1.2 shows the same data as figure 1.1, but in terms of number of nucleotides instead of database entries, which can give an idea of the average length of the sequences. One of the most used protein databases is SwissProt whose size for the last releases is shown in figure 1.4.1.

All of us have heard that genetic information resides in DNA. How does this information make the organisms be the way they are is a complex and still unclear process that starts with the synthesis of proteins. Indeed, proteins are the fundamental agents of life, as their properties and their interactions determine the way organisms are. It is DNA which contains all the necessary information to synthesize proteins. A gene contains the information for the synthesis of a single protein\footnote{Actually, there are also genes that encode something different from a protein, i.e., they contain information that is necessary for the event itself of the synthesis of a protein (e.g. tRNA).}. This is the reason for scientists to use the expression genetic code: triplets of nucleotides in the DNA sequence, named codon, codes for one of the 20 amino acids. There are $64 = 4^3$ possible triplets of nucleotides while there are only 20 amino acids. This unavoidable redundancy (for obvious reasons two nucleotides would not have been enough, as $4^2 = 16$ is lower than the number of distinct amino acids) implies that there are amino acids which are encoded by several different codons, and there are codons that encode other information such as the beginning and the end of the gene sequence. The first step of protein synthesis is caused by an enzyme which activates the transcription of a gene. Its DNA sequence is thus transcribed into an mRNA
(messenger RNA). This is identical to the DNA molecule except that nucleotide $T$
 is replaced by $U$ (Uracil). The function of the mRNA is indeed to carry the information
of the DNA (which is inside the nucleus of the cell) outside of the nucleus, that
is in the cytoplasm where the ribosomes are. The ribosomes (partly made of RNA
as well, that is of rRNA) place themselves along the mRNA molecule and build the
chain of amino acids that are encoded by the DNA starting sequence. This second
phase is called translation, and in fact the information here is translated into a new
language in a new alphabet (the one of the twenty amino acids). This is possible
thanks to a third type of RNA (called tRNA, that is transfer RNA) that carries the
amino acids with the anticodons that attach to corresponding triplets of nucleotide.
Finally, the amino acid chain is released. It consists of the linear (also known as
primary structure) of the newly synthesized protein. This linear structure will fold
into a secondary, and successively a tertiary final structure. There are actually sev-
eral variations and exceptions to the process we just described, but our purpose is
just to give a general view that enables to understand the analysis performed in the
next chapters.

1.4.2 The structure of genomes

Within DNA, genetic information is grouped into chromosomes, and in each chromo-
some there can be thousands of genes. Less complex organisms such as prokaryotes
consist of a single chromosome which is often arranged in a circular shape. More
complex organisms such as *eukaryotes* have their DNA divided in several chromosomes, and in general contain more genes. The protein synthesis as we described it in section 1.4.1, is actually what happens in eukaryotes. In *prokaryotes* the cell has no nucleus, and thus the whole process takes place in the cytoplasm. Therefore, the role of mRNA as messenger from the nucleus to the cytoplasm is basically for eukaryotes. Moreover, mRNA in eukaryotes has another important role, as it during the transcription of mRNA that only the coding part of the DNA sequence are selected. That is, the sequence encoding for a protein in eukaryotes is in general not contiguous, and a gene contains parts named *introns*, as well as some parts at the beginning and end of the sequence, that actually do not encode any amino acid fragment. In the process of transcription to the mRNA, the introns are eventually ignored and thus the encoding sequence becomes contiguous.

### 1.4.3 Paralogous and orthologous genes

As more genes (or entire genomes) are sequenced, it appears increasingly interesting to study the existence of evolutionarily related genes, both within a genome and between different genomes. Indeed, genes with similar sequences are likely to possess related biological functions. Mostly, genes that have very similar sequences are acknowledged as *homologous genes*. Homologous genes may be distinguished into *orthologs* and *paralogs*. The former indicates genes in different species that have evolved from a common ancestral gene by speciation. The latter are homologous genes in the same organism whose products perform related but not identical functions. Most probably, these are the results of gene duplications within a genome. The topology and amount of duplications in chromosomes are useful information for studying the evolution of organisms. In fact, according to an hypothesis made in the seventies ([Ohn70]), gene duplication is the predominant mechanism for the evolution of new gene functions: once redundant gene copies are available, mutations may accumulate in one of the copies, thus creating a new gene, while the other one(s) can go on carrying out the original function. This apparently crucial role of gene duplication motivates our special interest in this evolutionary event that we will investigate further in part I, and in particular in chapter 4 and chapter 5.
1. Introduction
Part I

Some studies on segment-based evolution events
Chapter 2

Introduction to part I

2.1 Segment-based events

Many studies on biosequences are nowadays heavily based on sequence comparison. One wants to know how much a certain sequence is similar to another, once having defined what similar means. Basically, similarity is measured by counting how many among a set of suitably defined mutations are necessary to transform one sequence into another. Basic choices for such operations are editing operations like deletion, insertion or substitution of a single base or amino acid. Counting point mutations is interesting when one wants to detect the degree of similarity between two relatively short sequences, for example two genes that might be evolutionarily related. Counting instead segment-based events is more suitable when whole genomes have to be compared (or just long sequences such as chromosomes in eukaryotes) in order to compute distances at a larger scale. In fact, in case of possible highly diverged sequences such as two different entire genomes, point mutations have occurred so much that they are very likely to have overlapped. Therefore, a sequence alignment analysis that counts point mutations could represent a distance which is no longer significant. Moreover, two sequences may present very high local similarities, but the order of the similar segments could be different on the two sequences (because such segments have moved during evolution). In this case, a sequence alignment involving the whole sequence would fail to align all the local common segments. On the other hand, segment-based distances do not require a high rate of similarity to give interesting information. The idea behind a segment-based distance is to capture the information that, for instance, the deletion of a whole block is a single event, and not a set of consecutive point level deletions. It is in fact proved that such events do happen in nature: entire segments of DNA may be deleted, duplicated, moved, reversed, copied, exchanged, etc. Such events are called genome rearrangements. They are the events considered when comparison are made at the genome level. Distances on such events are suitably defined, and algorithms for computing them are being designed. One of the most interesting use of distances
computed in this way is to infer phylogeny and properties of ancestral organisms (see for example [SLA+92]). This opens a whole different area of computational biology (see chapter 1) which is beyond the scope of this thesis. We start by giving a quick overview on genome rearrangement. To begin with, we list some of the segment-based events which are biologically pertinent. In section 2.2, we list a few results concerning the computational complexity of computing different variants of segment-based distances. Section 2.3 contains a deeper look at gene duplication, and finally section 2.4 gives a summary of the chapters of this part of the thesis.

Events that involve whole segments may take place within a gene, or they may involve more than one gene. In the latter case, the parts involved could even belong to different chromosomes. Depending on which event, which of these cases one wants to detect, and which kind of data is available, different types of rearrangement events can be considered. We start by listing events that involve a single sequence and whose result affects that sequence only.

**Deletion** A segment of the sequence disappears.

**Insertion** A segment is inserted somewhere in the sequence.

**Duplications** A segment of the sequence duplicates itself and the new copy may appear either next to the original (in this case one talks of a tandem duplication), or somewhere else in the sequence.

**Inversion** A segment reverses its orientation within the sequence. This event is also known in the literature as a reversal.

**Transposition** A segment moves somewhere else in the sequence.

**Block interchange** Two segments exchange their position. Note that transposition can be viewed as a special case of block interchange where the two blocks are consecutive.

**Example 1** As an example, we show some events applied to the simple sequence abcddefgh; for clarity, we underline the segments involved in the mutation.

\[
\begin{align*}
\underline{abcde}fgh & \rightarrow \underline{a}abc\underline{d}efgh \\
abcd\underline{ef}gh & \rightarrow ab\underline{cde}ab\underline{f}gh \\
\end{align*}
\]

are two duplications of ab, the first being a tandem.

\[
\underline{abcde}fgh \rightarrow \underline{a}bc\underline{f}\underline{d}gh
\]

is a transposition of the segment cd. Finally, we have an inversion of the segment cde:

\[
\underline{abcd}efgh \rightarrow ab\underline{cde}fgh.
\]

Among the events listed so far, it has been observed that reversals seem to be twice as frequent as the others ([BKS96]). The operations listed so far are defined on
single sequences. Therefore, a basic problem that arises is to compute the distance between two sequences based on such events. This distance is defined as the minimum number one or more among these segment-based operations that are necessary to transform one sequence into the other. This problem is often defined as a sorting problem in the genome rearrangement literature. For example, sorting by reversal is the problem of computing the reversal distance, that is the minimum number of reversal operations that change a sequence into another. In section 2.2 we give a brief overview of what has been solved among these problems.

As we have introduced in chapter 1, some organisms have a genome that is actually organized in several chromosomes. In this case, the genome can be represented by more than one sequence. In this view, one is interested in defining operations that are applied to (or that give as a result) more than just a single sequence, and that correspond again to real biological events:

**Concatenation** Two separate sequences become one by concatenating them together.  
**Fission** A sequence can be split into two parts creating two new sequences.  
**Reciprocal translocation** Two sequences may exchange a terminal part.

In other cases, the information about the relative order of the genes is not known. That is, it is known that a gene belongs to a given chromosome, but its orientation is not known, and nor the position. In this case, representing the genome as a sequence of label becomes impossible. For this reason, some segment-based events can be represented not just by one or more sequences, but rather by a set of objects. For example, a genome can be seen as a certain number of unordered sets, each one corresponding to a chromosome and containing elements that are the genes belonging to that chromosome. Representing a chromosome as a set of genes and not as a sequence is useful when one does not care about the order in which genes appear, or when this information is missing, as only their presence is taken into account. This is called *synteny* ([SN96, FNS96]). Two genes are said to be *syntenic* if they belong to the same chromosome. When only the synteny of genes is known (and not for example the relative positions of genes within the chromosome), or when only this kind of information is sought, then a genome can be represented as just a set of sets. Each chromosome is then a set containing the unordered list of genes that belong to that chromosome in that organism. In the case of synteny, the operations that can be considered are the following:

**Fusion** Two chromosomes fuse into one whose contents is the union of the two initial chromosomes.  
**Fission** A chromosome splits into two.  
**Translocation** Two chromosomes exchange part of their contents.
These are somehow events at an even larger scale than those listed above because the representation and the operations can also be seen as being at a higher level than those previously considered.

2.2 Sorting algorithms on genomes

In the genome rearrangement literature, the problem of finding the distance between two genomes in terms of the events described in section 2.1, is known as a sorting problem. For example, given two genomes and the reversal operation, the problem of finding the reversal distance between two genomes is known as sorting by reversal. The two input genomes are given as (one or more) sequences (or sets) of labels that represent genes. The problem then is to sort one of them to make it become equal to the other one. The same label (that usually represents a gene, but in general it may represent a common fragment) appearing in the two different genomes means that a certain gene in one genome corresponds (that is, is the ortholog) to a gene in the other genome. This implies a preprocessing where the sequence similarity, or orthology, of the genes in the two genomes is detected. A way to find such similar sequences is to use the common motif search methods such as those we describe in part II. Once the input genomes are given, the sorting problem typically corresponds to finding the minimum number of operations (among a given set of allowed ones) that are necessary to transform one genome into the other. The choice of the minimum guarantees uniqueness and is done according to a principle broadly used in biology and known as the parsimony principle. It assumes that in order to evolve from one configuration to another one, nature has more or less taken the shortest way (phylogeny reconstruction and sequence alignment also assume this principle). Depending on which operations are taken into account, sorting problems may be tractable or not. We now list some results on tractability and approximations. More can be found in [Chr98, Pev00, SM97], since what we indicate here is just a brief summary.

The general problem of sorting by reversals was proved to be NP-hard in [Cap97] when already several approximations had been suggested by other authors, reaching the ratio 3/2. Because of chemical properties of the elements that compose DNA strands, the DNA molecule has an orientation. Hence, it is reasonable to represent a sequence keeping trace of its orientation. In this case, when inverting a segment, one has to invert also the orientation of every single base contained in it in order to keep a consistent orientation of the resulting sequence. Fortunately, the oriented version of sorting by reversal is polynomially easy (in the literature this is known as the signed version of the problem), as proved in [HP95a], and improved in [KST97].

It is still an open problem whether sorting by transpositions is tractable or not. The best approximation known so far has ratio 3/2, as suggested in [BP98]. Again,
there is a better result for a restricted version of the problem: when the segment is transposed at a limited distance, there are better approximation algorithms. A transposition can be seen as a swapping of two adjacent segments where one is the segment that is being transposed and the other is the fragment that separates the two positions (the one before and the one after the event). One can also define a block interchange as the swapping of two non-intersecting segments not necessarily adjacent. It holds that sorting by block-interchange is solvable in polynomial time ([Chu96]). Such results can also be merged allowing more that one kind of rearrangement. For example, we have that the existence of a polynomial solution for sorting by reversals and transpositions is an open problem, but approximation algorithms have been developed which reach even better ratios in the signed case.

Similarly to sorting by reversals, sorting by translocations has an approximation solution (with ratio 2) for the general case, and a polynomial algorithm for the signed case ([Han96]), but here the $\mathcal{NP}$-hardness of the general case has not been proved. Sorting by translocations and reversals has only been approximated so far, but for sorting by signed translocations, concatenations, fissions and reversals a polynomial time algorithm is known ([HP95b]).

All sorting problems listed so far assume that the genomes given as input are composed of exactly the same set of genes. In fact, the inputs represent permutations of the same set of labels. In general, this will not be the case. Attempts to integrate sorting algorithms with deletions and insertions of segments have been done ([SLA+92]). Basically, deletions and insertions are computed in a first step by detecting all non common segments. Afterwords, such segments are removed, and sorting algorithms such as those above are applied on the remaining segments (which are then all common and thus correspond again to two permutations of the same set). The resulting distance is the sum of the deletion/insertion part and of the sorting step. The problem with such distances is that they are not metric distances (because, for example, triangular inequality does not hold), and they are therefore not useful for some purposes such as phylogenetic reconstruction (see section 1.3). Another attempt to take into account deletions and insertions has been done in [VDR99] with the transformation distance. This is also not a metric (therefore the term distance is not proper, but we keep it because this is how it is known in the literature) because it is not even symmetric. Nevertheless, it has the non trivial advantage of being computable in polynomial time (although this may be very expensive still) and actually take into account potentially all segment-based events. We will describe it in chapter 4 as we will make use of it for a specific purpose that fortunately does not require the distance to be symmetric. Another event that is somehow hard to take into account from the computational complexity point of view is duplication. A sorting by duplication would be an uninteresting problem, but combining duplication with the other sorting problems could be very useful. This would require the representation to be no longer just a permutation (or set)
but rather a string (or multiset) because repetitions of labels must be allowed. This results in most of the interesting cases to be \( \mathcal{NP} - \text{hard} \). An attempt to count duplications avoiding the multiset (or string) representation has been done with the so called exemplar distance in [San99]. The time cost of the resulting algorithm is exponential in the number of duplicated genes. As we discuss in section 2.3, gene duplication can be very frequent, and thus the exemplar distance could result unfeasible for many interesting applications.

The breakpoint distance differs from the other ones in the sense that it actually does not aim to count rearrangement events. [SB98]. It is defined again on permutations (although it can be extended to strings [San99]), and it counts all positions were adjacencies have changed in one permutation with respect to the other. This number is proven to give useful bound for the amount of rearrangement that might have caused the divergence between the two permutations, but it does not really count them. It is computable in time in the number of labels.

Computing syntenic distance is \( \mathcal{NP} - \text{hard} \) ([DJK+97]) and 2-approximation algorithms exist ([DJK+97, LN99]). We will tell more about syntenic distance in section 2.4 and chapter 3. For most rearrangement distances, the size of the diameter has also been investigated, that is the maximum possible distance for a given input size. This is interesting as it represents an upper bound on the distance (useful in the case where an approximation algorithm is being considered). Also, as we will see in chapter 3, being able to optimally compute the diameter may lead to other interesting results.

### 2.3 Gene duplication and its role in evolution

As it has been for other organisms, also the human genome revealed a surprisingly big amount of gene duplications. It has been claimed that “we actually have much fewer genes than expected” because many of them are just duplications of others. The release of the human genome has had a big media impact, but actually this high duplication rate surprise has been claimed for almost every newly available complete genome. For example, the completely sequenced genome of the bacterium Escherichia coli revealed a percentage of repeated genes close to 50% ([KTR95, Hin97]). The same holds for other bacteria such as *Mycoplasma genitalium* and *Mycoplasma pneumoniae* (see chapter 5), as well as for Bacillus subtilis ([K+97]), and the Saccharomyces cerevisiae yeast genome ([Han75, MAB+97]). There are also many theories supporting the duplication of whole genomes, especially in plants ([WS97]).

According to an hypothesis made in the seventies ([Ohn70]), gene duplication is the predominant mechanism for the evolution of new gene functions: once redun-
rable gene copies are available, mutations may accumulate in one of the copies, thus creating a new gene, while the other one(s) can go on carrying out the original function. Actually, not every duplicated gene gives rise to a new gene function. It can also become what is called a pseudo gene, that is a gene that eventually disappears after further evolution. The probably central role of gene duplication in evolution is the motivation for our special attention to this type of event. Both chapter 4 and chapter 5 will deal with (families of) paralogous genes. We believe that the topology and the amount of duplications in chromosomes are useful information for studying the evolution of organisms. Clustering paralogous genes within a genome means detecting families of genes that share a common origin and that have been recently duplicated possibly leading to the appearance of new functions. Analysing sets of recently duplicated genes, and studying the way their ordering and allocation changes along the chromosome, may provide interesting data about the way genes evolve. The way that a family of paralogous genes has evolved (possibly with respect to the same family in another organism in the case of a comparative study) may mirror some specific property of the organism it belongs to. Therefore, our studies on gene duplications are also meant as another way of possibly performing comparative genomics.

Another crucial role gene duplications have in relation to phylogenetic reconstruction comes from the observation [MMS95, Zha96, PH98, Pag98] that paralogy can be the cause of ambiguity when attempting to reconstruct evolutionary trees. Indeed, it was shown that two different sources of data like gene trees and species trees can result contradictory exactly because of gene duplications. A species tree is a tree where each node represents a specie, with present time ones at the leaves, and where a branching represents a speciation event that has taken place (or is supposed to have taken place) in evolution. A gene tree is a tree where the nodes are genes that belong to different species and that are homologs. Gene trees have been extensively used to reconstruct phylogenies (that is, species trees) under the assumption that gene trees are isomorphic to species trees. If some genes in the gene trees are paralogs, then there is actually a duplication event in the branching of the gene tree. In this case, the gene tree and the species tree in general do not agree. Therefore, in order to reconstruct the phylogeny, one should actually find a consensus tree among different gene trees such as the one suggested in [Pag98, MLZ00]. On the other hand, observations on evolutionary trees can be used to guess and date gene duplications as it is done in [CDFC00b, CDFC00a].

2.4 Overview of part I

Part I contains three chapters corresponding to three parallel investigations. Each chapter has its own introduction with preliminary definitions and description of previous work. Although they are all self-contained, they belong to the same domain of
investigation in the sense that they are all related to segment-based events such as those described in section 2.1. Moreover, they can all be used (some more directly than others) to perform genome comparisons. What follows are brief summaries of their contents.

Chapter 3 contains some new results on syntenic distance (which we introduced above) that are basically those of [PS00b]. It presents some new results on syntenic distance. First, we solve a conjecture by Liben-Nowell [LN99] for the syntenic diameter size, that is, for the length of the longest optimal move sequence for an instance of size \( n \), showing that it is, in fact, not \( 2n - 3 \) but \( 2n - 4 \) (an independent and different proof of this was provided by Kleinberg and Liben-Nowell in [LNK00b]). The structure of the proof further enables us to characterize the minimum number of translocations an optimal sequence of moves must have. This has important consequences for algorithmical purposes. The result is then generalized to what we call bidimensional syntenic diameter. We also address the apparently frequent case of multicomponent synteny (see section 3.4), and we suggest a way for performing inter-component moves, something that was not done in previous algorithms. We show however that, even in simple cases with more than two components involved, an optimal solution for a general instance of the problem using such moves is \( \mathcal{NP} \)-hard to obtain. Concerning nested synteny (see section 3.5), we introduce, we believe, a simpler, more general and efficient algorithm for obtaining a move sequence that is optimal for the cases that Liben-Nowell and Kleinberg called in [LNK00a] nested synteny, exact and linear (this last one is the case addressed in [LNK00a]). Moreover, we sketch an extension of the algorithm which we conjecture may optimally solve the connected nested synteny (without requiring linearity). Finally, we prove an inclusion theorem for the general synteny that is an extension of the one shown for linear synteny in [LNK00a].

Chapter 4 is basically [PMF+01]. It describes the tool PaTre which we developed with the aim of building paralogy trees. Given a family of paralogous genes, PaTre tries to infer a tree structure that represents which gene has been generated as a duplication of others. Observe that paralogy trees are different from gene trees as described in [Pag98]. We tested the tool on a simulator which is on its turn tested using suitable multiple alignments. Finally, we apply PaTre to some real gene families where it resulted that PaTre could still detect some acknowledged or very probable duplications of the family.

Finally, chapter 5 refers to [PV99] and to [Pis98] (under the supervision of A.Viari, INRIA). It addresses the problem of studying some properties of families of paralogous genes. The idea is to represent paralogy relations (at different rates of minimum sequence similarity required to establish that paralogy holds) by means of a graph, and then to compare this graph to random graphs. The resulting differences should correspond to biologically significant properties. Wherever biolog-
ical data significantly differs from random data, then further investigation is worth being done. The novelty of our approach comes from performing this comparison with random data at a graph level. Usually this is done at the sequence level and the comparisons are pairwise. Our approach enables to represent (and then study) whole families of paralogs, and not just pairwise similarity. Moreover, our notion of similarity that characterizes the paralogy relation is not fixed \textit{a priori}. The method is applied to a wide range of possible clusterings. The method itself can detect which of them are the interesting ones. The chapter shows some attempts for the detection of a suitable model of random graph. Finally, one of the models is chosen (although the search for a random graph model deserves further attention) and the method applied to the genome of the \textit{Mycoplasma genitalium} bacterium.
Chapter 3

On Syntenic Distance

3.1 Introduction

Similarity between biological sequences is frequently measured by counting point mutations (substitutions, deletions and insertions). However, mutations happen also at a larger scale. Whole fragments of genomes, involving sometimes thousands of DNA bases, often containing one or more genes, may thus be deleted, moved about, reversed, copied (duplicated) or exchanged (also between chromosomes in the case of multi-chromosomal genomes). One may therefore wish to measure intra or inter-species distances based on such larger scale rearrangement events happening at a genomic level, instead of on the history of point mutations.

Methods for measuring distances between genomes may be classified into three main categories depending on what type of rearrangement events are allowed and what kind of information is represented in the input. Data basically consist of genome representations (or representation of significant parts of it, such as chromosomes). The representations come under the form of one or more sequences of genes (this will be the case when the order of the genes along the chromosome is taken into account), or sets of genes (when that order is ignored). Each sequence or set corresponds to a chromosome. The types of rearrangement events allowed by the various approaches may of course depend on the data considered. Most rearrangement distances take the gene order into account and allow for one or more type of event among deletion, duplication, reversal, inter or intra-chromosome transposition and inter-chromosome translocation [BP98] [Chr96] [Han96] [Cap97] [HP95a] [HP95b]. In the case of breakpoint distances [Chr98], the gene order is considered as well, but the type of rearrangement events that may have changed it is not fixed. Finally, the syntenic distance [FNS96], which is exclusively reserved for multi-chromosomal genomes, ignores the gene order. A genome in this case consists of a number of unordered sets, each one corresponding to a chromosome and containing elements that are the genes belonging to that chromosome). Furthermore, only three types of
rearrangement events are considered. These are fusion (of two sets into one), fission (of one set into two), and translocation (two sets exchange part of their contents). Hence, the only information that syntenic distance is able to capture is the synteny of genes (two genes are syntenic if they belong to the same chromosome).

Syntenic distance cannot keep track of intra-chromosomal events, as no information about topological properties within chromosomes (the relative order of genes, their distances from one another and their sizes) is stored in the sets representation. Obtaining such distance requires, as usual, determining the minimum number of operations among the three just mentioned that are necessary to transform a first genome into a second.

Syntenic distance is proved to be $NP$-hard in [DJK+97] and a 2-approximation algorithm is suggested (algorithm $\mathcal{H}$). The bound is tight. This bound is the same for Liben-Nowell’s algorithm $\mathcal{F}$ [LN99, LN01] which is an improvement of Ferretti’s $\mathcal{F}$ algorithm [FNS96]. Liben-Nowell also proves that, assuming the instance is partitioned into components with distinct genes, no approximation ratio better than 2 can be reached as long as only intra-component moves are performed, which is what all algorithms proposed so far do. Both in [DJK+97] and [LN99], many interesting properties of syntenic distances are proved and some particular classes of instances, such as linear synteny and exact synteny, are examined. For one particular class of linear synteny, it is proved in [LNK00a] that under certain conditions (requiring inclusion properties among the sets of syntenic genes, resulting in what is called a nested synteny), the distance is computable in polynomial time. Finally, in [LN99], the syntenic diameter, that is, the length of the longest optimal move sequence for an instance of size $n$, is discussed. It was conjectured that its size is $2n-3$.

This chapter presents some new theoretical and algorithmical results concerning nested synteny and general synteny. We start by giving some definitions and recalling earlier results (section 3.2). Finding an optimal solution for the syntenic distance, in particular for nested synteny, is deeply dependent on being able to calculate the syntenic diameter (see [LN99]). We therefore provide a solution to a conjecture by Liben-Nowell [LN99] for the syntenic diameter size, that is, for the length of the longest optimal move sequence for an instance of size $n$. We use a direct proof to show that the size is, in fact, not $2n-3$ but $2n-4$ (an independent and different proof of this was provided by Kleinberg and Liben-Nowell in [LNK00b]). The structure of the proof further enables us to characterize the minimum number of translocations an optimal sequence of moves must have. This has important consequences for algorithmical purposes. We generalize the result to include what we call bidimensional syntenic diameter. When the initial instance has more than one component, we then suggest a way for performing inter-component moves, something that was not done in previous algorithms (section 3.4). We show however that, even in simple cases with more than two components involved, an optimal solution for a general instance
of the problem using such moves is \( \mathcal{NP} \)-hard to obtain. We follow this by introducing an algorithm which is, we believe, simpler, more general and efficient than in [LNK00a] for obtaining a move sequence that is optimal for the exact and linear cases when the instance is nested. In addition, we sketch an extension which we conjecture may optimally solve the connected general nested case (section 3.5). The algorithm time complexity is \( O(k^2 + kn + k \log k) \) where \( k \) and \( n \) are the respective number of chromosomes in the two genomes and \( k \leq n \). Finally, concerning general synteny, we prove an inclusion theorem equivalent to the one shown in [LNK00a] for linear synteny (section 3.6).

3.2 Definitions and previous results

A genome is a collection of \( k \) sets (chromosomes), each chromosome being a subset of a set of \( n \) objects (genes). A genome mutates through one among the following three possible moves involving the sets: fusion: \( \{S, T\} \rightarrow S' = S \cup T \), fission: \( S \rightarrow \{S', S''\} \) where \( S = S' \cup S'' \), and translocation: \( \{S, T\} \rightarrow \{S', T'\} \) with \( S \cup T = S' \cup T' \). Given two genomes \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \) over the same gene set \( \Sigma \), the syntentic distance between \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \), denoted by \( D(\mathcal{G}_1, \mathcal{G}_2) \), is the minimum number of moves needed to transform \( \mathcal{G}_1 \) into \( \mathcal{G}_2 \). Following the syntentic distance literature, we adopt the compact representation of the problem: suppose that genomes \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \) contain \( k \) and \( n \) sets respectively (these correspond to the number of chromosomes in each). The compact representation \( \mathcal{G}_2' \) of \( \mathcal{G}_2 \) with respect to \( \mathcal{G}_1 \) is defined as follows: replace the \( j \)th set \( \mathcal{G}_{2,j} \) of \( \mathcal{G}_2 \) for \( 1 \leq j \leq n \) by the set \( \mathcal{G}_{2,j}' = \{i \mid x \in \mathcal{G}_{1,i}, 1 \leq i \leq k\} \). Let \( \mathcal{G}_1' \) be defined in a symmetrical way as the compact representation of \( \mathcal{G}_1 \) with respect to \( \mathcal{G}_2 \). Computing \( D(\mathcal{G}_1, \mathcal{G}_2) \) is the same as computing the distance between \( \mathcal{G}_1' \) and \( \{1, 2, \ldots, n\} \), which is also the same as computing the distance between \( \mathcal{G}_2' \) and \( \{1, 2, \ldots, k\} \) (this latter is called the dual problem of the former one). Notice that, since the last one mentioned equals \( D(\mathcal{G}_2, \mathcal{G}_1) \), we have that \( D(\mathcal{G}_1, \mathcal{G}_2) = D(\mathcal{G}_2, \mathcal{G}_1) \), that is, the syntentic distance is symmetrical (reflexivity and triangular inequality also hold, and the syntentic distance is therefore a metric). Observe that we can assume without loss of generality that \( n \geq k \) since we can always consider the dual problem instead. We denote by \( S(n, k) \) an instance of synteny, that is an instance of the synteny problem in the compact representation where there are \( n \) elements and \( k \) initial sets, and by \( D(S(n, k)) \) the minimum number of moves required to solve \( S(n, k) \). Two sets \( S_1 \) and \( S_2 \) are connected if and only if \( S_1 \cap S_2 \neq \emptyset \). In that case, they belong to the same component. For an element \( \ell \), \( \text{count}(\ell) \) denotes the number of chromosomes in which \( \ell \) appears.

**Definition 1** [Canonical move sequence] Let \( \mathcal{G}_1, \mathcal{G}_2 \) be an instance of synteny. A move sequence \( \sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m) \) such that \( m = D(\mathcal{G}_1, \mathcal{G}_2) \) and all fusions appear before all translocations, which themselves appear before all fissions is called a canonical move sequence.
Definition 2 [Non-redundant move sequence] Let \( G_1, G_2 \) be an instance of synteny. A move sequence \( \sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m) \) containing no moves that produce two sets with a non-empty intersection is called a non-redundant move sequence.

In [DJK+97] and [LN99] the following is proved.

Proposition 1 [Optimal canonical and non-redundant move sequence] For any instance of synteny, there is always an optimal move sequence that is non-redundant and canonical.

Theorem 1 [Monotonicity] Let \( S_1, S_2, \ldots, S_k \) and \( T_1, T_2, \ldots, T_k \) be two collections of sets where for all \( 1 \leq i \leq k \), we have \( T_i \subseteq S_i \). Let \( n = |\bigcup_i S_i| \) and \( n' = |\bigcup_i T_i| \). Let \( S(n, k) = S_1, S_2, \ldots, S_k \) and \( T(n', k) = T_1, T_2, \ldots, T_k \). Then \( D(S(n, k)) \geq D(T(n', k)) \).

We say that a move solves an element \( x \) if it produces the set \( \{x\} \), and \( x \) appears nowhere else anymore.

There exists a special class of synteny instance, called exact synteny [DJK+97], which results in a conceptually simpler problem than the general synteny (although in [DJK+97] the \( \mathcal{NP} \)-hardness of syntenic distance is proved using an exact instance of the problem, and therefore also exact synteny is \( \mathcal{NP} \)-hard). Exact instances have the property that at each step a splitting move can be done. A splitting move is a move which increases by 1 the number of components. Observe that a translocation which solves an element is a special case of a splitting move.

Definition 3 An instance of synteny is connected if it consists of one component only. A connected instance \( S(n, k) \) of the synteny problem is exact if \( n = k \) and \( D(S(n, k)) = n - 1 \).

It is easy to see that in an exact synteny instance, all moves are splitting moves. The reason is that initially there is one component, at the end there are \( n \) of them, and each move increases by at most one the number of components.

In [LNK00a], Liben-Nowell and Kleinberg considered a special class of instances which they called nested synteny:

Definition 4 An instance of synteny \( S(n, k) = S_1, S_2, \ldots, S_k \) is nested if, for all \( i \neq j \), either (1) \( S_i \cap S_j = \emptyset \), (2) \( S_i \subseteq S_j \) or (3) \( S_j \subseteq S_i \).

Again in [LNK00a], a special case of move sequence is considered. This is called a linear move sequence. For an instance \( S(n, k) \), the sequence starts with \( k-1 \) moves that are either fusions or translocations that solve an element. The remaining moves are all fissions. We name linear synteny the problem of finding the shortest linear
move sequence for a given instance. In other words, there is a suitable order of the initial \( k \) sets such that the linear move sequence just consists of fusing the first two sets, or translocating if an element has count 1 or 2, for \( k-1 \) times, and then fissioning the remaining unique set of unsolved elements. In [DJK+97] it has been shown that, even if linear syntenic distance \( d' \) has little biological motivation, calculating it is interesting as we are guaranteed that within a component we have \( d \leq d' \leq d + \log_{1/\alpha}(d) \) where \( d \) is the unconstrained distance. Finally, an instance \( S(n,k) \) is linear exact if it can be solved with \( \max\{n,k\} - 1 \) moves.

### 3.3 Syntenic diameters

#### 3.3.1 Syntenic diameter

In [LN99], the syntenic diameter problem is addressed. The idea is to find the maximum possible distance between two genomes of a given size. Thanks to the monotonicity result, we can identify the instance of size \( n \) that requires the highest number of moves to be solved. This is the synteny instance consisting of \( n \) copies of the set \( \{1, 2, \ldots, n\} \). This instance was denoted \( K_n(n,n) \) by Liben-Nowell [LN99]. Where there is no ambiguity, we simplify this to \( K(n,n) \). Liben-Nowell conjectured that \( D(K(n,n)) = 2n-3 \). As will be seen later, finding an optimal solution for the synteny distance, in particular in the case of a nested synteny, is deeply dependent on being able to provide an optimal move sequence for an instance of the diameter. This is what we do now, proving that \( K(n,n) = 2n-4 \) differently from what was conjectured by Liben-Nowell (an independent and different proof of this was given by Kleinberg and Liben-Nowell in [LNK00b]. Their proof does not contain a result we prove and which we need for the algorithm we present later).

We start by giving a generalisable counter example that shows how to solve \( K(n,n) \) in \( 2n-4 \) steps: from the initial \( K(n,n) \), we perform \( n-4 \) fusions so that we obtain 4 times the set \( \{1, 2, \ldots, n\} \). We then do 2 translocations in order to get \( \{1, 2\}, \{1, 2\}, \{3, 4, \ldots, n\}, \{3, 4, \ldots, n\} \), and 2 more translocations to obtain \( \{1\}, \{2\}, \{3\}, \{4, \ldots, n\} \). Finally, with \( n-4 \) fissions we get the desired result. Overall, this takes \( n-4+2+2+n-4=2n-4 \) moves to solve \( K(n,n) \). Let us observe that there exists other optimal move sequences made of translocations only. Since there is no way to solve \( K(3,3) \) in only \( 2 \cdot 3 - 4 = 2 \) moves or less, we have proved that:

**Proposition 2** \( D(K(n,n)) \leq 2n - 4, \forall n \geq 4. \)

In fact \( 2n-4 \) is also a lower bound, and hence equality holds as we show at the end of the section. We start by proving some intermediate results.

**Lemma 1** In an optimal move sequence solving \( K(n,n) \), the number \( m_1 \) of fusions is equal to the number \( m_3 \) of fissions, and the number \( m_2 \) of translocations equals the optimal solution of \( K(n-m_1, n-m_1) \) when only translocations are allowed.
Proof Consider a non-redundant optimal move sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ for $K(n, n)$ in canonical order, that is we have $m_1$ fusions followed by $m_2$ translocations, followed by $m_3$ fissions, as shown below:

$$K_n(n, n) = S_0 \xrightarrow{m_1 \text{ fusions}} S_1 \xrightarrow{m_2 \text{ translocations}} S_2 \xrightarrow{m_3 \text{ fissions}} S_3$$

where sets $S_1$ and $S_2$ are intermediate instances and $S_3 = \{\{1\}, \{2\}, \ldots, \{n\}\}$ is the final instance. Since translocations do not change the number of sets, we have that $S_1$ and $S_2$ have the same number of sets. Also $S_0$ and $S_3$ have the same number of sets as they both have $n$ of them. Therefore, the $m_1$ fusions must have decreased the number of sets as much as the $m_3$ fissions increase it. Since each fusion (resp. fission) decreases (resp. increases) the number of sets by exactly one, then it must be $m_1 = m_3$.

Let us now examine the $m_2$ translocations which transform the instance $S_1$ into $S_2$. Consider the $n-m_1$ distinct sets of the instance $S_2$, and apply the following renaming to the elements in $S_1$ and $S_2$ (obtaining respectively $S'_1$ and $S'_2$): for each set $i$ of $S_1$ ($i = 1, \ldots, n-m_1$), rename $x_i$ all elements that end up in set $i$ in $S_2$. In this way, we obtain an instance $S'_1$ consisting of $n-m_1$ times the set $\{x_1, x_2, \ldots, x_{n-m_1}\}$ which is just the instance $K(n-m_1, n-m_1)$, and another instance $S'_2 = \{\{x_1\}, \{x_2\}, \ldots, \{x_{n-m_1}\}\}$ which is the solution for $S'_1$. An optimal move sequence leading to such a solution consists indeed in the $m_2$ translocations that, in the solution for $K(n, n)$, transform $S_1$ into $S_2$. We therefore have that the $m_2$ translocations which transform the instance $S_1$ into $S_2$ correspond to an optimal solution, containing only translocations, for the instance $K(n-m_1, n-m_1)$. □

After the $m_2$ translocations above, we have the instance $S_2$ with $n-m_1$ sets where all the $n$ elements are partitioned. In fact, each one of them only appears in only one of the sets (this must be the case otherwise we could not terminate the move sequence with only fissions). Notice that it is a property of general synteny the fact that a canonical move sequence has a partition of the set $\{1, 2, \ldots, n\}$ before the final fissions phase. Moreover, in all cases the renaming of the elements as in proof of lemma 1 can be done to show that actually the $m_1$ fusions plus the translocations represent an optimal move sequence for the instance that has as many elements as $n-m_1$.

Lemma 2 If only translocations are performed, then $D(K(n, n)) \geq 2n - 4$.

Proof Let us consider an arbitrary optimal move sequence for $K(n, n)$ containing only translocations. We now show that this contains at least $2n-4$ moves.

Since each element has initial count $n$ and a move may decrease such count by at most 1, no element can be solved in the first $n-2$ moves. At best, we end up with some elements having count 2 which may therefore be solved in subsequent steps.
We claim now that, at the end of the first \(n-2\) moves, all elements with count 2, let us say we have \(p\) of them (\(p\) may be 0), will find themselves in at most four sets \(S_1, S_2, S_3\) and \(S_4\). Indeed, in order to obtain a count of 2 for elements \(x_1, \ldots, x_p\) which all appear \(n\) times at the beginning, each of the first \(n-2\) moves must involve two sets that both contain \(x_1, \ldots, x_p\). Hence, the first \(n-4\) operations are of the type \(\{x_1, \ldots, x_p, U_1\}, \{x_1, \ldots, x_p, U_2\} \rightarrow \{x_1, \ldots, x_p, U_3\}, \{U_4\}\), where \(U_1\) or \(U_2\) (but not both) may be empty, \(U_3\) may be empty but never \(U_4\) (otherwise, the operation would not be a translocation). After these translocations, the elements \(x_1, \ldots, x_p\) have all count 4. In the next two operations, we may either leave the \(p\) elements together (performing again moves like the one just shown), or we may split them by doing the following move: \(\{x_1, \ldots, x_p, U_1\}, \{x_1, \ldots, x_p, U_2\} \rightarrow \{x_1, \ldots, x_h, U_3\}, \{x_{h+1}, \ldots, x_p, U_4\}\), where \(1 \leq h < p\) and \(U_1, U_2, U_3\) and \(U_4\) may, this time, all be empty. In the case where the last two moves split the \(p\) elements with count 2 among 4 sets, the elements in all the other \(n-4\) sets will have count at least 4 (since such sets were translocated \(n-4\) times only).

Let \(S(n, k)\) be the instance of synteny obtained and \(c\) the number of its components. We have that \(D(S(n, k)) \geq n - c\) [DJK+97]. The bound is absolute: no optimal move sequence, whether it contains inter-component moves (as shall be considered later) or not, can do less operations than that. This gives us a lower bound on the number of steps we still need to perform after the \(n-2\) translocations just described. If \(c\) is 1, then we are done, since we would need at least \(n-1\) further moves giving a total of at least \((n-2) + (n-1) = 2n-3\) instead of the \(2n-4\) that we are capable of producing. If \(c\) is 2, we need at least \(n-2\) more moves after the \(n-2\) initial ones, giving the total lower bound of \(2n-4\). Finally, if \(c\) is greater than 2, we have just shown above that only 2 (say the first and the second) of the \(c\) components may contain all the elements with count 2, if any such exist.

The other components all contain elements with higher counts. Let us call \textit{intra-component moves} those involving only sets that belong to a same component. Moves that involve sets belonging to two different components are called \textit{inter-component moves}. If only intra-component moves were done, each these other components would therefore require at least one extra move with respect to \(n_i - 1\), where \(n_i\) is the number of elements of component \(i\). If inter-component moves were allowed, \(\sum_{j=3}^{c} (n_j - 1) + (c - 2)\) would be a lower bound on the number of moves required. The \((c - 2)\) extra moves in relation to an exact instance with \((c - 2)\) components would come from the fact that an inter-component move (that is, the first move between two different components) does not decrease the count of any element. Even if an inter-component move enabled to solve an element, at least \((c - 2)\) moves would therefore not decrease the counts of any element. Since each of these other components has at least one element which has count at least 4, overall at least \((c - 2) = (\sum_{j=3}^{c} 1)\) extra moves will be “wasted” for getting nearer to the solution.
Hence, we need at least
\[
(n-2) + \sum_{i=1}^{c} D(S(n_i, k_i)) \geq n - 2 + (n_1 - 1) + (n_2 - 1) + \sum_{j=3}^{c} (n_j - 1) + 1 = 2n - 4.
\]

moves in this case also, where \( k_i \) is the number of sets in component \( i \). \( \Box \)

A direct consequence of Lemma 2 is that the diameter is exactly \( 2n-4 \) for all \( n \geq 4 \) in the case where neither fusions nor fissions are performed. The question remains open whether by using also fusions and fissions a \( K(n,n) \) instance could be solved in even less moves. Such question will be answered by Theorem 2, but we can already state the following result.

**Proposition 3** The \( m_2 \) translocations of Lemma 1 correspond to an optimal solution for \( K(n-m_1,n-m_1) \).

**Proof** The thesis follows directly from Lemma 1 and Lemma 2. \( \Box \)

We can now prove the following theorem.

**Theorem 2** \( D(K(n,n)) = 2n-4, \forall n \geq 4 \), and the maximum number of fusions allowed in an optimal move sequence is \( n-4 \).

**Proof** If the number \( f \) of fusions is equal to 0, then the result follows immediately from Lemma 2. Let us now suppose that there are \( f > 0 \) fusions. By Lemma 1, any canonical optimal move sequence contains also \( f \) fissions and \( D(K(n-f,n-f)) \) translocations (leading to \( n-f-1 \) singleton sets and a set with \( f+1 \) elements that have to be fissioned). We prove by induction on \( n \) that in this case also, we have \( D(K(n,n))=2n-4 \) for all \( n \geq 4 \).

The base case is \( K(4,4) \) for which it holds that \( D(K(4,4)) = 4 = 2n-4 \). The inductive step consists in showing that \( D(K(n,n)) = 2n-4 \) holds if it holds for all \( n_0 \) such that \( 4 \leq n_0 < n \). By hypothesis (since \( f \geq 1 \) and therefore \( n-f < n \)), we know that \( D(K(n-f,n-f)) = 2(n-f)-4 = 2n-4-2f \). We therefore have that \( D(K(n,n)) = D(K(n-f,n-f)) + 2f = 2n-4 \).

It is important to observe here that \( D(K(n,n)) = 2n-4 \) holds if \( n \geq 4 \). As a consequence, if we fuse more than \( n-4 \) times, we would have that the \( D(K(n-f,n-f)) \) translocations would be more than \( 2(n-f)-4 \), and then we would no longer obtain an optimal move sequence for \( K(n,n) \). Therefore, we must have at most \( n-4 \) fusions. \( \Box \)

As a consequence of Theorem 2, since there are at most \( 2n-8 \) fusions and fissions, we have that:
Corollary 1 In an optimal move sequence for $\mathcal{K}(n, n)$ with $n \geq 4$, there are always at least 4 translocations.

If $n = 1, 2$, we have trivially that $D(\mathcal{K}(n, n)) = n - 1$. For $n = 3$, it can be checked that $D(\mathcal{K}(n, n)) = n = 3$.

3.3.2 Bidimensional syntenic diameter

The $\mathcal{K}(n, n)$ instance is a particular case of $\mathcal{K}(n, m)$, which consists of $m$ copies of the set $\{1, 2, \ldots, n\}$. We call instances of $\mathcal{K}(n, m)$ with $n \neq m$, bidimensional syntenic diameter since in this case the size depends on two input parameters. The value of $D(\mathcal{K}(n, m))$ will therefore be in general a function of both $n$ and $m$. Notice that $D(\mathcal{K}(n, k))$ gives an upper bound for $D(S(n, k))$. When $n \neq m$, it is no longer true that the number of fusions equals the number of fissions. In this case in fact, we have the following.

Lemma 3 Let $n_1$ be the number of fusions in an optimal solution of $D(\mathcal{K}(n, m))$. We then have that the number of fissions is $n_3 = n - m + n_1$.

Proof Again, let us consider a canonical optimal move sequence solving $D(\mathcal{K}(n, m))$. Initially there are $m$ sets, which become $m - n_1$ after the fusions. Translocations do not change the number of sets, and fissions only increase it. Since we must have $n$ sets left at the end, we have that $m - n_1 \leq n$. Furthermore, the number of fissions are exactly $n - (m - n_1)$. $\square$

Theorem 3 $D(\mathcal{K}(n, m)) = m + n - 4$ for all $n, m \geq 4$. Furthermore, at most $\max\{n, m\} - 4$ of these operations may be fusions and $n - 4$ may be fissions.

Proof We have to analyse separately various subcases:

$4 \leq m \leq n$ In this case, the $\mathcal{K}(n, m)$ is just an intermediate case of the optimal solution we gave for $\mathcal{K}(n, n)$. Therefore, $m - 4$ fusions followed by $4$ translocations and $n - 4$ fissions as in the solution for $\mathcal{K}(n, n)$ represent an optimal move sequence for $\mathcal{K}(n, m)$. We thus have $D(\mathcal{K}(n, m)) = m - 4 + 4 + n - 4 = m + n - 4$.

$m > n$ In this case, we have an instance with more sets than elements, and we consider its dual, which contains $n$ copies of $\{1, 2, \ldots, m\}$, bringing us back to the previous case.

Therefore, we conclude that $D(\mathcal{K}(n, m)) = m + n - 4$ for all $n, m \geq 4$. The results concerning upper bounds on the number of fusions and fissions follow directly from those of $\mathcal{K}(n, n)$ and from Lemma 3: as in the case of $\mathcal{K}(n, n)$, we must have
at least 4 translocations. Therefore, the number \( n_1 \) of fusions are at most \( m - 4 \) if \( m \geq n \), and at most \( n - 4 \) otherwise. Finally, since we have at least 4 sets after the translocations and the goal is to end up with \( n \) sets, then we may have at most \( n - 4 \) fissions. \( \square \)

If \( m < 4 \), we split into three possible cases.

\( m = 1 \) We have one copy of \( \{1, 2, \ldots, n\} \) and clearly the only thing to do is \( n - 1 \) fissions, giving \( D(K(n, 1)) = n - 1, \forall n \).

\( m = 2 \) Here the best thing to do is to translocate the two copies of \( \{1, 2, \ldots, n\} \) into \( \{1, 2, \ldots, h\} \) and \( \{h + 1, \ldots, n\} \) for any \( 1 \leq h \leq n \) (or to fuse if \( n = 1 \), in which case we are done), and then do \( n - 2 \) fissions. We therefore have \( D(K(n, 2)) = n - 1 \) if \( n \geq 2 \), (and \( D(K(1, 2)) = 1 \) for \( n = 1 \)).

\( m = 3 \) It is easy to check that we cannot make it in less than \( n \) moves (for example, fusing first and then doing as for \( m = 2 \)). Thus \( D(K(n, 3)) = n \). Again, \( n = 1 \) is an exception as \( D(K(1, 3)) = 2 \).

Finally, if \( n < 4 \) and \( m \geq 4 \):

\( n = 1 \) We have \( m \) copies of the set \( \{1\} \), and clearly \( m - 1 \) fusions is the move sequence to do. Therefore \( D(K(1, m)) = m - 1, \forall m \).

\( n = 2 \) Fusing the initial \( m \) copies of \( \{1, 2\} \) \( m - 2 \) times, and then translocating \( \{1, 2\} \) and \( \{1, 2\} \) into \( \{1\} \) and \( \{2\} \) is the optimal move sequence, giving \( D(K(2, m)) = m - 1 \).

\( n = 3 \) The move sequence consisting in \( m - 3 \) fusions followed by the 3 translocations which solve \( K(3, 3) \) is optimal (although not unique), hence \( D(K(3, m)) = m \).

Finally, let us observe that Lemma 1 and Lemma 3 can be extended to any instance \( S(n, k) \) of general synteny with \( k \leq n \) in the sense that the number \( m_3 \) of fissions is always \( m_3 = n - k + m_1 \) where \( m_1 \) is the number of fusions.

### 3.4 The multi-component case

#### 3.4.1 General observations

All algorithms for syntenic distance presented in the literature so far perform only moves involving sets that belong to a same component. We call such moves *intra-component moves*. Moves that involve sets belonging to two different components are called *inter-component moves*. 
3.4. The multi-component case

As mentioned in the introduction, Liben-Nowell has shown in [LN99] that, in the case of instances with more than one component, no approximation ratio better than 2 may be reached as long as only intra-component moves are performed. In fact, there are instances with $c > 1$ components for which the concatenation of optimal local solutions for the $c$ components leads to a global solution that is close to twice the optimal. An example is the instance composed of the set $\{1, 2, \ldots, n-1\}$ and $n-1$ times the set $\{n\}$. Clearly, the only thing to do within component $\{1, 2, \ldots, n-1\}$ is to fission $n-2$ times. Inside the other component, we need to fuse $n-2$ times the copies of $\{n\}$ in order to have only one left. The overall number of moves is $2n-4$ while the instance could have been solved with $n-1$ translocations by performing, for example, first the translocation $\{1, 2, \ldots, n-1\}, \{n\} \rightarrow \{2, 3, \ldots, n\}, \{1\}$, and then $n-3$ more translocations of the kind $\{i, \ldots, n-1, n\}, \{n\} \rightarrow \{i\}, \{i+1, \ldots, n\}$ using one of the copies of $\{n\}$ at each step while solving $i \in \{2, \ldots, n-1\}$. At the end, the only two remaining sets are $\{n-1, n\}$ and $\{n\}$. One more translocation is then enough for solving the instance. This gives a total of $1 + n - 3 + 1 = n - 1$ moves, which represents almost half the number we would need if we did not perform the inter-component moves. The reason is that almost each translocation of the globally optimal move sequence has the same effect as a fission in one component (an element with count one is solved) plus a fusion in the other (one of the redundant copies of the set $\{n\}$ disappears). The only exception to this concerns the first translocation between two different components. Such a translocation has the same effect as a fission but fuses no elements together. This last is an important observation and will be used in section 3.4.3 to show that optimizing by performing inter-component moves is an $\mathcal{NP}$-hard problem.

3.4.2 Optimizing in a special case of two components by performing inter-component moves

We start by showing how, in the special case of just two components, a move sequence may be optimized by performing inter-component moves.

**Theorem 4** Let $S(n, k)$ be an instance composed of:

- a component $C$ which contains a set of $g$ sets – denoted by $S_1, \ldots, S_g$ – which have to be fused all together in an optimal local move sequence, and such that the set is maximal (no other set needs to be fused with these) and unique (there exists no other set of sets in $C$ needing to be fused);

- a component $C'$ consisting of the single set $\{x_1, \ldots, x_f\}$.

Then performing inter-component moves saves $\min\{f, g\} - 2$ moves.
Proof Let $C' = \{x_1, x_2, \ldots, x_f\}$ and $C$ contain the sets $S_1, \ldots, S_g$ that must be fused together. Our goal is to obtain the sets $\{x_1\}, \{x_2\}, \ldots, \{x_f\}$ and the set $S = S_1 \cup \cdots \cup S_g$. We observe that the global solution consisting of the concatenation of the two separate intra-component solutions that reach this goal takes $(f-1) + (g-1) = f + g - 2$ moves overall. Let us first consider the case $g \leq f$. The following $g$ inter-component moves can be performed:

1. $\{x_1, \ldots, x_f\}, S_1 \rightarrow \{x_1\}, \{x_2, \ldots, x_f\} \cup S_1$
2. $\{x_2, \ldots, x_f\} \cup S_1, S_2 \rightarrow \{x_2\}, \{x_3, \ldots, x_f\} \cup S_1 \cup S_2$
   …
   g. $\{x_g, \ldots, x_f\} \cup S_1 \cup S_2 \cup \cdots \cup S_{g-1}, S_g \rightarrow \{x_g\}, \{x_{g+1}, \ldots, x_f\} \cup S_1 \cup S_2 \cup \cdots \cup S_g$

after which the only unsolved part left is the set $\{x_{g+1}, \ldots, x_f\} \cup S$. With $f-g$ more fissions, we reach our declared goal. This gives a total of $g+f-g = f$ moves. We therefore saved $g-2 = \min\{f, g\} - 2$ moves. Moreover, if we do not fission immediately, the $f-g$ elements left in component $C$ can be used later on for a further optimization.

Conversely, if $g > f$, we can perform the $f$ translocations:

1. $\{x_1, \ldots, x_f\}, S_1 \rightarrow \{x_1\}, \{x_2, \ldots, x_f\} \cup S_1$
2. $\{x_2, \ldots, x_f\} \cup S_1, S_2 \rightarrow \{x_2\}, \{x_3, \ldots, x_f\} \cup S_1 \cup S_2$
   …
   f. $\{x_f\} \cup S_1 \cup S_2 \cup \cdots \cup S_{f-1}, S_f \rightarrow \{x_f\}, S_1 \cup S_2 \cup \cdots \cup S_f$

after which all the $x_i$ are solved and there remain only the $g-f$ sets $S_{f+1}, \ldots, S_g$ which still need to be fused with the set $S_1 \cup S_2 \cup \cdots \cup S_f$. Hence, $g-f$ more fusions are required for a total of $g$ moves. In this case, we saved $f-2$ moves, which equals $\min\{f, g\} - 2$ again. □

In the case where inter-component moves such as those of Theorem 4 are performed, we say that components $C$ and $C'$ have been paired. We claim that such inter-component moves are the only ones that make us save operations with respect to the case where only intra-component moves are performed.

### 3.4.3 Finding an optimal components pairing is \textit{NP}-hard

Usually, while components such as $C'$ of Theorem 4 will contain a unique set that needs to be fissioned into singletons, components such as $C$ may contain more than one set of sets which need to be fused. In a more general way, an instance will be composed of components where both fusion and fission operations need to be performed. The general optimal pairing problem consists in finding the pairing (in the sense given at the end of section 3.4.2) of the components of an instance that
will result in an optimal move sequence.

For some classes of multi-component instances, it is possible to characterize an optimal move sequence when inter-component operations are allowed. An example of this is expressed in the following proposition.

**Proposition 4** Let $S(n, k)$ be an instance composed of:

- $c \geq 2$ components $C_1, \cdots, C_c$ where each $C_i$ contains $g_i$ times the set $\{y_i\}$;
- $d \geq 1$ components $C'_1, \cdots, C'_d$ with $C'_j = \{x^i_j \mid 1 \leq i \leq c\}$.

If inter-component moves are allowed, then an optimal move sequence solving $S(n, k)$ will be one which, at the same time:

- performs the maximum number of pairings between components $C_1, \cdots, C_c$ and components $C'_1, \cdots, C'_d$;
- translocates each component $C_i$ for $1 \leq i \leq c$ with a minimum number of different components among $C'_1, \cdots, C'_d$.

**Proof** Derives immediately from the observation at the end of section 3.4.1 that the first translocation done between two different components is the only one which saves no operation in relation to doing only intra-component moves. Indeed, such a translocation replaces a fission only instead of a fission and a fusion as do the other (new) translocation operations in the inter-component optimization procedure. An optimal move sequence must therefore economize as much as possible on such translocations while doing a maximum number of pairings. □

We now show that even for such a class of multi-component instances, the optimal pairing problem is $\mathcal{NP}$-hard. We prove this by reducing 3-Partition to the optimal pairing problem. We start by recalling the 3-Partition problem which has been proved to be $\mathcal{NP}$-hard in the strong sense [GJ78].

**Definition 5 [3-Partition problem]** Given a finite set $A$ of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$, check whether $A$ can be partitioned into $m$ disjoint sets $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$.

We now state and prove the $\mathcal{NP}$-hardness of our problem.

**Theorem 5** If $\mathcal{P} \neq \mathcal{NP}$, then finding an optimal component pairing for inter-component moves for the class of restricted synteny instances given in Proposition 4 is not solvable in polynomial time.
Proof Let us consider an arbitrary instance of the 3-Partition problem. We thus have a finite set \( A \) containing \( 3m \) elements, a bound \( B \in \mathbb{Z}^+ \), a size \( s(a) \in \mathbb{Z}^+ \) for each \( a \in A \) such that \( B/4 < s(a) < B/2 \) and \( \sum_{a \in A} s(a) = mB \). We build an instance \( S(n = 3m + Bm, k = (3 + 1)m) \) in the following way. The instance will contain:

- \( 3m \) sets \( C_1, \ldots, C_{3m} \) having each \( s(C_i) \) copies of a same element \( y_i \) with \( B/4 < s(C_i) < B/2 \) for a \( B \in \mathbb{Z}^+ \);
- \( m \) distinct components \( C'_1, \ldots, C'_m \) each containing a unique set with \( B \) distinct elements \( \{x'_1, \ldots, x'_m\} \) which need to be fissioned.

By Proposition 4, obtaining an optimal move sequence for instance \( S(n, k) \) requires finding an optimal pairing of the \( C_1, \ldots, C_{3m} \) and \( C'_1, \ldots, C'_m \) components of the instance. This is a pairing which makes the maximum number of inter-component translocations yet which, at the same time, translocates each of \( C_1, \ldots, C_{3m} \) with the minimum number of \( C'_j \) components for \( 1 \leq j \leq m \). Since the number \( s(C_i) \) of elements in each component \( C_i \) is such that \( B/4 < s(C_i) < B/2 \), the minimum will be reached if we can translocate exactly \( 3 \) components among \( C_1, \ldots, C_{3m} \) with each component \( C'_1, \ldots, C'_m \). Finding whether this is realisable requires solving the 3-Partition problem. This ends the proof. \( \square \)

Obviously, being the \( \mathcal{NP} \)-hard instance of theorem 5 a special case on multicomponent synteny, then we have that this latter is also \( \mathcal{NP} \)-hard.

3.5 Nested synteny

3.5.1 General observations

Let \( \mathcal{N}(n, k) \) be a nested synteny instance. In [LNK00a], Liben-Nowell and Kleinberg optimally solved such an instance in the case of a linear move sequence by using a job-scheduling algorithm [AWK78] as a black box in their own program. Their idea is the following. Let us consider a single component nested synteny instance \( \mathcal{N}(n, k) \). Let us denote by \( \bar{D}(\mathcal{N}(n, k)) \) the length of the optimal linear move sequence for \( \mathcal{N}(n, k) \). They first prove that \( \bar{D}(\mathcal{N}(n, k)) = \max\{n, k\} + d - 1 \) where \( d \) is the minimum number of elements that must be added to the biggest set in the component in order to obtain a linear exact instance. This result deals with the general case where we may also have \( k > n \). To facilitate, from now on we shall assume that \( k \leq n \). Finding a linear solution for \( \mathcal{N}(n, k) \), that is calculating \( \bar{D}(\mathcal{N}(n, k)) \), requires therefore only finding the value of \( d \). They further prove that the instance will be linear exact if, and only if, the graph model of the instance which they give as input to the job-scheduling algorithm results in an output that is equal to 0. The value of \( d \) is determined by performing a binary search between 0 and \( k \) (we never
3.5. Nested synteny

need more than \( k \) elements), interrogating the job-scheduling algorithm each time. The case where the instance is not linear exact is easy to determine because too many or too few extra elements were given to it. The job-scheduling algorithm serves therefore as an oracle answering the question: *is the instance linear exact?* However, such algorithm had been initially elaborated for solving a more general problem in another area. Therefore, we introduce below a much simpler algorithm that enables to answer the same question (i.e. "Is the instance linear exact?") in a more efficient way. Later, we suggest how Liben-Nowell and Kleinberg’s idea for the linear case could be extended in polynomial time to address the case of a connected general (that is, not necessarily linear) nested synteny. We conjecture the proposed approach is optimal.

3.5.2 Exact nested synteny

Tree representation of the problem

Let us consider a directed weighted tree \( T = (V \cup V', E \cup E') \). Nodes \( v \in V \) are the \( n \) sets \( S_i \) in \( N(n, k) \). Nodes \( v' \in V' \) are dummy nodes whose purpose will appear clear in a short while.

An edge \( e \in E \) leaves a node \( S_i \) (we shall describe below where it leads to) if there exists at least one \( S_j \) such that either of the two is satisfied:

- \( S_i \supset S_j \) and there exists no set \( S_k \) such that \( S_i \supset S_k \supset S_j \);
- \( S_i = S_j \) and \( i < j \) and there exists no \( i < k < j \) such that \( S_i = S_k \).

If there exists only one \( S_j \) satisfying one of the above conditions in relation to \( S_i \), edge \( e \) links \( S_i \) directly to \( S_j \). If there exists more than one \( S_j \), we add a dummy node \( v' \in V' \), and edge \( e \) will now link node \( S_i \) to \( v' \). We also place an edge \( e' \in E' \) from \( v' \) to node \( S_j \) for all \( S_j \)'s satisfying either condition.

Since this is a nested instance, \( T \) is a tree if there is only one component, or a forest otherwise. Let us call \( r_1, \ldots, r_c \) the roots of the forest for \( c \) the number of components. Each edge \( e \in E \) connecting a set \( S_i \) directly to a set \( S_j \) is weighted by an integer, namely \( w_e = |S_i - S_j| - 1 \). Edges \( e \in E \) leading to a dummy node are weighted \( w_e = |S_i \cup S_j| - 1 \) for all \( S_j \) satisfying one of the two subset conditions given above. Finally, edges \( e' \in E' \) have weight zero. An example is given in Figure 1.

We now enrich the tree by placing some information on the nodes in \( V \).

**Definition 6** Let us call frontier node, a node \( v \in V \) such that the edge leading to \( v \) is not negative while the one leaving \( v \) is less than zero. We shall also consider all leaves as frontier nodes.

To each frontier node \( v \in V \) is attached a pair of integers \((v_b, v_t)\) (\( b \) for "bad" and \( t \) for "total") such that:
Figure 3.1: Tree representation of a nested instance of the synteny problem.

- \( v_t = \max\{\sum w_e, 0\} \) for all \( w_e \) (positive and negative) along the path from \( v \) to the son of its first ancestor that is also a frontier node; if there is no ancestor, then \( v_t = 0 \);
- \( v_b = (\sum w_e) \) for all \( w_e < 0 \) along the same path as above; if there is no ancestor or there is no negatively weighted edge \( e \), then \( v_b = 0 \).

Values \( v_b \) and \( v_t \) indicate, respectively, the number of moves that would solve no element and the number of elements whose count becomes one (this last number can never be negative) by performing an optimal move sequence between two frontier nodes along a same branch of \( \mathcal{T} \) (these are \( v \) and the first ancestor of \( v \) which is a frontier node). Such values are equivalent to the notion of profit in [LNK00a]. In the case of an exact synteny, the idea is to traverse the tree \( \mathcal{T} \) by stepping on frontier nodes only (in fact, \( \mathcal{T} \) could be contracted to contain only such nodes), possibly jumping from arc to arc, always choosing as next stepping stone the one that allows us to earn the maximum number of elements with count one (possibly none).

**Theorem 6** Building the tree \( \mathcal{T} \) for an instance \( \mathcal{N}(n, k) \) takes \( O(k^2 + kn) \) time and \( kn \) space.

**Proof** Let us assume that the input data is correct (that is, the nested condition is satisfied) and that the cardinalities of each set have been pre-calculated (this takes at most \( kn \) time). Building \( \mathcal{T} \) takes then at most \( k^2 \) time because each comparison of two sets can be done in constant time (using \( kn \) space – each set is represented
as an array of dimension $n$, or $k^2 \log n$ \(\text{-- each set is represented as a balanced tree}\). Indeed, for each set \(u\) in turn by order of decreasing cardinality (sorting the sets takes at most \(O(k \log k)\) time), we need to consider one of its elements only and ask whether it is in one of the nodes already in \(T\) starting with the root. If it is in one of the nodes, let us say \(v\), then \(u\) is either identical with \(v\) (just check cardinality), or it is included in it and we recursively compare \(u\) with the sons of \(v\). Once the tree \(T\) is built, adding information to the edges and nodes can be done by just traversing the tree depth-first, each operation taking constant time. \(\Box\)

Algorithm

Once we have represented the instance by means of a tree such as \(T\), an algorithm for optimally solving the exact nested synteny is the following:

1. \textbf{while} the root has at least one son \textbf{do}
2. \quad let \(w_e\) be the weight of the edge \(e\) leaving \(r\)
3. \quad if \(w_e < 0\) \textbf{then}
4. \quad \quad if find son \(v\) of \(r\) that has an element \(x\) whose count is 2 (there must be one if exact)
5. \quad \quad \quad translocate \(v\) with \(r\) solving \(x\)
6. \quad \quad attach node \(z\) (maybe dummy) that is son of \(v\) to \(r\)
7. \quad \quad \textbf{else} return: “this is not an exact instance of synteny”
8. \quad \textbf{else if} \(w_e \geq 0\) \textbf{then}
9. \quad \quad if find descendant frontier node \(v \in V\) with information \((v_b, v_t)\) such that
\[|v_b| \leq w_e\] and \(new_{w_e} = v_t\) is maximum (there must be one if exact)
10. \quad \quad if there are two or more with maximum value \(v_t\) \textbf{then}
11. \quad \quad \quad \textbf{repeat}
12. \quad \quad \quad \quad pick the one with a descendant frontier node \(u\) such that \(|v'_{b}^u| + |u_b| \leq w_e + v_t^u\)
13. \quad \quad \quad \quad and \(new_{w_e} = v_t^u + u_t\) is maximum where \(v'\) is the first frontier ancestor of \(u\)
14. \quad \quad \quad \quad until \(u\) is unique or all \(u'\)'s are leaves, in which case pick any of them
15. \quad \quad \quad \(v := u\)
16. \quad \quad translocate with \(r\) all sets \((v\) included) along the path from \(r\) to \(v\) (in order)
17. \quad \quad \quad preferentially solving an element with count 2, otherwise one with count 1
18. \quad \quad detach the node from \(V\) that is a first descendant of \(r\) and an ancestor of \(v\)
19. \quad \quad attach at its place node \(z\) that is son of \(v\)
20. \quad \quad if \(z\) is a dummy node \textbf{then}
21. \quad \quad \quad attach to a dummy node son of \(r\) the sons of \(z\)
22. \quad \quad \quad give weight \(new_{w_e}\) to the edge \(e\) from \(r\) to its son
23. \quad \quad \textbf{else} \(|v_b| > w_e\) return: “this is not an exact instance of synteny”

**Theorem 7** The algorithm has a time complexity in \(O(k \log k)\).

**Proof** Calculating the syntenic distance can be done in \(O(k \log k)\) time if we maintain priority queues for the frontier nodes and, for each frontier node, a pointer to
its position in the queue. We need in fact one queue for each possible distance between a frontier node $u$ and the root $r$ of the tree (where the distance corresponds to the number of frontier nodes in the unique path between $u$ and $r$). We need to update the queues for all nodes below $z$ in line 17. This can be done in $O(k \log k)$ time. Finding node $v$ in lines 9-13 with the priority queues takes the same time ($O(\log k)$ per queue, and there can be up to $O(k)$ queues). A smart implementation of the tree allows to recover an optimal move sequence corresponding to the distance without increasing the asymptotic complexity of the algorithm. This can be done, for instance, by coloring at construction time the elements in the set corresponding to a node $u$ that do not belong to any of the sets associated with the children of $u$. The algorithm has therefore a time complexity in $O(k \log k)$. □

**Theorem 8** The algorithm solves the exact nested synteny optimally.

**Proof** If there are no dummy nodes, translocating the root with its unique son is the best we can do as was shown by Liben-Nowell and Kleinberg in [LNK00a], and indeed this is what the algorithm will do at each step. Suppose now that there are dummy nodes. Let us assume there is one dummy node $d$ just below the root $r$ (we can assume this without loss of generality since if this was not the case, we know how we can reach this situation by doing translocations between the root and its unique son as indicated above). Let $w_e$ be the weight of the arc between $r$ and $d$. We must consider two cases:

- $w_e < 0$
  - all elements present in the root are also present in one of its sons, and at least one of these elements must have count at most 2 (otherwise the instance would not be exact). The algorithm solves each of these elements in turn and optimality is preserved.

- $w_e \geq 0$
  - Let $v_1, v_2, \ldots, v_k$ be the first frontiers node that are descendants of $r$ (there may be just two). Since none of them has a frontier node for ancestor, all have $(v_b, v_t) = (0, 0)$. Whether line 10 of the algorithm is executed or not, for all nodes $u$ until the last, condition $|v'_b| + |u_b| \leq w_e + v'_t$ (let us call it $\alpha$) will be satisfied and $u_t$ will always be maximum. After $u$ is identified (and assigned to $v$), we translocate with $r$ all nodes in the path from $r$ to $v$, each time solving one element. This is possible thanks to condition $\alpha$. Clearly, solving first any element with count equal to 2 preserves optimality, as does solving any element having count equal to 1 if none with count 2 exists. Moreover, since the $v$ chosen had maximum value $v_t$, in the end, the edge leaving $r$ will receive a value $new_{w_e}$ that is also maximum. This means that all the other moves we could have done instead remain feasible. □
3.5.3 Linear nested synteny

Using the same idea as in Liben-Nowell and Kleinberg (sketched in section 3.5.1), we can employ the above algorithm as a black box for optimally solving the linear nested synteny in $O(k^2 + kn + c k \log^2 k)$ time where $c$ is the number of components. We recall that the $(k^2 + kn)$ term comes from the construction of $T$, which needs to be done once only. We remind also that the optimal linear move sequence in the case of multiple components is defined as just the union of all local optimal move sequences. The instance will be linear exact if the algorithm runs to conclusion, possibly leaving some sets which need to be fissioned (unless the instance is exact which may be the case only if $n = k$). However, these sets must not contain any of the extra elements given to it. If any one does, this means we utilized too many extra elements. If, on the other hand, the algorithm is not able to run to conclusion (because at some point we have that either $w_e < 0$ or $|v_h| > w_e$), this means we need more extra elements. These two observations are what allow us to search for the value of $d$ such that $\bar{D}(N(n, k)) = \max\{n, k\} + d - 1$ in a dichotomous way.

3.5.4 General nested synteny

Compulsory translocations

We show now how the above idea can be extended to deal with a general nested synteny. We use for this the concept of good and bad moves. The difficulty in calculating $D(N(n, k))$ without the linearity constraint comes from the fact that when, at a given step, we meet the situation where the only moves we can make are bad ones (basically in our model, either when $w_e < 0$ and there is no element whose count is 2, or when $|v_h| > w_e \geq 0$), then we have no means to decide which bad move we should choose without trying all possible complete move sequences. This does not happen when all moves until the next frontier node are good, that is, each solves an element as it happens with an exact instance of synteny. In that case, it is easy to decide which is the best move to make at any given step as was shown above. Liben-Nowell and Kleinberg’s nice idea for the linear case is to create a situation where we have to face good moves only. This is done through the use of extra elements which allow to transform the bad moves (in the case, the fusion operations) into translocations. The solution requires just finding how many extra elements are required. Fusions are the only bad moves (as defined above) one can make in the linear case. The situation of a general nested synteny is more complicated in the sense there are now three possible bad moves one can make. These are:

- translocations that cannot be substituted by a fusion otherwise the move sequence will not be optimal anymore;
- translocations that can be replaced by fusions without affecting the total number of operations needed to solve the instance;
• fusions that could be replaced by translocations without affecting the total number of operations needed to solve the instance.

Two examples of the first type are given by the diameter $\mathcal{K}(n, n)$ for $n \geq 4$ and by the instance in $\mathcal{N}(8, 8)$ composed of the sets $\{1, \ldots, 8\}, \{1, \ldots, 8\}, \{1, \ldots, 8\}, \{1, \ldots, 8\}, \{1, \ldots, 4\}, \{1, \ldots, 4\}, \{5, \ldots, 8\}, \{5, \ldots, 8\}$. The two instances can be solved optimally by using, respectively, four and ten translocations. Among these translocations, respectively two and six solve no element. However, if extra elements were added to the instances to transform such translocations that do not solve an element into ones that do, we would not obtain an optimal move sequence. In fact, these translocations have the property that they cannot be replaced by fusions. We call them compulsory translocations. We say that a $\mathcal{S}(n, k)$ instance is an exact diameter instance if any optimal move sequence contains only translocations that solve an element and compulsory translocations (corresponding to the first type above). On the other hand, there are translocations that solve no element and could be replaced by fusions. Such translocations, as well as all fusions, can be replaced by translocations solving extra elements given to the original instance without increasing the number of steps needed to arrive at a solution. Having those extra elements can only enable us (if we have enough of them) to decide, as for the linear case, what is the next move to do. Deciding which translocations are of the first type, that is, compulsory, and which are not is one of the main problems we must therefore treat. We know how to address it in the case of a diameter instance, $\mathcal{K}(n, n)$, and of a bidimensional diameter instance, $\mathcal{K}(n, m)$. We consider next the general case.

Instance $\mathcal{K}(n, m)$ with descendants

The general case of a nested synteny corresponds in fact to a $\mathcal{K}(n, m)$ with descendants, that is to a $\mathcal{K}(n, m)$ which contains also one or more subsets of $\{1, 2, \ldots, n\}$ (we call children), plus possibly proper subsets or twins of these (we call them grandchildren), and so on. We denote such cases by $\mathcal{K}(n, m, c)$ where $c$ indicates the number of distinct maximal proper subsets of the $m$ sets $\{1, 2, \ldots, n\}$. In the tree $\mathcal{T}$ associated with $\mathcal{K}(n, m, c)$, we have $m$ copies of the root, of which the lowest one has either $c = 1$ son or is connected to a dummy node having $c > 1$ children (observe that each such child is itself the root of a $\mathcal{K}(n', m', c')$ instance for appropriate values of $n', m', c'$). If a linear move sequence is considered, then just fusing at least $m - 2$ times is the only thing required, as until then no element can be solved and thus no translocations performed. We suggest that in general a shorter (no longer linear) move sequence may be found. We conjecture such sequence is optimal.

As an example, consider the instance $\{\{1, 2, 3, 4\}, \{1, 2, 3, 4\}, \{1, 2\}, \{3, 4\}\}$. No element has count 2 or less, so a linear move sequence would require a fusion as a first step. Assume that the fusion involves the two identical sets (we shall
see in Proposition 6 that this is a reasonable assumption), so that we are left with the three sets \( \{1, 2, 3, 4\}, \{1, 2\}, \{3, 4\} \). From now on, all elements can be solved, but at least 3 more moves are necessary to terminate, leading to a total of 4 moves. If linearity had not been required, the first move could have been the translocation \( \{1, 2, 3, 4\}, \{1, 2, 3, 4\} \to \{1, 2\}, \{3, 4\} \), resulting in the four sets \( \{1, 2\}, \{1, 2\}, \{3, 4\}, \{3, 4\} \). Two more translocations \( \{1, 2\}, \{1, 2\} \to \{1\}, \{2\} \) and \( \{3, 4\}, \{3, 4\} \to \{3\}, \{4\} \) would then have been enough to terminate, leading to a total of 3 moves only. This latter is the optimal (not linear) move sequence. Observe that the translocation we did represents a compulsory one. Intuitively, the reason for which performing first a translocation may result in a shorter solution is that the translocation can split a set into two, which correspond to the children. The problem is thus split into two subcomponents.

Let us consider another example. This is represented by the instance composed of twice the set \( \{1, 2, \ldots, n\} \) plus sets that are children of this one and where all elements have count 3 (hence the union of the children equals the father and there are no grandchildren). If we start by performing a fusion, \( n-1 \) more moves will be required to solve the instance. Indeed, at each step after the fusion, only at most one element can be solved, until the last move that solves two of them. If, instead, we start by splitting the instance into two components, performing for this the move \( \{1, 2, \ldots, n\}, \{1, 2, \ldots, n\} \to \{1, 2, \ldots, h\}, \{h+1, \ldots, n\} \) with a suitable choice for \( h \), the children can be partitioned into those that are children of \( \{1, 2, \ldots, h\} \) and those that are children of \( \{h+1, \ldots, n\} \) (this is always feasible after possibly renumbering the elements). After such a move, we are left with two separate components, consisting respectively of \( h \) and \( n-h \) elements. Each one is of the same type we would have obtained if we had performed a fusion: a father with children that are a partition of the father, and no grandchildren. However, only \( (h-1)+(n-h-1)=n-2 \) moves are required now to finish solving the instance. This is one move less than if we had started with a fusion. Splitting the instance into two subcomponents therefore helps. Similarly, if \( m \) in \( K(n, m, c) \) is greater than 4, then splitting the problem into 4 subcomponents (performing only \( m-4 \) fusions followed by \( 4 \) translocations), will in some cases represent an even better choice. An example is the instance \( \{\{1, \ldots, 8\}, \{1, \ldots, 8\}, \{1, \ldots, 8\}, \{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}\} \).

In general, when there are descendants (that is \( c \neq 0 \)), the best will often be to solve \( K(n, m, c) \) by starting to transform the \( m \) copies of the set \( \{1, \ldots, n\} \) into a suitable \( d \)-partition of it. The instance is thus split into \( d \) subcomponents. We shall see that the value of \( d \) as well as the choice of which elements go into which partition depends upon the value of \( m \), the number of children \( c \), and whether there are grandchildren or not. Before examining this, we start by giving some intuition for the conjecture that we shall state later. First, we need to prove a proposition.

**Proposition 5** Transforming \( m \) copies of the set \( \{1, \ldots, n\} \) into a \( d \)-partition of
\{1, \ldots, n\} \text{ takes } m + d - 4 \text{ moves for all } m, d \geq 4.

**Proof** Let \( S_1, \ldots, S_d \) be the \( d \) sets we wish to produce. These sets must be pairwise disjoint and their union give \( \{1, \ldots, n\} \). Rename \( x_i \) all elements that end up in set \( S_i \) for all \( i = 1 \ldots d \). Such a renaming transforms the problem into that of solving a \( \mathcal{K}(d, m) \) instance, which we know requires \( m + d - 4 \) moves (Lemma 3). These moves are those needed to perform the transformation we wish to obtain. If we could make it with less moves, by applying the renaming backwards we would be able to find a better than \( m + d - 4 \) solution for \( \mathcal{K}(d, m) \), which is a contradiction. \( \square \)

Let us now consider as above the simple case of a \( \mathcal{K}(n, m, c) \) instance with \( m \) equal to 2 and where each element has count exactly 3. We have seen two examples where the best to do is to split the instance into, respectively, 2 and 4 subcomponents. It can be checked that splitting into three subcomponents is not convenient, even if there are 3 children. This is because the split into 3 subcomponents can never be done in an efficient way. The reason is the same for which the size of \( 2n - 4 \) for the diameter does not hold for \( n = 3 \). If there are more than 4 children, then splitting the instance according to the number of children (and therefore in more than 4 subcomponents) does not help either: the move gained because of the creation of one more component (as a result of a translocation which solves two elements simultaneously) is lost during the splitting process, as shown by the Proposition 5.

Suppose now that \( m \) is 2 in the instance \( \mathcal{K}(n, m, c) \) but that the union of the children represents this time only a proper subset of the father set. In this case there are, say, \( f \) elements that have count 2 that the split process will have to assign to one of the new subcomponents. After the split, such elements have count 1. If all the other \( n - f \) elements have initial count 3 (which will become 2 after the split), that is, if there are no grandchildren, then it does not matter to which subcomponent the elements with initial count 2 are assigned after the split process. Otherwise, that is, in the general case where there are grandchildren in the instance \( \mathcal{K}(n, m, c) \), then it is not indifferent to which subcomponent will go the elements that, after the split, will get a count of 1. The reason is that such elements may prove useful in the same way as happened in inter-component moves as seen in section 3.4. Indeed, if they are assigned to a subcomponent that will not need them, additional operations will be required even if inter-component moves are later performed. Furthermore, we have seen that deciding which inter-component moves should be done in an optimal move sequence is an \( \mathcal{NP} \)-hard problem. The best to do seems therefore to split from the instance only those subcomponents which represent exact diameter instances and to leave all the other elements, including the \( f \) which will have count 1 after the split, into a single remaining subcomponent. These observations are the motivation for the following conjecture.
Conjecture 1  Let $S_1, S_2, \ldots, S_c$ be the $c$ children in a $K(n,m,c)$ instance for $n, m \geq 4$ and let $d$ be the number of sets into which $\{1, 2, \ldots, n\}$ could be partitioned. We have that:

if $c \geq 3$ then $d = \min \{1, \text{number of children that in } T \text{ become, with the split part containing them, an exact diameter instance} + 1\}$, and

if $c = 1(2)$ then $d = \min \{1(2), \text{number of children that in } G \text{ become, with the split part containing them, an exact diameter instance} + 1\}$.

where we call split part of the root $S$ of a $K(n,m,c)$ instance, one of the sets into which $S$ is partitioned. In both cases, the additional 1 stands for the component that does not correspond to an exact diameter.

As a consequence, we have that in an optimal move sequence for $K(n,m,c)$ with $n, m \geq 4$ and $c \geq 1$, we need at least 4 translocations if $c \geq 3$ and at least 2 otherwise ($c = 1, 2$).

We now have all the tools we need for suggesting an algorithm for the general nested synteny. We merge the results concerning the diameters with a modified version of the black box for the exact nested case in order to obtain a general (that is, not linear) solution. We use the same idea as for solving a linear nested synteny (section 3.5.3), that is, we determine by a binary search between 0 and $k$ the minimum number of extra elements we must give to the instance to make it, not an exact instance this time, but an exact diameter instance. The main difference is that, when meeting a $K(n,m)$ or $K(n,m,c)$ instance, the extra elements must be used only where a fusion would have been possible or needed, never where a translocation was absolutely required (see Theorem 3 and the analysis of $K(n,m,c)$ instances). Lines 7 and 21 are the only things that change in the algorithm given in section 3.5.2. They become:

7.  else (at this point, we have a $K(n,m,c)$ instance)
   7a.  if $(m > 4) \text{ or } (2 < m \leq 4 \text{ and } c = 1, 2)$ then return:
   “this is not an exact diameter instance”
   7b.  else choose the next set of compulsory translocations (following Lemma 1) as the one which leads to the maximum $v_t$
   (iterate if there are more than two)

21.  else (we have a $K(n,m,c)$ instance with $m = (|v_0| - w_e) > 0$)
   21a.  if $(m > 4) \text{ or } (2 < m \leq 4 \text{ and } c = 1, 2)$ then return:
   “this is not an exact diameter instance”
   21b.  else choose the next set of compulsory translocations (following conjecture 1) as the one which leads to the maximum $v_t$
   (iterate if there are more than two)
With these additions, we obtain a new black box which allows us to determine, by a binary search, the number of moves in what we conjecture is an optimal move sequence for solving a connected general nested synteny. This is the sum of the number of “good” translocations (which solve an element from the original instance), the number of compulsory translocations, the minimum number of translocations that solve an extra element and, finally, the number of fissions involving only elements from the original instance.

3.6 An inclusion theorem for the general case

In the case of a general synteny, sets are not necessarily nested. For some pairs \( (S_i, S_j) \) however, we may still have that either \( S_i \subseteq S_j \) or \( S_j \subseteq S_i \). The following result states that also in the general synteny case, there always exists an optimal move sequence that treats supersets first.

**Proposition 6** If \( S(n, k) = S_1, S_2, \ldots, S_k \) and \( S_i \subseteq S_j \) with \( S_i \neq \emptyset \), then there exists an optimal move sequence in which \( S_j \) is used before \( S_i \).

**Proof** If \( S_i = S_j \), then the theorem is trivially true. Let us now consider the case where \( S_i \subsetneq S_j \). Let a canonical optimal move sequence of \( S(n, k) \) be \( \sigma = \sigma_1, \sigma_2, \ldots, \sigma_p, \ldots, \sigma_q, \ldots, \sigma_m \). Let \( \sigma_p \) (resp. \( \sigma_q \)) be the first move using \( S_i \) (resp. \( S_j \)) with \( p < q \) (if such a move sequence does not exist, then we are done). We show that we can modify \( \sigma \) in such a way that we obtain a move sequence \( \sigma' \) that has also length \( m \) and uses \( S_j \) before \( S_i \). We have to treat only the cases where \( \sigma_p \) and \( \sigma_q \) are both fusions, or \( \sigma_q \) is a translocation and \( \sigma_p \) is either a translocation or a fusion. Indeed, if \( \sigma_q \) were a fission, we would have that the first operation made on the set \( S_j \) would be to fission at least one of its elements, that is to solve it. However, this contradicts the hypothesis that there exists the set \( S_i \neq S_j \) sharing some elements with \( S_j \).

Let \( \sigma_h \) be the first move of the original move sequence where products of \( \sigma_p \) are involved in the same operation as products of \( \sigma_q \). Notice that \( \sigma_h \) has to be either a fusion or a translocation because it takes two sets as an input. Notice that \( h \geq q \). We have to consider two cases.

1. \( h = q \)

   This means that the move \( \sigma_q \) involves \( S_j \) (which until then remains untouched and cannot therefore be a product of \( \sigma_p \)), and another set, say \( T \), that is a product of \( \sigma_p \), directly or indirectly (that is, is a product of products of \( \sigma_p \) or etc).

   If \( \sigma_p \) is a fusion, let \( S \) be the set such that \( \sigma_p \) is the operation \( (S, S_i) \rightarrow U = S \cup S_i \). The question is whether this step can be replaced with \( \sigma_p' : (S, S_j) \rightarrow U' = S \cup S_j \) (leaving \( S_i \) instead of \( S_j \) untouched). We have that \( U' = U \cup S_j \),
that is, it is $\mathcal{U}$ with additional elements that were in $S_j$ but not in $S_i$ nor in $S$. Let $\mathcal{W} = (S_j \setminus S_i) \setminus S$. We have $\mathcal{U}' = \mathcal{U} \cup \mathcal{W}$. We assume $\mathcal{W} \neq \emptyset$, otherwise the problem is trivial. Any other possible fusion that followed $\sigma_p$ and preceded $\sigma_q$ in the original move sequence, and that used $\mathcal{U}$, can be replaced with a fusion that uses $\mathcal{U}'$ instead. For simplicity, we name again $\mathcal{U}'$ the result of these possible further fusions: in all these cases, we are talking about a set that in the original move sequence was a set $\mathcal{U}$, and that now contains also $\mathcal{W}$, resulting in something that we keep on naming $\mathcal{U}'$. If there were some translocations in $\sigma$ between move $\sigma_p$ and move $\sigma_q$ that used $\mathcal{U}$, this would result in the same case as we treat next.

If $\sigma_p$ is a translocation, we may start by observing that $\sigma_q$ is also a translocation. Let us call $\mathcal{U}$ and $\mathcal{X}$ the two sets resulting from the translocation $\sigma_p$. If by $\mathcal{S}$ we denote the set with which $S_i$ is now translocated in $\sigma_p$, the sets $\mathcal{U}$ and $\mathcal{X}$ form a partition of the set $\mathcal{S} \cup S_i$ so that, if we use $S_j$ instead of $S_i$, we just have a bigger set $\mathcal{S} \cup S_j$ to partition. This is what we do in the operation $\sigma_p'$ which again replaces $\sigma_p$. Let us assume further that the extra elements $\mathcal{W}' = (S_j \setminus S_i) \setminus \mathcal{S}$ land all together in one of the sets resulting from $\sigma_p'$, that is, $\mathcal{W}$ is not split by $\sigma_p'$ (note that both $\mathcal{S}$ and $\mathcal{W}$ may be different from those of the previous case but since they represent analogous situations for the purpose of this proof, we use the same names). Let $\mathcal{U}'$ be the set such that $\mathcal{U}' = \mathcal{U} \cup \mathcal{W}$ where $\mathcal{U}$ was the set resulting from $\sigma_p'$. Again here, any other possible translocation that followed $\sigma_p$ and preceded $\sigma_q$ in the original move sequence, and that used $\mathcal{U}$, can be replaced with a fusion that uses $\mathcal{U}'$ instead.

The case of a translocation is thus similar to the case of a fusion.

We show now how in both cases (whether $\sigma_p$ was a fusion or a translocation), optimality is preserved when using $\sigma_p'$ rather than $\sigma_p$, provided we suitably modify also move $\sigma_q$ (which coincides with $\sigma_h$) in the following way. We have observed above that, if $h = q$, then we have that the set $\mathcal{T}$ such that $\sigma_q$ took as input $S_j$ and $\mathcal{T}$ must be a product of $\sigma_p$. It is therefore enough to place $\mathcal{W}$ in $\mathcal{T}$ and leave it there ($\mathcal{T}$ is thus what we called set $\mathcal{U}$, and $\mathcal{U}'$ is the union of $\mathcal{T}$ and $\mathcal{W}$), and to replace $\sigma_q$ (which took $S_j$ and $\mathcal{T}$ as input) with the move $\sigma_q'$ which takes $S_i$ and $\mathcal{U}' = \mathcal{T} \cup \mathcal{W}$ instead. Since $S_j \cup \mathcal{T} = S_j \cup \mathcal{U} = S_i \cup \mathcal{U}'$, the result of $\sigma_q'$ can be the same as the result of $\sigma_q$, and hence the remaining of the move sequence can remain as it was.

$h > q$

Let us call again $\mathcal{T}$ the set with which $S_j$ is either fused or translocated in move $\sigma_q$. If $h > q$, we have that move $\sigma_q$ of the original sequence does not correspond to an operation between a product (or product of products) of $\sigma_p$ and $S_j$. Set $\mathcal{T}$ is therefore either also untouched from the beginning (like $S_j$), or it is among the products of other moves $\sigma_1, \ldots, \sigma_t$ with $1 \leq t_i < q$ and $t_i \neq p$ that are completely independent from $\sigma_p$. We can therefore build a new
move sequence \( \sigma' \) which is equal to \( \sigma \) except that the order of the moves is changed in such a way that all the moves \( \sigma_{t_1}, \ldots, \sigma_{t_i} \) as well as move \( \sigma_q \) happen before \( \sigma_p \). In this way, we again obtain a move sequence that has the same length as \( \sigma \) and uses \( S_j \) before \( S_i \).

These transformations to the move sequence can be performed independently for all pairs \( S_i \) and \( S_j \) such that \( S_i \subset S_j \) and \( S_i \) is used first. None of these changes creates a new pair of the same type, hence the process will eventually finish and result in a move sequence which satisfies the desired property while remaining optimal. This proves the theorem. \( \square \)

### 3.7 Further work

Since, as we have shown in this chapter, finding the optimal pairing for multicomponent synteny is \( \mathcal{NP} \)-hard, the only possible solution when there are more than one component is an heuristic that attempts to approximate the optimal pairing. In general, given a multicomponent instance of synteny, we do not know the optimal canonical move sequence for solving the single components, as connected synteny is also \( \mathcal{NP} \)-hard ([DJK+97]). In particular, known approximation algorithms for syntenic distance, in general cannot give an a priori global view of how many fusions have to be done in total in a component. Nor we know how many fissions are required at the end for completing the solution of all elements of the component. For these reasons, in practice only a greedy approach seems feasible for finding the best inter-components. A possible future work is therefore to conceive a greedy algorithm to approximate the optimal components pairing problem introduced in section 3.4.3.
Chapter 4

PaTre: a tool for building paralogy trees

4.1 Introduction

Biological evolution has generated increasingly more complex organisms, starting from relatively simple ones. In order to obtain a complex organism, it is necessary to increase the global information content of a simple genome by means of new genes and the generations of new features and functions. Since the studies on the yeast genome [Han75], the duplication-with-modification paradigm has been considered to play a major role in the information increase phenomenon [Ohn70, Oht87], at least from a quantitative point of view. Indeed, often newly available complete genomes releases contain a representation of the genome in terms of clusters of paralogous genes. Disregarding the possible contribution of other mechanisms, in this chapter we assume that paralogous genes in a gene family have been generated by a iterated duplication-with-modification process only. With this assumption, we try to design a method to reconstruct the history of the family in terms of duplication events.

We are not aware of algorithms conceived specifically for this task without the constraint of the duplicated genes being in tandem. In [BD99, EGL02], two differenr parsimony based methods are presented for reconstructing the duplication history of tandemly repeated genes under different hypothesis. In [TWY01], another method for inferring tandem duplication events is presented. Differently from all these previous work, our method is not restricted to tandemly repeated genes because our data are the gene sequences and not their relative order or positions. Dating gene duplication [CDFC00b] and building gene trees [Pag98] are related but different works. The former aims to estimate duplication times, which is a more specific goal than ours, and makes use of evolutionary trees, which we do not assume to know. The latter also makes use of assumed phylogenetic information, and gene trees aims to represent relation among ortholog and not just paralog genes. Standard phylogeny
reconstruction methods are conceived to answer again a related but different question than the one we address in this chapter, and their application for our purpose does not seem to give satisfying results ([Tav00, TWY01]). Indeed, our approach is close to distance based parsimony methods for phylogeny reconstruction, but as we will see in the next section, its actions are driven from the specific problem, leading to a new ad hoc solution for paralogy tree reconstruction. The main novelties of our approach with respect to the general phylogeny reconstruction and the methods for reconstruct tandem duplications are two. The first stems from the non symmetry of the master-copy relation in the duplication event, as we need a measure to capture the different cost of generating a gene $a$ from another gene $b$ by a duplication of $a$ followed by mutations, rather than the opposite. For this reason, we chose to use the transformation distance which we describe in section 4.2.2. This choice will lead us to resort the Lightest Spanning Arborescence problem which we discuss in section 4.2.3. The second novelty is that our output is a (rooted) tree where the present time genes (which are the input) are not only placed as leaves, but they are also internal nodes. In other words, our paralogy tree has no steiner point. We are aware that this is an extreme choice, but on the other hand methods that output trees where all input sequences/genes result in leaves are extreme as well. With our new approach, we mean to suggest a possibly different hypothesis as starting point for further investigations.

Section 4.2 will describe and motivate more these two choices. Section 4.3 describes our method, called PaTre (Paralogy Trees), which is validated both with applications on synthetic (section 4.4) and real (section 4.5) data. PaTre has been implemented in C++ using the LEDA library (http://www.algorithmic-solutions.com/). In the next sections we describe this method, called PaTre (Paralogy Trees), evaluate its performance, and discuss some applications on real and on synthetic data.

4.2 Preliminaries

4.2.1 The problem: from paralogous genes to paralogy trees

Paralog genes can be defined as homologous genes in the same organism whose products perform related but not identical functions [Hin97], and are the results of gene duplications. According to an emerging biological paradigm [Ohn70, Oht87], gene duplication is the predominant mechanism for the evolution of new gene functions: once redundant gene copies are available, mutations may accumulate in one of the copies, thus creating a new gene, while the other one(s) can go on carrying out the original function. From this duplication with modification mechanism follows the interest of reconstructing the history of duplications and modifications within a family of paralog genes in an organism. Such a history can be represented as a directed tree
in which each node represents a paralog gene, and each (oriented) arc represents a duplication event in which the gene placed at the source node has plaied the role of the template, and the gene placed at the destination node is the (modified) copy. If a gene is placed at a node which is the source for more than one arc, then this means that that gene has been the template for more than one duplications. Our goal is, given a family of paralog genes, to reconstruct such a tree which we name paralogy tree.

4.2.2 The transformation “distance”

As we have introduced in section 1, we aim to capture the difference between the event of a gene \(a\) being the result of a duplication (with modification) of gene \(b\) and the event that \(b\) has duplicated generating \(a\). For this reason, we use the Transformation Distance \(TD(a, b)\) ([VDR99]), that has the property of not being symmetric\(^1\). The transformation distance (TD for short) has been introduced in [VDR99] as a measure of the cost of transforming a source sequence \(S\) into a target sequence \(T\) by means of the operations of: inserting a new segment, copying a segment from \(S\), and reverse copying a segment from \(S\). This requires a preprocessing phase to detect common segments between \(S\) and \(T\), possibly allowing for some differences among the occurrences of such segments. Therefore, in practice, point mutations that might have taken place in fragment of sequences that come from a common ancestor should not affect the detection of similarity among such fragments. Once the common segments are detected, \(TD\) actually measures the minimum weighted chain of segment translocations, insertions, duplications, and reversals that transform a sequence into another. Intuitively, the weight assigned to each one of these operations coincides with its description length. As a consequence, insertion has a high weight since the whole sequence to be inserted has to be described. The weight of the copy operation amounts instead to the cost of specifying the starting position of the fragment in the source sequence, and its length. Finally, the weight of the reverse copy is just one bit more expensive than the normal copy because the reversal information must be given. Notice that these choices for the operations weights agree with the energetic chemical costs of these mutation events, e.g. the insertion of a fragment into a DNA sequence requires an amount of energy that is proportional to the length of the fragment. As an example, in figure 4.1 we show a script that spells how a target sequence \(T\) is obtained from a source sequence \(S\), with minimum total weight\(^2\). The figure also shows the description length computed as in [VDR99]. The single operations are encoded as in [VDR99], with a code identifying the operation followed by its parameters. The description having minimum length

\(^1\)In general \(TD(a, b) \neq TD(b, a)\). In this sense the term “distance” is not proper. Nevertheless we will keep on addressing this measure as transformation distance as this is how it is known in the literature.

\(^2\)We have put identical occurrences of segments to simplify the example. Recall that, in general, point mutations are allowed.
Figure 4.1: A script that transforms a sequence S into a sequence T, and its description as in [VDR99].

is the transformation distance between S and T. The *copy* operation, for example, is described by its identification code and three parameters \((i, j, k)\), where \(i\) is the starting position in T where the segment is copied, \(j\) is the position of the fragment in S, and \(k\) is the length of the fragment. The reverse copy is similar, plus an additional bit-parameter whose presence indicates the reversal. The description of the insertion operation requires that the sequence to be inserted is given as a parameter. The transformation distance is not symmetric because, in general, a script describing a re-writing of T in terms of S is completely different from one describing S in terms of T. This is basically due to the fact that insertions and deletions are evaluated in a very different way, and thus the symmetry is lost. We will not explain in details how the transformation distance can be computed, but we just recall that this can be done in polynomial time\(^3\) in the length of the sequences. This is done representing all possible scripts as paths on a graph and then finding the shortest path. For details, we refer the interested reader to [VDR99]. We believe that the transformation distance \(TD(a,b)\) is a good measure of the cost of generating the gene \(b\) by means as a duplication with modification from gene \(a\). In fact, since our paralogy tree is built using a parsimony criteria on such distances, we have that a small value of, say, \(TD(g_i, g_j)\) is likely to lead to the presence of the arc \(g_i \rightarrow g_j\) in the paralogy tree, that is to suggest a duplication of \(g_i\) into \(g_j\) in the story of the family. The most fragments two genes share (possibly reversed and with some point mutations as differences), the smallest is the distance, which agrees with the hypothesis that duplicated genes have similar sequences. Moreover, if from gene \(g_i\) to gene \(g_j\) there are more deletions than insertions, then in general \(TD(g_i, g_j) > TD(g_j, g_i)\). As a consequence, the parsimony would rather designate \(g_i\) as a template for the duplication that generated \(g_j\) than the opposite. This agrees with the belief, which we share, that actually during and, above all, after the duplication event the new gene undergoes deletions with respect to the template gene. In fact, it has been observed that duplication events can have involved big fragments (containing several genes or even the whole genome), but that present time consequences in terms of sequence similarities are evident only in some parts [WS97]. Finally, the three

\(^3\)Computing the transformation distance takes \(O(n^6)\) time where \(n\) is the length of the longest sequence. Note this is a worst case complexity. In practice, the method is applicable to input up to 100kb [VDR99], and over this threshold, it is its space costs that makes it unusable.
operations considered in the transformation distance and the corresponding weights ensure that the distance between \( A \) and \( B \) grows when successive segment reversals, segment insertions, and transpositions take place. In genome rearrangement literature ([Chr98, BKS96]) these operations are the most acknowledged candidates for segment level mutations. Concerning the computational complexity, we have that the computation of no other distance that takes into account these events results in a computationally tractable problem.

### 4.2.3 The lightest spanning arborescence problem

Given a directed graph \( \mathcal{G} \) with weighted edges, and a distinguished node \( r \) designated as the root, the lightest spanning arborescence problem asks for the minimum cost tree rooted in \( r \) that spans all the vertices of \( \mathcal{G} \). This problem has been solved in [Edm67] in \( O(n^3) \) time, where \( n \) is the number of nodes of \( \mathcal{G} \). For the case of a complete graph \( \mathcal{G} \), a modification of Edmonds’ algorithm with better practical performances was suggested in [FT93], and slightly more efficient algorithms have been more recently conceived for the general case [AM01, GGST86, Tar77] as well. However, as we will see later, the tree computation time is not a critical step in PaTre tool because here the number of vertices of \( \mathcal{G} \) is at most a few tens. Hence, in our implementation, we adopted Edmonds’ algorithm with the variation suggested in [FT93]. Our choice of the root is explained in section 4.3.

### 4.3 Our proposal: the PaTre tool

Given a family of \( n \) paralog genes \( g_1, \ldots, g_n \), in order to construct a paralogy tree from this gene family, our tool takes in input the sequences corresponding to the \( n \) genes, and performs the following three main steps:

1. Compute \( TD(g_i, g_j) \) and \( TD(g_j, g_i) \) for each pair of genes \( g_i \) and \( g_j \).
2. Represent the result in the weighted complete directed graph \( TD\)-graph.
3. Compute the lightest spanning arborescence of the \( TD\)-graph.

In step 1 we compute the transformation distance between every ordered pair of genes belonging to the family of paralogs that we are willing to investigate. The distances information can be represented by a complete directed graph, the \( TD\)-graph, having one node for each gene \( g_i \), and an arc from \( g_i \) to \( g_j \) weighted \( TD(g_i, g_j) \) (step 2). Once the \( TD\)-graph is built, in step 3 we apply the algorithm that finds the lightest arborescence algorithm on it. We cannot use traditional algorithms solving the minimum spanning tree problem because our distance is not symmetrical, and therefore the resulting \( TD\)-graph is directed. Therefore, we apply the algorithm that finds the lightest arborescence algorithm on the graph. For the same reason (non
symmetry of the distance measure), traditional phylogeny tree inference methods based on distance matrices cannot be of any use here in order to build a tree.

Since there is one minimum weight arborescence for each possible fixed root, the computation in step 3 should be repeated $O(n)$ times, one for each possible gene as a root, to find the arborescence having minimum weight. This would result in a $O(n^4)$ time complexity algorithm using Edmonds’ approach [Edm67]. It should be pointed again that the value of $n$ is the cardinality of the paralogs gene family, which hardly is more than a few tens genes. This is why the computation of the lightest arborescence tree is not computationally critical. Nevertheless, in order to avoid the $O(n^4)$ time cost, we add a dummy node $r$ to the $TD$-graph, having $n$ arcs with very high weights leaving it, each one reaching one of the other $n$ nodes, and no arc entering it. The minimum weight spanning arborescence is computed using $r$ as a root, and the resulting tree will contain only one arc among those leaving $r$, due to the high weights given to these arcs. Therefore, we can just designate the sink of that arc as the root of the paralogy tree, and the tree obtained by pruning the dummy node is exactly the minimum weight arborescence of the $TD$-graph. This unique rooted tree is the one we suggest as a paralogy tree for the gene family under investigation. As already mentioned in section 1, the paralogy tree output by PaTre has no steiner point. That is, it is implicitly assumed that ancestors of present time genes are also present time genes. We are aware that this is not always true in real data, as there can be genes that have belonged to the family and plaied a role in its evolution before disappearing. We do not exclude that these possibly missing genes might also lead to an unreliable output paralogy tree. For this reason, we conceived a possible way to evaluate the reliability of our result by observing other solutions close to the optimal one, if any. We will see some examples in our experiments of section 4.5. Nevertheless, we observe that trees such as those output by traditional multiple alignments and phylogeny reconstruction methods where the input sequences end up in the leaves only, is an extreme case as well which is particularly not suitable for representing a paralogy tree. Indeed, we actually mean PaTre as a possible suggestion of a new and different view which has to be evaluated (like it is always the case in phylogeny reconstruction, and in section 4.5 we give some examples of such evaluation). Moreover, some studies [LC00] support the hypothesis that pseudo genes hardly disappear even if they remain silent. This would decrease the possibility of incompleteness of families of paralogs. Nevertheless, it remains open the not negligible probability that a family misses a gene because this has not been detected or classified in the proper family. Assuming that the family contains $N$ genes of length $n$, the worst case complexity of the algorithm is $O(N^2n^6)$, given by the cost of computing all pairwise transformation distance (the computation of the minimum arborescence takes $O(N^3)$) Notice that $N << n$, as the value of $N$ in our experiments went from a minimum of 4 to a maximum of 20, and their length (that is, $n$) can be up to a few hundreds bases. We observed that this worst case scenario is far from the behaviour of the method
on real data, which resulted applicable also for large data size (see section 4.5).

4.4 Experimental results based on synthetic data

In order to have a first experimental assessment of the soundness of our method, initial tests have been carried out on synthetic data, whose expected optimal tree is already known because it is the output of a simulator. The simulator is described in section 4.4.1, and a validation of it is given in section 4.4.2. The experimental results are presented in section 4.4.3.

4.4.1 The simulator

Our simulator of evolution generates a paralogy tree of a gene family starting from an arbitrary initial sequence. Starting from a given unique sequence, the simulator performs duplications and mutations according to the following model. A new sequence \( s_1 \) can be generated from a sequence \( s_0 \) because a few among the following events take place:

1. A fragment of \( s_0 \) is inserted in \( s_1 \).
2. A fragment of \( s_0 \) is reversed and inserted in \( s_1 \).
3. Single nucleotides are added or modified in \( s_1 \).
4. Fragments are inserted in \( s_1 \).

Notice that the insertion of fragments from sources other than \( s_0 \) corresponds to biological events such as horizontal transfers, which are segment-based mutations that are external to the paralogy relation. This is why in our simulator such events are introduced as random insertions. The simulator implements this model in several steps. First, a random number of random points are selected in \( s_0 \), thus actually defining fragments of \( s_0 \). Second, a random number of random sequences are generated. Third, a new sequence is generated using fragments of \( s_0 \) as they are selected at the first step, and the random ones generated at the second step. During this step, the fragments have a 50% probability to be reversed. Moreover, once \( s_1 \) is built, random point mutations are performed. This process is then iterated after a new sequence playing the role of \( s_0 \) (the template) is selected. These operations correspond to applying random insertions, reversal, and transposition on fragments, and random point mutations on all sequences. Our tool allows two possible ways to choose at each iteration the new sequence that is a template. The first way just uses an uniform probability, that is all genes have the same probability to be selected as a template for a new duplication. Otherwise, we can assume that during the evolution, the probability of selecting a gene as a template for a duplication decreases each time that the gene behaves as a template. In this way, the youngest genes
Figure 4.2: Example of a simulator output. The labels assigned to the nodes show the order of generation of the corresponding genes.

have a higher probability to generate a duplicate. This strategy is motivated by a qualitative observation that the trees generated with this probability distribution are more similar to those reconstructed on real biological sequences.

In order to test PaTre on simulated data, we first apply our simulator to a randomly generated initial gene, thus obtaining an arborescence $A$ like the one shown in Figure 4.2. Hence, we feed PaTre with the paralogous gene family consisting of the sequences associated with the nodes of $A$. What we then expect as a result from PaTre is exactly $A$, formerly obtained from the simulator.

4.4.2 Validating the simulator

In order to validate the reliability of the simulated generation of paralogous genes, we have designed a test which uses ClustalW\(^4\), a clustering algorithm. The tests work as follows. Given a family of paralogs, we randomly extract one of the genes and we give it as an input to the simulator, to which we ask to generate as many paralogs as the number of real genes in the family. Afterwards, we gather together real and simulated genes, and run ClustalW. Intuitively, the more faithful is the simulator to the real paralogous genes generation, the more it generates genes that are similar to those actually belonging to the family. In this case, what we expect from ClustalW

\(^4\)ClustalW ([THG94]) is a widely used tool for multiple alignment that, among other facilities, groups some given input sequences based on the results of their pairwise comparisons (using the method known as neighbor joining). The output clustering is given as a tree. The version we used is the one hosted by EBI, the European Bioinformatic Institute (http://www.ebi.ac.uk/clustalw/).
is an output that exactly pairs one real and one simulated gene. If this happens, we say that a *perfect mix* has taken place. More generally, we define a mixing index $\mu$ that measures how much real and simulated data have mixed. We compute it as follows. Let $k'$ be the number of internal nodes (that is without the leaves and the root) in the tree output by ClustalW. We set $\mu = k'/k$, where $k'$ is the number of internal nodes that have both real and simulated genes as descendants. It is worth observing that there is a large variety of families that show a $\mu$ index of 100%, that is real and simulated sequences perfectly mix. This means a good *mimetic* capability of the simulated data with respect to the real one. As a side result of this test, it is interesting to notice that the value of the parameter $\mu$ remains approximately constant when we explore the same family in different organisms, even if they are very distant in evolutionary terms.

### 4.4.3 Results on synthetic data

To test PaTre on simulated data we first apply our simulator to an initial gene, thus obtaining several sequences and an arborescence $\mathcal{A}$ that represents their duplication history. Hence, we feed PaTre with the paralog genes family consisting of the sequences associated with the nodes of $\mathcal{A}$. What we expect then as a result from PaTre is exactly $\mathcal{A}$, formerly obtained from the simulator. We performed this experiment for ten paralog genes family and the results showed that, on these data, PaTre is *always* able to reconstruct correctly the paralogy tree generated by the simulator. For these output trees we went to check whether possible solutions different from the optimum (that is other trees obtained from the *TD-graph* whose cost is higher than the optimum one) are very close to it and represent substantially different scenario. In fact, we believe that the existence of a tree whose cost is not far from the optimum and that results in a essentially different duplication history, would invalidate the result obtained with PaTre. For example, we have detected some of such cases in some of the experiments in which we fed PaTre with incomplete data with respect to the one generated by the simulator. Nevertheless, in our experiments on simulated data the output tree resulted always in a robust solution under this point of view.

### 4.5 Preliminary experimental results on real data

Before applying PaTre to real biological data, we would like to point out that in biology there are no direct evidences concerning either the history of a family of paralogs, or the loss of genes from a family. Therefore, the best evaluation of our method remains the test on simulated data. On the other hand, it is useful to discuss the results obtained by applying PaTre on real sequences in order to test the reliability of the detected duplication relationships.
4.5.1 Application to histone family in *Saccharomyces cerevisiae*

The histone family is a very conserved one as it consists of the same 4 or 5 genes in all organisms. Two of these genes, *ACH2A1* and *ACH2A2*, encode for the same protein. For our first application of PaTre we chose the histone family of *Saccharomyces cerevisiae*, which is composed of 4 genes. The resulting paralogy tree is shown is the left one of figure 4.3, and the best alternative solution, which costs 13% more, is the right one. There are no other solutions in the range of 25% of the optimum. This second solution arises because the costs of the arcs (*ACH2A2, SCH2B1*) and (*ACH2A1, SCH2B1*) are very similar, and thus the exchange of *ACH2A1* and *ACH2A2* increases the total cost with basically the difference between $TD(ACH2A1, ACH2A2)$ and $TD(ACH2A2, ACH2A1)$ only. Since *ACH2A1* and *ACH2A2* encode for the same protein, the two trees do not give such a different information. It is interesting to notice that the subtree *SCH2B1* → *SCHIS2* is confirmed.

![Paralogy tree](image)

Figure 4.3: Two paralogy trees for the histone family of *Saccharomyces cerevisiae*. The labels on the edges represent the transformation distances.

4.5.2 Some studies on a family of *Mycoplasma pneumoniae*

*Mycoplasma pneumoniae* is a bacterium whose complete genome sequence has been available for a few years now. We chose to work with the *heat shock protein* family, which contains eight genes. It should be complete because the genome has been
Figure 4.4: The paralogy tree output by PaTre for the 8 genes of the heat shock protein family of *Mycoplasma pneumoniae*.

sequenced completely and extensively studied.

The tree generated by PaTre is shown in Figure 4.5.2. The nodes are labeled with the names of the genes. The number that ends the name of the gene is related to the relative position of the gene, in the sense that consecutive numbers refer to adjacent genes. We can therefore notice that there are two adjacent genes in the family that appear in the lowest part of the tree, as well as a group of four consecutive ones that include the root. These are very likely duplicates of one another because duplications can happen *in tandem* (although this is not necessarily always the case), that is the copy is placed right next to the template. The PaTre tool does not have this information, that is there is no step in which it uses data regarding the position of the genes. Nevertheless, it does locate these genes close to one another in the tree. Finally, we have that the output tree is the only solution in a wide range (15%) around the value of the optimum.

4.5.3 The case of a higher organism: the plant *Arabidopsis thaliana*

The genome of *Arabidopsis thaliana* is the only complete genome of a plant available so far\(^5\). For this family we do know studies that formulate hypothesis of past duplications.

\(^5\)The complete sequence is available at http://www.arabidopsis.org.
cation events and hierarchical relation between presently existing genes ([Tav00]), and therefore we know what to expect as a reasonable result.

The SHAGGY/GSK-3 family

The Shaggy like kinase family in *Arabidopsis thaliana* (AtSK, see [TTT+98, TALK00]) is actually a sub-family of the GSK-3 one. The biochemical role of the proteins encoded by these genes is widely studied in mammals, while their function in plants is less clear. Nevertheless, it is known that the biological role of these proteins is very variable, and that this family seems to be very ancient. At present time, in *Arabidopsis thaliana* ten genes have been classified into this family. Therefore, we used for our applications ten genes, which are divided into four groups: AtSK-alpha, AtSK-gamma, and AtSK-epsilon of group I, AtSK-eta, AtSK-dzeta, and AtSK- iota of group II, AtSK-beta and AtSK-theta of group III, and finally AtSK-kappa and AtSK-delta of group IV. The partition into groups corresponds to acknowledged sequence similarities. These ten genes are arranged in five different chromosomes in *Arabidopsis thaliana*. Previous studies on this family (see [Tav00] for a review) have suggested models for its evolution that include the following duplications.

- In group II, AtSK-dzeta is ancestor of AtSK-iota by translocation of a fragment.
- In group II, AtSK-dzeta is also ancestor of AtSK-eta, by another translocation of another fragment.
- Within group I, AtSK-alpha is an ancestor of AtSK-gamma, again by translocation.

These assumptions are motivated by the observation of high similarities in the sequence and by the fact that in AtSK-dzeta (resp. AtSK-alpha) there is an additional intron with respect to AtSK-eta (resp. AtSK-gamma), which can be justified by a translocation event that moves a fragment thus splitting a coding fragment into two. The application of a phylogenetic reconstruction based on distances between the amino acid sequences of the corresponding proteins also suggests the following possible divergence by duplications.

- Genes AtSK-delta and AtSK-kappa (that is, group IV).
- Genes AtSK-beta and AtSK-theta (group III).
- Gene AtSK-gamma related to AtSK-epsilon, while the results above only mention the relation of the former with AtSK-alpha. These three genes compose group I.

In these cases the direction cannot be specified, as the common ancestor may not be among the existing genes.
Cost: 4430 - Distance: 0%

![Paralogy tree](image)

**Figure 4.5:** The paralogy tree output by PaTre for the SHAGGY/GSK-3 family in *Arabidopsis thaliana.*

**The paralogy tree output by PaTre**

First of all, we have measured the mixing rate as explained in section 4.4.2 between the SHAGGY/GSK-3 family, and the one data obtained by the simulator starting with one of the ten genes as initial sequence. The result is a 100% mixing rate. Secondly, we have applied PaTre giving as an input the DNA sequences of the genes. Figure 4.5.3 shows the paralogy tree output by PaTre.

We can immediately see that the four groups are well separated in the obtained tree. In a more detailed way, all the assumed duplications are represented in the tree as the corresponding genes are connected by a direct edge. Moreover AtSK-dzeta indeed results in an ancestor of AtSK-iota, as well as of AtSK-eta, as expected. The only difference with the assumptions is that AtSK-gamma is an ancestor of AtSK-alpha rather than the other way around. This can be explained by the fact that AtSK-gamma results acknowledged as a duplication of AtSK-epsilon (as expected), and eventually this result is incompatible with an edge directed from AtSK-alpha to AtSK-gamma. Within the range of 5% of the optimal tree (that costs 4430), there are in total 19 solutions.

Since we detected the presence of other solutions whose cost is close to the optimum, we went to explore further. In particular, the second best solution (shown in figure 4.6) deserves attention because it has a cost very close to the optimum (namely 4469, that is only within the 1%). Basically, only a subtree of AtSK-epsilon
changes, and the resulting tree maintains all the arcs representing the expected duplications. We believe that this is an interesting confirm.

Cost: 4469 - Distance: 8.803612E-1%

![Paralogy Tree Diagram]

Figure 4.6: The second best solution for the SHAGGY/GSK-3 family in *Arabidopsis thaliana*.

### 4.6 Further work

We plan to investigate further, by means of both simulations and experiments on real data, the possibility of guessing which arcs of the output tree should contain a steiner node in order to give a result more faithful to the truth. This could be done by looking at the cost of arcs.

As a further step in this project, we aim to compare paralogy trees of different families, both in the same genome and in different ones. In particular, as a new way of performing genome comparisons and phylogenetic reconstruction, it could be interesting to compare the shape (*i.e.*, some topological properties) of the paralogy tree of the same families in different organisms. For example, we could check whether certain organisms tend to have deeper trees, with respect to others, for the same family. With respect to traditional phylogenetic studies, instead of comparing single genes belonging to different organisms, we could compare entire paralogy trees for a given family of paralogous genes in different species. This approach could avoid many problems related to the choice of the gene and the relative cross-similarity between orthologs and paralogs. Moreover, it could give information about the
strategy of new genes creation. It could also address the question posed by the simulator structure, that is, whether the template for a new copy should be chosen among all the existing paralogs with uniform probability, or, on the contrary, the more recent genes have a higher probability to be chosen. Finally, within the same genome, it could be interesting to compare the shapes of different trees related to different families. This could help in current studies of the different duplication rates of genes.
4. PaTre: A tool for building paralogy trees
Chapter 5

Clustering families of paralogous genes by using random graphs

5.1 Introduction

We are interested in clustering gene families according to a variable range of sequence similarity, and then to try to detect biologically significant properties of such families by seeking for differences with respect to suitable random data. These families are meant to be genes within the same organism that are supposed to share a common origin, that is to be duplicated from a common ancestor. To this purpose, our data are all the genes of a complete genome, and a distance function between them. Within such a genome, the distance is computed for every pair of genes and thus similarities among them are detected when a certain distance is smaller than a given threshold. The resulting data, depending upon the threshold, can be represented by a graph (section 5.4.2). The usual approaches fix such a threshold and then study the properties of the resulting families of similar genes. This is not our case: we build the graph and study its properties for thresholds that cover all the ranges of connectivity of the graph (that correspond to all the degrees of similarity required for genes to end in the same family). Such a graph is in fact our subject of study, and its properties are investigated with respect to those of a corresponding random graph (we will summarize in section 5.4.3 the terms of such a comparison). Our basic idea is that the properties of the graph that models the biological data are interesting wherever they differ from the properties of random graphs. In the literature, there are several analyses where the biological data is compared with that obtained with a shuffled version of the genes. Nevertheless, the comparison takes place not at the graph level, but rather at the sequence level. Comparing to random data at the graph level is a novelty of our approach. To this purpose, a first aim of our work was to find a suitable model of random graphs having properties such that it is actually comparable to the graph representing the biological data. After a preliminary investigation (section 5.5), we chose a particular model of random
graphs that is detailed in section 5.5.4. This results in a much better model than the basic one, which, however, does not yet reach our goal. We plan in future to find a better model (see section 5.7). Nevertheless, section 5.6 describes the results of the application of the present method to a bacterial genome, *Mycoplasma genitalium*. After having briefly presented, in section 5.6.1, some characteristics of the organism, we compare the graphs representing the biological data for 100 values of the threshold, with graphs generated with all the models we mention in this work, including data resulting from the shuffling of the genome sequence. We then analyse the behaviour of one particular model for some particular thresholds that resulted in the detection of unusual properties of the graphs representing *Mycoplasma genitalium*. In fact, we computed numerical values for suitable differences between the graphs (section 5.6.2), and we investigated biological data when and where it significantly differs from the random model.

5.2 Generalities of the problem

5.2.1 The problem

As mentioned in chapter 2, gene duplication seems to play an important role in the evolution of each genome. If genes have duplicated in order to let the genome gain new functions, then we can assume that a classification of them which groups similar sequences could possibly correspond to families of inheritance. Thus, while listing such families, we actually produce sets of genes that share a common origin and that have been recently duplicated to let part of them evolve into new functions. Once we accept this, it is easy to understand why analysing such sets of recently duplicated genes provide interesting data (possibly allowing both to answer questions and, why not?, to formulate new hypotheses) about the way genes evolve, how their ordering and positioning changes along the chromosome, and many other properties that could be observed. Next to the study of evolution itself, grouping genes that have evolved from the same ancestor can also lead to some other kinds of useful results. Once we have classified a set of genes having a common origin (because their sequences share extensive homologies), if we knew well the proteins resulting from a subset of them, then we could infer information about the remaining ones of the family.

We fix our attention on a particular aspect of duplicated genes. In fact, our purpose is to simplify the analysis, considering only non-topological parameters, and then to focus on simpler but already significant aspects of genes families. More precisely, we decided not to take into account the relative order and positions of the genes, but rather to start by studying the number of different families, and the number of different genes they contain, which is related to their cardinalities. These
5.3 Some previous work

In [Oht87], a general model of evolution by gene duplication in eukaryotes is analyzed by simulating the evolution of a genetic system, starting from a single gene copy and following by the occurrence of several events with suitably maintained properties. The interest of such a study is to understand the origin of large multigene families with functionally different gene members in higher order organisms. The experimental results of this paper show how copy number, pseudogene number, and other parameters change with time, under different intensities of selection after a certain number of generations.

In [Wal95], the problem of the fate of a recently duplicated gene is addressed: it can either become a pseudogene, or it can mutate in such a way that it gives rise to a new function. The probabilities of these two events are estimated.

Computer simulations, together with analytic methods, are used in [SW98] to estimate some parameters describing the evolution of a yeast genome. This analysis considers in particular the ordering and other topological properties of the genome.

In [SMG+97], after a suitable clustering of genes of several different organisms\(^1\), two parameters are taken into account. The first one is \(S_n\), the total number of sequences that appear in clusters of \(n\) sequences, and the second is \(C_n\), the number of clusters of \(n\) sequences. By sequences here we mean gene sequences, also known as Open Reading Frames. Results of their pairwise alignments and subsequent clustering, show an unexpected regularity in the frequency of duplications:

\[
S_n = \frac{N}{2^{(n+1)}}
\]

and

\[
C_n = \frac{N}{n2^{(n+1)}}
\]

where \(N\) is the total number of sequences in a given genome. Thus 1/8 of the total number of genes in a given genome is present in clusters of two (and only two) homologous sequences (setting \(n = 2\)), 1/16 in clusters of three (and only three) homologous sequences (with \(n = 3\)), and so on. This paper is an example of an analysis of gene families, made with the assumption that the families are actually clusters, i.e., equivalence classes. This can only be achieved (as it is actually done in the paper) performing a transitive closure on the similarity relation. Once again our

\(^1\)Both clusters of paralogs (within the same genome) and orthologs (among different genomes) sequences are built, but here we report only the results concerning paralogous homologies.
choice here is different: we will not perform any transitive closure on the biological data.

5.4 The graph theory approach

5.4.1 The Levenshtein distance

It is now worth spending some words on the kind of biological data we will work on. We have all the amino acid sequences corresponding to the genes of the genome we investigate. In order to study homologous genes in a specific genome, it is necessary to choose a notion of similarity. Since, as we already stated, we consider a couple of genes to be similar if a given distance between them is smaller than a certain threshold, then we need to choose the distance we will use in this pairwise comparison of genes.

The comparison of biosequences has become one of the most important tools for molecular biologists. This explains the huge literature developed in recent years, devoted to sequence comparisons in general, and especially designed for molecules (see, e.g., [SK83, Wat95]). In particular, dynamic programming approaches for the alignment of sequences have been conceived and applied to biosequences. When comparing two sequences, one in general does not know which part of a sequence corresponds (if an analogy exists) to which part in the other one (two sequences in general may even have a different length) in order to obtain the highest similarity. This is the reason for talking about alignments: the goal is to seek along the sequences the “shift” (indeed, the best alignment) that maximizes a suitable score function (or minimizes a defined distance). For this purpose, dynamic programming algorithms\(^2\) work on a \(n \times m\) matrix (where \(n\) and \(m\) are the sizes of the two sequences to be aligned) that, after a suitable initialization of the first row and column, inductively compute the distance (to be minimized, or a score to be maximized) of the prefixes \(A_i\) and \(B_j\) (resp. of length \(i\) and \(j\)) of two sequences \(A\) and \(B\).

\[
d(A_i, B_j) = \min / \max \left\{ \begin{array}{l} d(A_{i-1}, B_{j-1}) + w(a_i, b_j) \\ d(A_i, B_j) + w_{DEL} \\ d(A_i, B_{j-1}) + w_{INS} \end{array} \right. 
\]

where \(a_i\) is the \(i-th\) element of the sequence \(A\) and \(b_j\) is the \(j-th\) of \(B\), and \(w(a_i, b_j)\) indicates the price to be paid for the substitution of \(a_i\) with \(b_j\) in the case that \(a_i\) and \(b_j\) are different amino acids, and the score gained in the case of a matching between them. Moreover, \(w_{INS}\) and \(w_{DEL}\) are the costs of, respectively, insertion and deletion of an amino acid.

\(^2\)More details on sequence alignments techniques can be found in [SK83, SM97, Gus97, CR94].
One of the simplest distance that has been suggested is the Levenshtein distance. We will use this to estimate the similarity of genes. In the recurrence formula we have just described, this corresponds to \( w_{\text{DEL}} = w_{\text{INS}} = 1 \) and \( w(a_i, b_j) = 0 \) if \( a_i = b_j \), and 1 otherwise. This means that a substitution, insertion and deletion, cost 1 in an alignment. The Levenshtein distance is a global distance in the sense that the aim is to entirely align both sequences so that if the lengths of \( A \) and \( B \) are different, this will cause a larger number of forced insertions or deletions. The internal similarities between one sequence and part of the other one will then be hardly detected.

### 5.4.2 The graph \( G_\Gamma \)

The graph \( G_\Gamma \) represents, in our study, the biological data. As we have already mentioned, such a graph is built for several different thresholds covering all possible ranges. What is obtained goes from a graph with almost no edges (i.e., only perfectly identical sequences are detected as similar) to a fully connected graph (i.e., all genes are similar to one another). Once we know the pairwise Levenshtein distance for all the genes, we construct the graph \( G_\Gamma \) with respect to the threshold \( \Gamma \). For each threshold \( \Gamma \), we build \( G_\Gamma \) by creating a node for each gene, and then inserting the edge \((i, j)\) if and only if the (Levenshtein) distance between gene \( i \) and gene \( j \) is smaller than or equal to \( \Gamma \), that is

\[
(i, j) \in G_\Gamma \Leftrightarrow d_L(i, j) \leq \Gamma
\]

Since the distance between \( i \) and \( j \) is the same as the one between \( j \) and \( i \), such edges are not directed and each one of them represents the similarity between the unordered pair \((i, j)\). Let \( D \) be the maximum distance detected among pairs of input data. We consider 100 different thresholds that filter the most similar pairs of genes, namely from 0.00 to 0.99 times \( D \). The former corresponds to allowing no difference and the latter corresponds to allowing the maximum distance detected between pairs of genes. Of course, the interesting cases are not these two extreme ones.

We will denote by Levenshtein graphs all graphs built in this way where the Levenshtein distance is used and a threshold is fixed. The novelty of our approach is that we operate on these graphs directly, and not on the distance. We therefore study the biological data at a graph level skipping the sequence one. In fact, in the literature, in order to study properties of biosequences, random biosequences are generated and compared to the biological data. What we try to do here is something truly different, as our randomly generated data is not at the level of sequences, but directly at the level of graphs. Since the use of random sequences is, among pre-existing methods, the most similar to our idea of comparing with random graphs, we will use this known method to test the behavior of our models. In fact, since our final aim is to detect cases where the biological data differ from
Figure 5.1: Distance distributions in *Mycoplasma genitalium*; myge is the distribution of the organism, smyge is the one obtained from the shuffling within genes, and xmyge is the one from the shuffling of the complete genome.

the random one, it would make no sense to compare a new random model directly with the biological data, as we could not know whether possible differences are due to the significance of the biological data (which is what we are looking for), or to a bad choice of the model (which is what we want to avoid), resulting in graphs that cannot be compared at all. For this reason, the first test we do of our random model is to compare it with Levenshtein graphs built from random sequences, as these are models that are known to be comparable to the biological data. More precisely, given the genome to be analysed, we generate two versions of shuffled sequences. In the first version, the sequence of each gene is randomly shuffled. In the second version, all the sequences of the genes in the genome are shuffled, and then the original number of genes (with their original sizes) is generated. In this way, we do have random data comparable to the one we are studying, and we will use this to test our new approach before possibly applying it. Histograms of figures 5.1 and 5.2 show the distribution of all distances between genes for the biological data and for the two versions of shuffled genomes. Figure 5.1 refers to *Mycoplasma genitalium*, while figure 5.2 concerns *Mycoplasma pneumoniae*

It is evident that there is not much difference between the two shuffling techniques. What is also evident, especially in *Mycoplasma pneumoniae*, is the presence

---

3 This is another bacterium belonging to the Mycoplasma family and with characteristics analogous to *Mycoplasma genitalium*. It has 676 genes.
5.4. The graph theory approach

Figure 5.2: Distance distributions in *Mycoplasma pneumoniae*; mypn is the distribution of the organism, smypn is the one obtained from the shuffling within genes, and xmypn is the one from the shuffling of the complete genome.

...of a few more small distances in the unshuffled biological data with respect to the shuffled one. As a preliminary use of the comparing-to-random approach, this shows how biologically significant things happen for small values of the threshold where the biology shows more similarities than expected. In the next chapter we will see the reasons of this.

Starting from the distance distributions showed in figures 5.1 and 5.2, we obtain, for different values of $\Gamma$, very different graphs $G_{\Gamma}$. Namely, for small values of $\Gamma$ the graphs result more sparse, while for higher $\Gamma$’s we get in general more edges connecting the $n$ nodes. In order to let the comparison with random graphs make sense, the biological data graph and the random one should have (at least approximately) the same number of edges. Therefore, for each $\Gamma$, after we generate $G_{\Gamma}$, we count the number of edges it has. This is the number of edges we require the random graph to have. In order to have an idea of how the number of edges changes in Levenshtein graphs with respect to the threshold, figure 5.3 shows the number of resulting edges in the graphs for all values of the threshold in *Mycoplasma genitalium*. 
5. Clustering families of paralogous genes by using random graphs

5.4.3 Comparing with random graphs

Before showing how our random graphs are built, we describe which are the properties of $G_T$ that we want to compare with the properties of the corresponding random graph. Returning to the original biological problem, there are several graph properties that reflect interesting characteristics of genes and gene families. An initial important one is the degree of the nodes. The degree of node $i$ of $G_T$ corresponds to the number of genes that are similar to the $i$-th gene. Concerning the degrees of nodes, we will take the following parameters into account:

**Number of nodes** Of course this value is always the same for all the $G_T$ representing data of a same genome, as it corresponds to the number of genes. Hence, this will also be the number of nodes of the random graphs we will use for the comparison. In the following we will denote this value by $n$.

**Number of edges** This is an important data of the $G_T$ which we will use for the generation of the corresponding random graph. In fact, only once $\Gamma$ is known and $G_T$ built, do we know how many edges we need for the random graph. In the sequel, we will denote this value by $m$. At this point we need to report a choice we made concerning loops of the graphs. Both in the graph representing the biological data and in the random model, we do have all edges $(i, i)$, because...
obviously every gene is similar to itself (whatever the threshold). But since we are eventually interested in the differences between the biological and the random data only, we will not consider loops in the statistics to be compared, as they will never result in interesting information. This choice will concern also the values of all the parameters that follow.

Mean node degree and its variance

Degree distribution This is actually a histogram that, for each possible value $k$ of the degree, plots how many nodes have degree $k$.

Another set of parameters concerns the lengths of paths in the graph. Given $\mathcal{G}_T$, we compute the minimal paths among all pair of nodes, i.e., the paths having minimal number of edges that go from one node to another. Since a minimal path from $i$ to $j$ is the reverse of a minimal one from $j$ to $i$, (and in particular it has the same length), in our statistics we only count once the path from one node to the other ignoring the reverse one. For the computation of all paths of the graph we used the Floyd algorithm ([AHU82]), which has time complexity $O(n^3)$ (with $n$ being the number of nodes). This is the most time consuming step for the computation of the parameters, as the degree information can be computed in linear time. The parameters we will consider are the following:

Number of minimal paths This is the total number of minimal paths that appear in the graph. It obviously corresponds to the number of all possible pairs of nodes (that is $n(n-1)/2$, as we count only one path for each pair of nodes if the minimal path is not unique) when the graph represents a single connected component, and is lower than that otherwise.

Mean path length and its variance

Path lengths distribution This is another histogram that shows, for each possible value of path length, how many pairs of nodes are connected by a minimum path of such length.

Number of connected components

These are the values we computed for the graphs $\mathcal{G}_T$ for all values of $\Gamma$. For each one of them, we then generated the random graph with (approximately) the same number $m$ of edges, and also for this latter the values of these parameters are computed and compared to those of $\mathcal{G}_T$. The two histograms of the degree distribution and paths length distribution include all the other information, and for this reason our attention focused on them. Our first attempt was to use a simple random graph whose only requirements were that it should have $n$ nodes and $m$ edges. The results of this comparison will lead us to further searches that are described in section 5.5.
5.5 Searching for a good random graph model

5.5.1 The model $\mathcal{G}(n, m)$

There is a huge literature on random graphs. The two main models in the literature (see [Bol85] for a good review) are $\mathcal{G}(n, m)$ and $\mathcal{G}(n, p)$. The first one consists in all graphs having $n$ nodes and $m$ edges, where each graph has the same probability. The second model, $\mathcal{G}(n, p)$ with $0 < p < 1$, contains all graphs having $n$ nodes and where each of the possible edges may appear independently with probability $p$. Let $N$ be the total number of possible edges that can appear in a graph having $n$ nodes. Clearly, $N = n(n - 1)/2$ in undirected graphs (not considering loops). It is quite intuitive that, as concerns most graph properties, these two models are practically interchangeable provided $m \approx Np$. We will use this opportunity when we generate our desired random graphs. In fact, once we have built $\mathcal{G}_r$, as we described in section 5.4.3, we compute the number of edges $m$, which we then impose to the random graph. Actually, a random generator is much easier to build using the $\mathcal{G}(n, p)$ models and, moreover, many interesting results about properties of the node degrees of a random graph are known for $\mathcal{G}(n, p)$ only. Therefore, once we have the number of edges of the biological graph $\mathcal{G}_r$, we actually compute $p = m/N$, and we generate the random graph using the model $\mathcal{G}(n, p)$ instead of $\mathcal{G}(n, m)$.

After the random graph is generated, we compare the degree distribution of such a simple model with that of the corresponding Levenshtein graphs. Here we used a result of the theory of random graphs: in the model $\mathcal{G}(n, p)$, the degree distribution follows a binomial law with parameters $n - 1$ and $p$ ([Bol85]). The binomial law is one of the simplest ones in probability theory, and according to it, the just mentioned result gives the following formula for the probability of a node to have degree $k$:

$$Prob(\text{degree} = k) = \binom{n - 1}{k} p^k (1 - p)^{n - 1 - k}$$

Using this formula (and [PTVF92] for its implementation), we generated the histogram of the theoretical degree distribution of a random graph having $n$ nodes and $m$ edges (here we changed from $\mathcal{G}(n, m)$ to $\mathcal{G}(n, p = m/N)$, as we do know how many edges we want the graph to contain, but the distribution function we have depends on $p$ and not on $m$). We compared the obtained distribution with that computed on the Levenshtein graphs built from the shuffled sequences. This is achieved by first computing the distribution for a shuffled version of Mycoplasma genitalium for a given threshold, and then computing the theoretical distribution of a random graph having the same number of edges, together with the corresponding binomial distribution. In figure 5.4 the curves for a threshold $\Gamma = 0.74$ are shown, while in figure 5.5 we have those for $\Gamma = 0.81$.

In figure 5.5, it is clear that the behaviour of the random graph $\mathcal{G}(n, p = m/N)$
5.5. Searching for a good random graph model

Figure 5.4: Degree distributions for a random graph of the model $G(468, m = 133)$ (curve gnm) and for a shuffled *Mycoplasma genitalium* (curve xmyge) with a threshold 0.74. Both the graphs have 133 edges. The curve bino shows the binomial distribution of parameters $n - 1 = 467$ and $p = 133/N$. The histograms represent, for all $k$, the number of nodes of degree $k$.

is too different from that of the corresponding Levenshtein graph (that is, having the same number of edges) built from the shuffled sequences. Indeed, for the highest value of the threshold (see figure 5.5) it is clear that the degree distribution of the Levenshtein graphs does not even look like a binomial one. Increasing the threshold (and then the number of edges), the curve of the binomial distribution (as well as that of the random graph) simply shifts to the right, showing higher degrees for the nodes. The curve for the shuffled data, instead, presents lower degrees and then the curve assumes a different shape, showing maximum values for higher degrees. These results make the simple model $G(n, m)$ of random graphs not suitable for our aim, as it is so different from the shuffled biological sequence that it cannot be used for comparison. It seems that in the shuffled data, once the threshold is increased, there remain nodes with low degrees that do not appear anymore in the random graph. The reason for this resides in the nature of the Levenshtein graphs: in fact, if our random model has a fixed number of edges, they are spread evenly in the graph because the probability $p$ of any of the edges is assumed to be independent from the presence of any other edge. This does not hold in Levenshtein graphs. In some sense we can say that Levenshtein graphs are not as completely random as the basic $G(n, m)$ model. In fact, assuming that two sequences $i$ and $j$ are similar (i.e., the
edge \((i, j)\) appears in the graph), and that \(j\) is also similar to \(k\) (i.e., the edge \((j, k)\) is also there), then a similarity between \(i\) and \(k\) has a high probability to appear, at least higher than the average (that would correspond to the presence of the edge \((i, k)\) in the graph), while the random graph gives this event the same probability as any other edge. This property appears in the graph built from the shuffled sequences, and appears even more does in the \(G_r\) built from the pure (not shuffled) biological data, since we know that duplications did take place. The model of random graphs must therefore be modified in order to resemble Levenshtein graphs more closely so as to be comparable to them. What is the nature of the modification we need? The graphs derived from biological sequences have the property that some edges are more probable than others. This corresponds to a kind of (partial) transitive closure of the similarity relation: \((i, j)\) and \((j, k)\) imply a high expectation for \((i, k)\) to be present as well. This is the reason for our claim that Levenshtein graphs differ from random graphs in the sense that they present a certain degree of partial transitive closure for edges. We explicitly call it a partial closure because a total closure would affect the data too much. We will see in section 5.5.3 that a certain degree of closure is necessary to approximate similarities in the biological data.
5.5.2 Ultrametrics and molecular evolution

In order to address transitive closure properties of Levenshtein graphs, it is worthwhile to discuss some theories of molecular evolution and corresponding analyses of the Levenshtein distance. For this purpose we remind the reader that a distance has the following properties:

- **Nonnegative property**: \( d(i, j) \geq 0 \) for all \( i \) and \( j \),
- **Zero property**: \( d(i, j) = 0 \) if and only if \( i = j \),
- **Symmetry property**: \( d(i, j) = d(j, i) \) for all \( i \) and \( j \),
- **Triangle inequality**: \( d(i, k) \leq d(i, j) + d(j, k) \) for all \( i, j \) and \( k \),

where \( d(i, j) \) indicates the distance between \( i \) and \( j \). The relation we are interested in (and that corresponds to the existence of edges in Levenshtein graphs), is the following:

\[
iR j \text{ if and only if } d_L(i, j) \leq \Gamma
\]

where \( \Gamma \) is the threshold and \( d_L() \) denotes the Levenshtein distance. The metric properties we just listed do not imply any form of transitive closure of the relation, but another class of metrics with a stronger version of the triangle inequality property does: a distance is said to be an **ultrametric** if and only if it has the following properties:

- **Nonnegative property**: \( d(i, j) \geq 0 \) for all \( i \) and \( j \),
- **Zero property**: \( d(i, j) = 0 \iff i = j \).
- **Symmetry property**: \( d(i, j) = d(j, i) \) for all \( i \) and \( j \).
- **Ultra triangle inequality**: \( d(i, j) \leq \max\{d(i, k), d(j, k)\} \) for all \( i, j \) and \( k \).

It is easy to see that ultrametrics imply that the relation \( R \) is transitively closed. In fact, we have

\[
iR j \iff d(i, j) \leq \Gamma \text{ and } jRk \iff d(j, k) \leq \Gamma.
\]

Since \( d(i, k) \leq \max\{d(i, h), d(k, h)\} \) for all \( h \), in particular we have for \( h = j \) that

\[
d(i, k) \leq \max\{d(i, j), d(k, j)\} \leq \Gamma,
\]

thus implying \( iRk \).

We have thus proved that \( iRj \) and \( jRk \) implies \( iRk \), i.e., the definition of transitive closure.

Our aim is now to show that the Levenshtein distance partially behaves as an ultrametric. As explained in section 2, genes duplicate during evolution and the
copies then mutate successively in different ways so that new proteins become available to the organism. If mutations always took place in different amino acids, then the number of mutations could easily be counted; it would be the distance in relation to the ancestor sequence. But since mutations can happen more than once at the same position, a more recent mutation could overwrite an older one, thus deleting its trace. In the molecular evolution literature, this effect is called saturation. If we go back to the Levenshtein distance among genes of a same organism, we observe that, when this linearity property holds, the distance is an ultrametric (see below for a brief explanation of this) and then our relation is transitive. But since this linearity property holds only up to a certain point, the result is something we assume to be a partial transitive closure only. This is the biological explanation of the model we supposed to be a better candidate for the comparison with Levenshtein graphs, and that we will detail in the next sections. We just state that when the number of differences among genes is proportional to the number of mutations (which itself is proportional to time), then the triangle property holds for the Levenshtein distance. In short, the reason is the following: if we imagine the classical representation of evolution as a tree where each node is the ancestor gene of its sons, then the distance of two nodes appearing at the same level (an organism possesses at any time only genes that appear at the same level of such a tree) is equal to the sum of their distances to the lowest common ancestor, which is also the maximum distance that any other node at the same level may have to any of the two of them.

5.5.3 The model $G(n, m, f)$

Since the partial transitive closure of Levenshtein graphs is assumed, we elaborated a new model of random graphs that reflects this property. We say that a transitive closure of degree $f$ of a graph $G$ is the graph $G^f$ whose set of nodes is the same as that of $G$, and the set of edges is the one of $G$ plus all edges linking nodes that in $G$ are connected by a path of length at most $f$. For example, for $f=2$, we add to $G$ all edges $(i, k)$ where in $G$ we had the edges $(i, j)$ and $(j, k)$, but not $(i, k)$. Hence, our new model is indeed $G(n, m, f)$; a graph belonging to this set is a random graph having $n$ nodes, $m$ edges, and having been transitively closed with degree $f$. We stress that $m$ is the number of edges obtained after the partial transitive closure applied to a normal random graph having a suitable number $m_0$ of edges. For this purpose, we have to generate graphs in $\hat{G}(n, m, f)$ in the following way. Once $n$, $m$ and $f$ are known, we have to recover the value of $m_0$ such that the $f-$closure of $G(n, m_0)$ has $m$ edges. In order to avoid searching it by means of an expensive dichotomy (the closure algorithm has a time cost of $O(fn^2)$), we used the following result which holds for graphs in $G(n, p)$:

$$P_{n, p}(k) \approx 1 - (1 - p^k) \left(1 - p^{(k+1)(k-2)/2}(k! \cdots 2! \cdots 1! \right)^{(k-1)}),$$

where $P_{n, p}(k)$ is the probability that any two distinct nodes are connected by at
least one path of length \( k \). From this formula, which holds for \( k \geq 2 \) (for \( k = 1 \) we have trivially \( P_{n,p}(1) = p \)), we obtain the following:

\[
P_{n,p}^{\text{MIN}}(k) \approx P_{n,p}(k) \prod_{i=1}^{k-1} (1 - P_{n,p}(i)),
\]

where \( P_{n,p}^{\text{MIN}}(k) \) denotes the probability that the minimal path is of length \( k \). It is clear that, for \( p_0 = m_0/N \) and \( p = m/N \) (where \( m \) is the number of edges after the \( f \)-closure and \( m_0 \) the one before), we have:

\[
p \approx \sum_{h=1}^{f} P_{n,p_0}^{\text{MIN}}(h).
\]

Using this property, our algorithm can compute the requested \( m_0 \) in just one step and then the suitable graph of \( G(n, m, f) \) can be obtained by the \( f \)-closure of one in \( G(n, m, m_0) \). The result is that what we compare with the Levenshtein graph is now a random graph having the same number of nodes and edges, but moreover partially transitively closed. As we will see, the results are better than the basic model \( G(n, m) \), for degree as well as for path statistics. Regarding the latter, we have to say that not every parameter among those we listed in section 5.4.3 is equally significant. The number of minimal paths, as an example, does not change in a graph once we add edges among nodes that were already connected (by a path of \( f \) or less than \( f \) edges), while this is what we do when performing the partial closure. The same holds for the number of connected components and the reasons are analogous. Consequently, in a graph \( G \) of the set \( G(n, m, f) \), these values are the same as those of a random graph of \( G(n, m_0) \) whose \( f \)-closure results in \( G \).

All the other parameters in the path statistics, mostly regarding the lengths of the paths are, however, definitely significant. Since graphs in \( G(n, m, f) \) are randomly generated, in order to guarantee the significance of this random data, we generated 100 graphs for each case. The statistics we report are then simply the average of all those obtained with these generations (and in particular the average number of edges will be very close to the requested one). Moreover, for these 100 representatives of \( G(n, m, f) \), we compute the variance so that we will understand what we can expect to observe when working with generations of random graphs. The result is that for the degree distribution, the variance appears to be very low, and therefore even less than 100 random generations would have been enough to give significant results. On the other hand, the variance for path statistics is not trascurable.

Figures 5.6, 5.7 and 5.8 represent histograms of the degree distributions for three representative values of the thresholds, that are 0.75, 0.81, and 0.86. They result respectively in 1287, 77323, and 101431 edges for the strongly shuffled version (xmyge) of the genome of \textit{Mycoplasma genitalium}. The requested number of edges for the graphs in \( G(n, m, f) \) are these, but an approximation of 5% will be allowed. All the
Figure 5.6: Degree distributions. The curve xmyge refers to the graph built for the shuffled sequences (1287 edges), while f2 and f3 are the distributions of the corresponding graphs in $\mathcal{G}(n, m, 2)$ and in $\mathcal{G}(n, m, 3)$, respectively.

The figures show the distributions for the shuffled genome and the corresponding graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$ having the same number of edges. Figure 5.6 refers to graphs having all 1287 edges, while in figure 5.7 graphs have 77323 edges, and finally graphs whose degree distributions are shown in figure 5.8 have 101431 edges (fully saturated graphs will have $468 \times 467/2 = 109278$ edges). It is immediately evident that the curves related to the graphs in $\mathcal{G}(n, m, 3)$ are further away from the biology than those of $\mathcal{G}(n, m, 2)$. This is due to a phenomenon that also causes an undesired behaviour of $\mathcal{G}(n, m, 2)$. The problem is that, in order to obtain a number $m$ of edges, the closure is performed on a random graph in $\mathcal{G}(n, m_0)$ with $m_0$ considerably smaller than $m$, especially for a high $m$ and above all especially for a closure of degree 3. With such a low value of $m_0$, it is easy for some nodes not to be connected with any other one and the transitive closure cannot change this. In this section, data about the model $\mathcal{G}(n, m, 3)$ is still reported, so that we will also see its behaviour concerning paths, but for the aforementioned reasons, we will later abandon the closure of degree 3.

Figures 5.9, 5.10 and 5.11 represent histograms of the path distributions for the same three different thresholds. In each one, we give the distribution for the shuffled genome and those of the corresponding graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$.

One can easily see that the partially closed random graphs behave better than the
Figure 5.7: Degree distributions. The curve xmyge refers to the graph built for the shuffled sequences (77323 edges), while f2 and f3 are the distributions of graphs in $\mathcal{G}(n,m,2)$ and in $\mathcal{G}(n,m,3)$.

simple random graphs, but despite encouraging results concerning path statistics for the two highest values of the threshold, they in general differ still too much from Levenshtein graphs. The reason, as we already mentioned, is that in order to obtain the desired number of edges after the closure, we work with graphs that are disconnected before, and thus also after, the closure. This especially affects the data related to $\mathcal{G}(n,m,3)$, but also that of $\mathcal{G}(n,m,2)$. We then tried to correct this by conceiving a new model, which is actually a soft version of $\mathcal{G}(n,m,2)$. It takes as an input, instead of $f$, a parameter $\alpha$, ($0 \leq \alpha \leq 1$), which indicates the probability with which the transitive closure takes place among nodes. Details and results appear in the next section.

5.5.4 The model $\mathcal{G}(n,m,\alpha + 1)$

This new model builds graphs having $n$ nodes, $m$ edges, and transitively closed with a degree $1 \leq \alpha + 1 \leq 2$. In fact, these graphs are built similarly to those in $\mathcal{G}(n,m,2)$ except that in the closure, once we have the edges $(i,j)$ and $(j,k)$, we add the edge $(i,k)$ only with a probability $\alpha$, unless $i$ and $k$ were already connected in $\mathcal{G}(n,m)$. It is clear that for $\alpha = 0$ we obtain simple graphs in $\mathcal{G}(n,m)$ (no closure at all has taken place), while for $\alpha = 1$ we are back to the model $\mathcal{G}(n,m,2)$.

With such a modification, we can handle a degree of closure lower than 2 and thus
Figure 5.8: Degree distributions. The curve xmyge refers to the graph built for the shuffled sequences (101431 edges), while f2 and f3 are the distributions of graphs in $\mathcal{G}(n, m, 2)$ and $\mathcal{G}(n, m, 3)$.

operate this closure starting from a graph with a higher number of edges than in the case of $\mathcal{G}(n, m, f)$, partly avoiding the problem of disconnection of nodes. In figures 5.12, 5.13, and 5.14 we show the distributions (both for degrees and paths) for such graphs where $\alpha = 0.5$, together with (for a comparison) the already known curves obtained with $\mathcal{G}(n, m, 2)$ and the strongly shuffled genome. Again, we report histograms for 1287 edges (figure 5.12) in the shuffled genome and corresponding random graphs, 77323 edges (figure 5.13) and finally for 101431 edges (figure 5.14).

It is evident that the model $\mathcal{G}(n, m, 1 + 0.5)$ is better than $\mathcal{G}(n, m, 2)$ and that it goes in the good direction. Nevertheless, the graph built from the shuffled genomes looks still quite better than our models of random graphs. Therefore, the problem of finding a really suitable model of random graph is still open. Some ideas in this direction are mentioned in section 5.7. Hence, in next section we will compare the graphs built from the two genomes of *Mycoplasma genitalium* for all the thresholds with the corresponding Levenshtein graphs built from the shuffled sequences and, at the same time, just to test the random models, we will compute also the differences among the biological and the random data of the different models we elaborated so far.
5.6 Applications to a bacterium

5.6.1 Description of the organism

We give here a brief description of the bacterial genome on which we test our approach. More information than the content of this section can be found in [BLW97].

Mycoplasmae are members of a genus belonging to the class Mollicutes (soft skin), due to their complete lack of cell wall components. They have the smallest and simplest genome of all free living species. At least 24 separate species have been characterised so far. These bacteria exist in nature in relationship with some specific higher eukaryotic plants or animals. Mycoplasma genitalium and Mycoplasma pneumoniae are the only two associated with the human species. Since they are pathogenic to humans, they are the most studied among mycoplasmae. It is for this reason that the complete genome is already available for both of them.

5.6.2 Comparing histograms

If we had had a theoretical model (that is one with statistical results about mean and variance that could avoid the random generation of data) to be compared with the graphs representing the genomes, we would have easily detected values of the
threshold for which biology sensibly differs from the expected values with respect to the variance, and consequent detailed analyses for such values. For example, supposing that we knew that the degree of a node has a mean value \( <a> \) and a variance \( \sigma^2 \), then we could compute, for nodes in \( G_T \) whose degree is \( a \), the value

\[
z = \frac{a - <a>}{\sigma}
\]

that gives a measure of how much the observed value differs from the expected one with respect to the variance. When we detect nodes for which \( z \) is larger than (say) 3, we may then claim that the gene represented by that node has unexpected high amount of similarities with other genes, and then a more detailed biological analysis could investigate in more depth the reasons for this. Unfortunately, we do not have here such a theoretical model except for the known binomial distribution of the degree in the \( G(n, m) \) model. Therefore, we have to compute means and variances after multiple generations of random graphs. Concerning the histograms, in the previous sections, for a preliminary test of the models, we just compared histograms by plotting the curves and looking at the shape and values. In particular, we will compute a numerical difference between each pair of histograms we will compare. Let us suppose that we want to compare the histogram \( B \) representing biological data with the random one \( R \), where both have values for all \( i = 1..n \). We will
Figure 5.11: Path distributions. The curve xmyge refers to the graph built for the shuffled sequences (101431 edges), while f2 and f3 are the distributions of the graphs in \( \mathcal{G}(n, m, 2) \) and \( \mathcal{G}(n, m, 3) \).

measure the difference between them by using the \( \chi^2 \):

\[
\chi^2 = \sum_{i=1}^{n} \frac{(B[v] - R[v])^2}{R[v]}
\]

The histograms of degree distributions are computed for the biological data for each threshold and then compared, for all of them, to the corresponding random graphs. In fact, using the previous formula, we will compute the difference with (graphs built from) the shuffled sequences, the graph in \( \mathcal{G}(n, m) \), the one in \( \mathcal{G}(n, m, 2) \), and finally the one in \( \mathcal{G}(n, m, 1+0.5) \). It should be pointed out that when comparing the graph representing the genome (\( \mathcal{G}_T \)) with that built from the shuffled sequences for each threshold of the graph \( \mathcal{G}_T \), we do not use the same threshold \( \Gamma \) for both graphs. We rather search for the corresponding threshold of the shuffled version such that the two graphs have the same number of edges. Most of the time the threshold happens to be the same, but sometimes there appear slight differences. With the same constraints, (same number of edges of \( \mathcal{G}_T \) for each value of \( \Gamma \)), we generate the three different kinds of random graphs.

Figure 5.15 shows the \( \chi^2 \) for all the thresholds (only relevant data, that is from \( \Gamma = 0.7 \) to \( \Gamma = 1.0 \), is shown). As we expected, the lowest values appear for the shuffled genome, as it is still the most comparable to Mycoplasma genitalium.
Figure 5.12: Degree and (length of) path distributions (respectively at the top and at the bottom). The curve xmyge refers to the graph built for the shuffled sequences (1287 edges), while $f_2$ and $f_{1.5}$ are the distributions of, respectively, the corresponding graphs in $G(n, m, 2)$ and $G(n, m, 1 + 0.5)$. 
Figure 5.13: Degree and (length of) path distributions (respectively at the top and at the bottom). The curve xmyge refers to the graph built for the shuffled sequences (77323 edges), while f2 and f1.5 are the distributions of, respectively, the corresponding graphs in $G(n, m, 2)$ and $G(n, m, 1 + 0.5)$. 
Figure 5.14: Degree and (length of) path distributions (respectively at the top and at the bottom). The curve xmyge refers to the graph built for the shuffled sequences (101431 edges), while f2 and f1.5 are the distributions of, respectively, the corresponding graphs in $\mathcal{G}(n,m,2)$ and $\mathcal{G}(n,m,1+0.5)$. 

Figure 5.15: $\chi^2$ of degree histograms for all the thresholds (differences from degree distribution of the Levenshtein graph built from *Mycoplasma genitalium*). The curve xmyge refers to the shuffled genome. f1 is the one corresponding to random graphs of the basic model $\mathcal{G}(468, m)$, f2 corresponds to graphs in $\mathcal{G}(468, m, 2)$, and finally f1.5 corresponds to graphs of $\mathcal{G}(468, m, 1 + 0.5)$.

...it, comes the curve of $\mathcal{G}(468, m, 1 + 0.5)$, better than the 2 – *closed* one, while the basic random graph is confirmed as a bad model. In particular, one can see that for the lowest thresholds (among those shown in figure 5.15), our theoretical model $\mathcal{G}(468, m, 1 + 0.5)$ does not behave much worse than the shuffled genome. We then look in a more detailed way the data detected by $\mathcal{G}(468, m, 1 + 0.5)$ for low thresholds. We do not claim here that this is the perfect model for the comparison and that the following results are definitely biologically significant. We just want to investigate the kind of *unusual* properties that such model is able to detect in the graph $\mathcal{G}_T$.

### 5.6.3 Back to biology

We focused our attention on values of the threshold up to 0.73, that are those for which the model $\mathcal{G}(468, m, 1 + 0.5)$ still remains close to the shuffled data. In particular, for such values of the threshold, we looked at which genes happen to be similar to others and may then form a group. In *Mycoplasma genitalium*, even with a threshold $\Gamma = 0.0$ we have an edge in $\mathcal{G}_T$ as there are two perfectly identical genes. Raising the threshold, new edges appear resulting in new groups of similar
genes. For values of the threshold up to 0.67, the groups that are detected contain mostly genes that are so similar to each other that classical alignment techniques working pairwise on gene sequences could easily detect them. We list below all groups appearing for these lower thresholds:

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Gene 1</th>
<th>Gene 2</th>
<th>Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>Unknown_261</td>
<td>polI_262</td>
<td>bb</td>
</tr>
<tr>
<td>0.30</td>
<td>Unknown_413</td>
<td>Unknown_414</td>
<td>b</td>
</tr>
<tr>
<td>0.55</td>
<td>Unknown_96</td>
<td>Unknown_288</td>
<td>b</td>
</tr>
<tr>
<td>0.55</td>
<td>gyrB_3</td>
<td>parE_202</td>
<td>bb</td>
</tr>
<tr>
<td>0.57</td>
<td>Unknown_185</td>
<td>Unknown_259</td>
<td>bb</td>
</tr>
<tr>
<td>0.61</td>
<td>Unknown_68</td>
<td>Unknown_395</td>
<td>b</td>
</tr>
<tr>
<td>0.63</td>
<td>SPase_67</td>
<td>Unknown_68</td>
<td>b</td>
</tr>
<tr>
<td>0.63</td>
<td>SPase_67</td>
<td>Unknown_68</td>
<td>bb</td>
</tr>
<tr>
<td>0.64</td>
<td>glpQ_293</td>
<td>Unknown_385</td>
<td>b</td>
</tr>
<tr>
<td>0.64</td>
<td>hlyB_179</td>
<td>glnQ_180</td>
<td>b</td>
</tr>
<tr>
<td>0.65</td>
<td>Unknown_438</td>
<td>Unknown_439</td>
<td>b</td>
</tr>
<tr>
<td>0.66</td>
<td>Unknown_25</td>
<td>rfbV_60</td>
<td>bb</td>
</tr>
<tr>
<td>0.67</td>
<td>Unknown_224</td>
<td>aroP_225</td>
<td>bb</td>
</tr>
<tr>
<td>0.67</td>
<td>Unknown_431</td>
<td>Unknown_442</td>
<td>bb</td>
</tr>
<tr>
<td>0.67</td>
<td>gyrA_4</td>
<td>parC_203</td>
<td>bb</td>
</tr>
<tr>
<td>0.67</td>
<td>hlyB_179</td>
<td>cbiO_304</td>
<td>b</td>
</tr>
<tr>
<td>0.67</td>
<td>hlyB_179</td>
<td>glnQ_180</td>
<td>cbiO_304</td>
</tr>
</tbody>
</table>

Each line starts with the threshold and then lists the group of genes that resulted similar. For each Γ, we listed only new groups, avoiding to list, for example, the pair of two identical genes Unknown_261 and polI_262 that appear already for Γ = 0.0. The sign 'bb' denotes that a running of BLAST\(^4\) detected exactly the same group of genes as being similar to one another by means of pairwise sequence comparisons. The sign 'b' indicates that, next to the genes listed, BLAST also detected similarities with other sequences (with a p-value < 10\(^{-3}\)). The reason for this is that BLAST does not use the Levenshtein distance and thus catches similarities among genes even when they have much different lengths. These are missed by our method in its current state. The range of Γ where our \(\mathcal{G}(468, m, 1+0.5)\) detects something and where the comparison with Mycoplasma genitalium seems to be still feasible, is the one that comes right after the above data and up to Γ = 0.72. Here is the complete list of our results:

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Gene 1</th>
<th>Gene 2</th>
<th>Alphabet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.68</td>
<td>rpL35_196</td>
<td>rpL34_465</td>
<td>n</td>
</tr>
<tr>
<td>0.68</td>
<td>rpL36_174</td>
<td>rpL35_196</td>
<td>n</td>
</tr>
<tr>
<td>0.68</td>
<td>rpL36_174</td>
<td>rpL35_196</td>
<td>rpL34_465</td>
</tr>
<tr>
<td>0.69</td>
<td>hlyB_179</td>
<td>glnQ_180</td>
<td>glnQ_303</td>
</tr>
</tbody>
</table>

\(^4\)BLAST is probably the fastest database scanning technique available at the current time.
<table>
<thead>
<tr>
<th>Similarity</th>
<th>Gene 1</th>
<th>Gene 2</th>
<th>Gene 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.70</td>
<td>rpL24_162</td>
<td>Unknown_393</td>
<td>Unknown_448</td>
</tr>
<tr>
<td></td>
<td>rpL36_174</td>
<td>rpL35_196</td>
<td>BS18_423</td>
</tr>
<tr>
<td></td>
<td>rpL34_465</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.71</td>
<td>Unknown_267</td>
<td>Unknown_319</td>
<td>P69_406</td>
</tr>
<tr>
<td></td>
<td>glnQ_180</td>
<td>Unknown_290</td>
<td>glnQ_303</td>
</tr>
<tr>
<td>0.72</td>
<td>Unknown_198</td>
<td>Unknown_242</td>
<td>nodF_287</td>
</tr>
<tr>
<td></td>
<td>Unknown_337</td>
<td>Unknown_389</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unknown_55</td>
<td>ptc1_108</td>
<td>Unknown_221</td>
</tr>
<tr>
<td></td>
<td>Unknown_239</td>
<td>pip_310</td>
<td>gidB_380</td>
</tr>
<tr>
<td></td>
<td>Unknown_415</td>
<td>Unknown_418</td>
<td></td>
</tr>
<tr>
<td>0.72</td>
<td>amiE_79</td>
<td>hlyB_179</td>
<td>glnQ_180</td>
</tr>
<tr>
<td></td>
<td>cbiO_304</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.72</td>
<td>dnaE_10</td>
<td>Unknown_57</td>
<td>nodF_287</td>
</tr>
<tr>
<td></td>
<td>Unknown_337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.72</td>
<td>rpS19_155</td>
<td>rpL36_174</td>
<td>rpL33_325</td>
</tr>
<tr>
<td></td>
<td>BS18_423</td>
<td>rpL28_425</td>
<td></td>
</tr>
</tbody>
</table>

As we can see, only a few of them are detected by BLAST (the sign 'n' means that BLAST did not detect any significant similarity within the group). Most of them are only suggested by $\mathcal{G}(468, m, 1+0.5)$. The first three groups, appearing for $\Gamma = 0.68$, correspond indeed to families of ribosomal proteins, as well as the last group of 5 genes at $\Gamma = 0.72$. We had a more detailed look at the second group appearing at $\Gamma = 0.72$ that consists of 8 genes. We ran BLAST for the gene gidB_380 (in the graph $\mathcal{G}_\Gamma$, it is this gene whose node has seven edges linking it to all the group), and we observed that none of the similarities detected here are found by BLAST. What it did detect are similarities (even strong ones) with genes of very different size, for the reasons we already explained. Thus, since we had a similarity detected by $\mathcal{G}(468, m, 1+0.5)$ and not by BLAST, we went to investigate one of them at the level of the sequences. What follows is the alignment of the sequences gidB_380 and Unknown_418 where the '*' denotes a perfect matching, the symbol ':' indicates that the two amino acids are very similar, and a less strong similitude is denoted by ':'

```
Unknown_418
gidB_380
```

```
M I E T L N F E K Q H Y A
```

```
Unknown_418
gidB_380
```

```
F L I K A I E E N T N F G
M N N A N F
```

```
*: * : *
```

```
Unknown_418
gidB_380
```

```
L S Q L T L I D R L K A I
E K Y V D L
```

```
*: :
```
| Unknown_418 | V I S Y N E F F N Q K P L  |
| Unknown_418 | V F E A N K N F N  |
| Unknown_418 | * : - * : * :  |
| gdbB_380    | T I S S P S N E K S L H L  |

| Unknown_418 | E T E Y L E K K K I K K S  |
| Unknown_418 | L T G F K T K E A I Y Q N  |
| Unknown_418 | * : * : * : -  |
| gdbB_380    | N H K Q D Q K H F S L F E  |
| Unknown_418 | L V I E I L T L F K G Y E  |
| Unknown_418 | : - * - :  |
| gdbB_380    | K S F I D K S E K T P K N  |
| Unknown_418 | K F F I D K T V A D L G S  |
| Unknown_418 | * * * * : -  |
| gdbB_380    | D E V T N N K F L D T S K  |
| Unknown_418 | G N G S P G I I L K L L F  |
| Unknown_418 | - : - : * : : : :  *
| gdbB_380    | L N L A N I A L A I N A F  |
| Unknown_418 | Q K I K K L V L I D S K H  |
| Unknown_418 | : : : : - * - -  |
| gdbB_380    | N D N K W I N H F Q N L L  |
| Unknown_418 | K K I S F L N K L T K Q L  |
| Unknown_418 | S V F Q T K F N D K D K Q  |
| gdbB_380    | N L E K T V A I C E R  |
| Unknown_418 | : : : * : :  |
| gdbB_380    | N N L S Y F N N F I D K Y  |
| Unknown_418 | I E V H K N H Y D V I  |
| gdbB_380    | : - - : * - *  |
| Unknown_418 | S A R D I V K A T K I V K  |
| gdbB_380    | C S R G L S T I I K  |
| Unknown_418 | A S S F G I V I L F E D Q  |
| gdbB_380    | V N D L A F S L L N S  |
| Unknown_418 | K I A M R L W K E A I E E  |
| gdbB_380    | K  |
| Unknown_418 | G N V Q A T I F Q I F N Q  |
| gdbB_380    | G I I F H I K Q S  |
| Unknown_418 | * * * : * : -  |
| gdbB_380    | N L F L A S F S E H Q Y K  |
| Unknown_418 | D  |
5.7 Further work

If and when a safer random graph model is found, then it would be interesting to apply our method to other available complete genomes such as that of Bacillus subtilis ([K+97]) as well as to the other Mycoplasma whose genome has already been sequenced (that is Mycoplasma pneumoniae). Also, as we have seen in the tests, we can reasonably hope, for the model $G(n, m, \alpha + 1)$, to obtain better results when comparing data concerning paths in the graphs. Hence, such an application is another possible further work.

Nevertheless, our main goal is to improve the random graph model, that is to find a better one. It seems that a somehow high local connectivity is still missing in
the model $\mathcal{G}(n, m, \alpha+1)$. To this purpose, some recent work caught our attention [Kle00, KKR+99, Kle99]. A few studies performed mainly by J. Kleinberg at Cornell University attempt to characterize the graph that represents the web. Nodes are sites and edges are links. It has been observed that lengths of minimal paths connecting nodes are surprising low. It seems also [Mar01] that gene duplication in evolution could present some analogies with the way new sites are created and modified. These similarities could be worth further investigations.

Another notion of distance among biosequences could be used. For example, a more suitable one could be what is often denoted as the best fit distance that could better detect events of duplications in bacteria. In a few words, the best fit distance between two genes, having in general different sizes, detects the best alignment of the smaller sequence into the larger one. In fact, duplications in bacteria consist mostly of two events: firstly, a gene could truncate and thus a subsequence of it become a new shorter gene; secondly, two genes could concatenate and give rise to a new longer one. It is easy to see that both these events are missed by the Levenshtein distance.

Another possible improvement concerns the different rates of duplications in different families of genes. It is documented ([Coi96]) that some genes duplicate more often that others. In our model, this could be represented by a larger probability of the presence of certain edges: genes of the same and quickly duplicating family should correspond, in the random graphs, to nodes that have a larger probability of being connected by an edge.
Part II

Motif extraction
Chapter 6

Introduction to part II

6.1 Motif extraction

Patterns appearing repeated either inside a same string or over a set of strings are important objects to identify. Such repeated patterns are called *motifs* and their identification, *motif inference* or *motif extraction*. The area has many potential applications, namely to data compression ([CL96, CL98]), natural languages, music, data bases, basically, any activity or research requiring text mining. The field of applications that concerns us is molecular biology. Here the motifs may correspond to functional elements in DNA, RNA or protein molecules, or to whole genes whose sequence appear strongly similar. In biological applications, it is mandatory to allow for some mismatches between different occurrences of the same motif. In fact point mutations might have taken place, as well as errors in the sequencing procedure, so that molecules that are basically the same have no longer identical sequences. This is what makes the problem difficult from the computational complexity point of view. Known exact algorithms have a time complexity that is exponential in the number of allowed mismatches. This number cannot be considered a constant because for reasonable queries, it is actually proportional to the length of the motif. It would not make sense (nor would it be feasible) to ask for long motifs with few mismatches, as there are too many of them and the search space would explode. In this overview (which does not aim to cover the area, but rather to just introduce the specific problems that we address in the next chapters), we only consider approaches that find motifs when differences are allowed among distinct occurrences. Several different kinds of motifs have been considered in the past for many sorts of investigations on biosequences. See for example [BJEG95, SV96], [Sag00] and [VMS99, VMLS00] for reviews (the last two ones also contain applications to the detection of promoter sequences in bacterial genomes). Specific algorithms for the detection of specific kinds of repetitions such as palindromes and mirror repeats are presented [SV97], while in [SM98] an algorithm for detecting satellites is proposed. There are different kinds of repetitions that one can be interested in. In some applications, a minimum
number $q$ (often called *quorum*) of occurrences is required within an unique input sequence. In other cases, one can be interested in seeking motifs that appear in at least $q$ out of $N$ given input sequences. Moreover, one can ask for motifs that appear at least $q$ times while some queries can ask for motifs that appear *exactly* $q$ times. This version of the problem is the one stated in the seminal paper [KMR72] where for the first time the algorithmical problem of finding repetitions has been addressed.

Many of the techniques used (especially at the beginning) are common to data compression, pattern matching, and algorithms on strings in general [CH98, Ste94], but motif search has nowadays become a specific research field whose applications in molecular biology are of high impact. In fact, as we already stated in chapter 1, due to the huge amount of data entering genomic data bases, there is an urgent need for tools that can help molecular biologist to interpret this data. In this direction, the search for repeated patterns is one of the first things one can do on a sequence in order to detect some properties or some particularly significant functional site. In fact, a typical way to start analyzing new sequences is to group them into families that are assumed to be biologically related because the have similar function or structure, or because they are evolutionary related. Many of these classifications into families have been done using different techniques, and finding shared properties in terms of common motifs in the sequence can become a powerful classification tool. Here motif extraction could play a crucial role both in detecting these common patterns in known families, and in characterizing a newly available sequence. This is all again under the assumption that syntactic similarity reflects biological correlations.

6.2 Some previous work

There are several approaches in the literature that address the high computational complexity problem simply by not attempting to search all possibly existing motifs, but rather avoid searching for unlikely existing ones. This pruning is done in a probabilistic way that makes use of assumptions that are mostly derived from biology or statistics. Examples of such approaches can be found in [BE94] where the MEME software is introduced, and in [Leu91] which is the motif search algorithm used as preprocessing in the software for computing the transformation distance (see sections 2.2 and 4.2.2). We name these approaches heuristics, and we distinguish them from those that are *exact algorithms* and that we will consider from now on.

Motifs that are interesting for applications in computer science fall into the class of patterns that can be expressed with *regular languages* (see [AHU82]). That is, all motifs or their models can be represented by a regular expression. Within these expressions, characters may represent single chemical elements, but also a group of them. For example, one can use the four nucleotides alphabet $\Sigma_{\text{DNA}} = \{ A, C, G, T \}$
when dealing with DNA sequences (or $\Sigma_{RNA} = \{A, C, G, U\}$ for RNA\(^1\)). In the case of proteic sequences, the alphabet would be that of the twenty amino acids. Both in the case of nucleotides and amino acids, extra symbols are also often used. One of them is the wildcard (denoted as ‘*’ or ‘s’, but also with an $N$), that is a symbol representing any nucleotide or amino acid. Other symbols represent subsets of the alphabet. For example, the symbol $R$ can be used to denote purine, that is either adenine or guanine, and the symbol $Y$ stands for any pyrimidine, that is either thymine or cytosine. Also in the case of the proteic sequences, the amino acids alphabet can be divided into overlapping subsets, or even partitioned in subsets according to suitable similarities between chemical properties [SS90] (such as hydrophobicity, acidity etc.). In other cases ([SV96]), the set of symbols allowed in a position is explicitly listed in the motif description.

Another aspect that in the literature distinguishes the way models are represented is in the way to allow differences between the distinct occurrences of a motif. Some allow up to a certain number of substitutions or other edit operations such as point insertions or deletions. In some cases, gaps of variable length are admitted. This has a biological use in the sense that allowing a (possibly long) gap in an occurrence could permit to detect the motif occurrence even if a segment insertion has taken place inside it. According to whether or not gaps or deletions (or insertions) are allowed, distinct occurrences of the same motif can have equal or different length. Another option is the use of wildcards in the model that represents the motifs, so that this latter can cover occurrences that differ in some points.

In the case of [MS00, MS01] the so-called structured motifs are taken into account. That is, the model seeked is composed of $p$ (in general $p > 1$) motifs that in all occurrences appear at suitably given distances. The algorithm that finds these motifs will be described in chapter 8, where a possible improvement is suggested. The algorithm presented in [MS00] uses the suffix tree data structure, that we will describe in section 6.3. Building a suffix tree of the input sequence, and spelling the motifs by traversing this tree is a technique that had already been used in [Sag98]. In [SVS95a, SVS95b] (which extend an idea introduced in [KMR72] for finding motifs with no mismatches) a tree is used for representing the search space of motifs to be spelled. The tree is actually pruned to motifs that satisfy the quorum, and when this does not hold anymore, the whole subtree is discarded (and thus actually never built).

\(^1\)Indeed, many gene sequences are actually present in data bases with their mRNA sequence, that is the one directly transcribed during the protein synthesis. This is the case of eukariotes because the mRNA certainly does not contain introns, and for some application this is the sequence actually suited because it represents coding regions only.
6.3 Preliminaries: suffix trees

A suffix tree ([McC76]) of a string $s \in \Sigma^n$, is a compact trie whose leaves correspond to the $n$ non-empty suffixes of $s$. It is built on an augmented version of the text, that is an endmarker `$\$$` (with $\$$ \not\in \Sigma$) is appended as last letter. It takes $O(n)$ space, which is optimal since it stores all the suffixes of an $n$ long string. Each arc is labeled with a substring of $s$, and sibling arcs are ordered according to the lexicographical order of the first letter of their labels. The labels of the arcs of each path from the root to a node spell a substring of $s$. The trie has a branching at node $v$ whenever the substring spelled at $v$ appears in the string at least twice and at least two of the occurrences are followed by different letters. Besides being a compact way of representing of a text, the efficiency of suffix trees is due to the fact that a suffix tree for a sequence $s$ such that $|s| = n$ can be built in $O(n)$ time [Ukk92, Ukk95, Gus97]. An example of suffix tree is shown in figure 6.1.

A suffix link is an edge on a suffix tree that connects node $v$ spelling the substring $\alpha x$ (with $\alpha \in \Sigma$ a letter and $x \in \Sigma^*$ a word) to the node that spells $x$. Suffix links can be built on the fly during the suffix tree contraction without increasing the linear time complexity.

A generalized suffix tree is a suffix tree of a set of $N > 1$ strings. To each one of the $N$ input strings is assigned its own endmarker, which will be used in the generalized suffix tree to represent in which one of the strings the occurrence is. Therefore, there will be as many leaves as there are suffixes of the $N$ strings. Assuming that each string has length $n$, then the generalized suffix tree takes $O(nN)$ space, and its construction takes $O(nN)$ time, which is again optimal as it is linear in the input size. An example of a generalized suffix tree is shown in figure 6.2.
Figure 6.2: The generalized suffix tree for the strings $s_1 = ababc$ and $s_2 = cbabb$. The two strings have endmarkers $\$1$ and $\$2$ respectively.

6.4 Overview of part II

This part of the thesis contains two chapters corresponding to two different studies on motif extraction. They are in some sense both attempts to improve the current feasibility of motif extraction. In particular, the common goal is to make motif extraction more efficient so that longer motifs can be found. On one hand, we address the problem of trying to reduce the search space of motif extraction, and in the other case we suggest a possible improvement of known algorithms that could result particularly interesting for long motifs.

Chapter 7 analyzes an attempt of defining a notion of maximality and redundancy for motifs presented in [Par98, PRF+00]. The idea is that just some motifs (that is, a polynomially bounded number of them) could be enough to build all the others (of which there are potentially an exponential number). We name such motifs *tiling motifs*. This is a very elegant and promising idea. Nevertheless, we show that the algorithm (suggested in [PRP01]) for finding these motifs is not complete nor correct, and we analyze the reasons for this. We then prove some results that could be useful to conceive a new algorithm. We then prove some properties of tiling motifs that could result useful for designing a new algorithm for finding tiling motifs. In particular, we show that the fragments of consecutive letters in tiling motifs are at most $n - 1$ (where $n$ is the length of the input sequence) and can be detected in linear time.
Chapter 8 suggests a possible improvement for the structured motif extraction algorithms of [MS00, MS01]. The idea is that if a pattern has failed to satisfy the quorum, then a suitable information can be stored for all prefixes of this pattern. For each potential prefix for which all extensions have been attempted (and have failed), it can be memorized by how many symbols at most that potential prefix can be extended. This information can be used when that potential prefix is met later as a factor of a motif which is being built: if the maximal possible extension is not enough to reach the required minimum length for the motif, then the motif is discarded (and with it the whole subtree rooted there) before reaching the actual length where the quorum is no longer satisfied. In chapter 8 we discuss how data structures and algorithms can be modified to allow this extension to the algorithm. Basically, the modification reasonably improves the performance of the algorithm but it may cost additional space. Both the entity of the improvement and the amount of needed extra space strongly depend on the instance and, above all, on the value given to parameters such as the quorum, the number of allowed mismatches, and the length of the motifs. We show some preliminary experiments on real data and some statistical observation on the data structure that motivate the belief that there are many interesting applications where the improvement is considerable and the extra space needed is negligible.
Chapter 7

Tiling motifs

Introduction

When performing motif inference for biological applications, it is necessary to allow for a suitably limited difference between distinct occurrences of the same motif. Similar but not necessarily identical subsequences must therefore be recognized as occurrences of the same motif, that somehow represents all of them. Once it is assumed that this flexibility is a necessary condition for motif inference in computational biology, we direct our attention to basic types of motifs. All occurrences of the same motif have the same length, they are not structured, and the alphabet used to describe them is just that of the string(s) from which they are inferred, plus a wild card (also called the don’t care symbol) which represents any letter of the alphabet. We assume there is only one string $s$ and the motifs to be searched are repeated words in $s$. If $\Sigma$ is the alphabet of $s$, then the motifs are elements of $(\Sigma \cup \{\cdot\})^*$ where “.” denotes the wild card and $(\Sigma \cup \{\cdot\})^*$ is the set of all finite words over the alphabet $\Sigma \cup \{\cdot\}$. As we have seen in chapter 6, motif inference is a difficult task. The difficulty is basically due to the possibly exponential size of the output. In fact, even if in general this does not hold in interesting applications, a potentially exponential number of motifs might be found.

It was shown in a recent paper [PRF+00] that the set $\mathcal{M}$ of all motifs of the above kind that appear $k \geq 2$ times\footnote{In the next chapter, we will use a different notation. Namely, the minimum number of occurrences required to the motif (quorum) will not be $k$ but rather $q$. Indeed, in chapter 8 $k$ will be the length of the motif, and in this chapter $q$ will be used for something else than the quorum. The reason of this non uniformity is due to our choice to stick to the notations used in the two different literatures.} in a string $s$ may be reconstructed from a subset of them, which represents a basis $\mathcal{B}$ from which all other motifs can be deduced. The motifs in $\mathcal{B}$ have the property of being what Parida \textit{et al.} call maximal and irredundant. Following their definition, a motif $m$ is maximal if no other motif $t$ is less specific than it (i.e. $t$ has wild cards at positions where $m$ has letters from $\Sigma$),
or $t$ is a substring of $m$, and it has exactly the same occurrences as $m$ in the input sequence. Furthermore, a motif is irredundant if it is maximal and there is no set of maximal motifs whose union of occurrences covers exactly the occurrences of $m$. The idea of a “basis of motifs” is a very elegant and powerful one, especially as it is shown in [PRF+00] that the size of $B$ is bounded above by $3n$ where $n$ is the length of $s$. That is, the potentially exponential set of all the motifs can be reconstructed from a set whose size is linear in the length of the input sequence. Furthermore, an $O(n^4 \log n)$ time algorithm\(^2\) for obtaining $B$ is sketched in the same paper. Although not stated there, the space complexity of the algorithm is $O(n^2)$.

In this chapter, we adopt the same definitions of maximality and irredundancy of motifs as in [PRF+00], correcting some minor errors, which we summarize in section 7.1. We conjecture that a tight bound $n$, instead of $3n$, holds for the size of $B$. We introduce a way to view motifs of $B$ as paths on the suffix tree of $s$, and prove some properties in this way. We then explore some inner properties of maximal and irredundant motifs, (i.e. of motifs in the basis). We also show that the problem of finding the motifs of $B$ needs further investigation because the algorithm presented in [PRP01] is not correct, nor is complete. Since maximal and irredundant motifs, (i.e. motifs belonging to $B$), enable to recover all other motifs, we call them tiling motifs, in the sense that they tile over the space of solutions.

The previous and new definitions describing tiling motifs are given in section 7.1. In section 7.2, we discuss the number of tiling motifs and we give a conjecture on the upper bound of the size of $B$. Section 7.3 presents the algorithm suggested in [PRP01] for finding tiling motifs and shows how and why it fails to detect some tiling motifs, and why it may output some motifs that are not in the basis. In section 7.4, we introduce a way of viewing tiling motifs as paths on the suffix tree of the input sequence, and we prove some properties by using this view. In section 7.5 a characterization of consecutive solid characters of tiling motifs is given using again the suffix tree. Section 7.6 introduces a new operator which is an extension of the operator $\otimes$ used in [PRP01]. This at least partially solves the problem of false positives in the algorithm mentioned above. Finally, in section 7.7, we mention future work plans.

### 7.1 Preliminary definitions

#### 7.1.1 Motifs

Let $s \in \Sigma^*$ be the string where repeated motifs are searched for. We denote by $s[i]$ the $i$-th character of $s$ and $n = |s|$ the length of $s$. We call $\Gamma$ the alphabet $\Sigma \cup \{\}$ where “:” is the wild card, and the elements of $\Sigma$ are solid characters. Following

\(^2\)In [PRF+00], an algorithm which is apparently the same was claimed to take $O(n^3 \log n)$ time.
[PRF+00], we define a partial order $\preceq$ on the elements of $\Gamma$, such that for all $\sigma \in \Gamma$ we have that $\cdot \preceq \sigma$, and for all $\sigma_1, \sigma_2 \in \Gamma$ we have that $\sigma_1 = \sigma_2$ if they are the same letter of $\Sigma$, and they are not comparable otherwise. We say that $\sigma_1 \preceq \sigma_2$ if $\sigma_1 = \sigma_2$ or $\sigma_1 \not\preceq \sigma_2$.

This enables us to define the notion of occurrence in a string for a given pattern (i.e. a string over the alphabet).

**Definition 7** A pattern $p \in \Gamma^*$ occurs at position $l$ in $s$ if $p[j] \preceq s[l + j], \forall 1 \leq j \leq |p|$.

Given a quorum $k$ which gives a lower bound on the number of required occurrences, a pattern is a *motif* if it appears at least $k$ times in the text, and it starts and ends with solid characters. Formally:

**Definition 8** Given an integer $k \geq 1$, a pattern $m \in \Gamma^*$ is a $k$-motif if $m[1], m[|m|] \in \Sigma$, and $m$ occurs at least $k$ times in $s$.

When $k$ is clear from the context, we will use the term *motif* rather than *$k$-motif*. We also make use of the notion of shifted location lists:

**Definition 9** For a motif $m$, the location list $L_m$ contains the positions of all the occurrences of $m$. $L + i = \{x + i \mid x \in L\}$ is the $i$-shift of $L$.

**Example 2** Consider the string $s = \text{abcababdcabd}$ on the alphabet $\Sigma = \{a, b, c, d\}$. If $k = 3$, we have that $m_1 = \text{abc}$ is a motif with $L_{m_1} = \{1, 6, 12\}$, $m_2 = \text{ab}$ is another one with $L_{m_2} = \{1, 4, 6, 12\}$, and $m_3 = \text{bc}$ is a third motif with $L_{m_3} = \{2, 7, 13\} = \bigcup_{i=1}^{3} L_{m_i}$. Moreover, we have that $m_4 = \text{bc} \cdot b$ is not a motif since it appears only twice in $s$, at positions 2 and 7, while $m_5 = a \cdot c$ is a motif with location list $L_{m_5} = \{1, 6, 12\} = L_{m_1}$.

Given two motifs $m_1$ and $m_2$, we write $m_1 \preceq m_2$ if $|m_1| \leq |m_2|$ and $m_1[j] \leq m_2[j]$, $\forall 1 \leq j \leq |m_1|$. Back to example 2, we have that $m_5 \preceq m_1$ and $m_2 \preceq m_1$.

### 7.1.2 Tiling motifs

We are now ready to define *maximal motifs*. Intuitively, a motif $m$ is maximal if replacing a wild card with a solid character, or extending it to the right or the left, does not result in the same set of occurrences (because if it does, then the motif with the wild card(s), or the shorter motif, brings no more information than the one obtained with $m$). Formally:

**Definition 10** Let $p_1, p_2, \ldots, p_k$ be the motifs in a string $s$. Let $p_l[j]$ be ‘.’ if $j > |p_l|$. A motif $p_i$ is maximal if and only if there exists no $p_l$ with $l \neq i$ and no integer $\delta \geq 0$ such that $L_{p_i} = L_{p_l} + \delta$ and $p_i[j] \preceq p_l[j + \delta]$ holds for $1 \leq j \leq |p_l|$.
In example 2, we have that the motif $m_1 = abc$ is maximal, while $m_2 = ab$ and $m_3 = bc$ are not, as they are just submotifs of $m_1$. Finally, $m_5 = a \cdot c$ is not maximal either, as we have that $\mathcal{L}_{m_1} = \mathcal{L}_{m_5}$ and $m_5[i] \preceq m_1[i]$, $i = 1, 2, 3$.

As shown in [PRF+00, Par98], a further requirement is needed to restrict ourselves to a polynomial number of interesting motifs:

**Definition 11** A maximal motif $m$ with location list $\mathcal{L}_m$ is **irredundant** if there are no maximal motifs $m_i \neq m$, $1 \leq i \leq p$, such that $\mathcal{L}_m = \mathcal{L}_{m_1} \cup \cdots \cup \mathcal{L}_{m_p}$. It is **redundant** otherwise.

The set $\mathcal{B}$ of irredundant motifs is sufficient to infer all interesting motifs of the input sequence by suitable combinations. We will call **tiling motifs** the motifs in $\mathcal{B}$.

### 7.2 Bound on the number of tiling motifs

The main reason for being interested in tiling motifs is the linear upper bound on their number. According to [PRF+00], for a sequence of size $n$, the cardinality of the basis $\mathcal{B}$ is at most $3n$. We believe that the following holds:

**Conjecture 2** The number of irredundant motifs is bounded above by the length $n$ of the sequence.

Our conjecture is motivated by the belief that a one to one correspondence could be conceived between irredundant motifs and the leaf of the suffix tree of the sequence (that are, indeed, as many as the length of the sequence). In fact, we believe that there are subtrees of the suffix tree where there can be more occurrences of irredundant motifs spelled than leaves, but that in this case other subtrees have no occurrences of irredundant motifs. This would be a tight bound as it is proven by any instance with $|\Sigma| = k = 1$. In this case, the input string is $s = a^n$ (if $\Sigma = \{a\}$), and all $a^i$ with $i = 1, \ldots, n$ are tiling motifs. When $k = 1$ (which is actually an uninteresting case), we always have that the whole sequence is a motif belonging to the basis because it is maximal and irredundant, independently from which are the other motifs. In fact, no motif with the same number of occurrence (that is, one) can be extended in length (for obvious reasons), nor it can replace a wildcard with a letter because there are no wild cards. In fact, no other motif which occurs only once is a tiling motif, as it would never be maximal with respect to $s \in \mathcal{B}$. Therefore, all other motifs occur at least twice in $s$, and hence they would be a tiling motif also with $k = 2$. In other words, raising the quorum from $k = 1$ to $k = 2$ only decreases by 1 the size of $\mathcal{B}$. Since the case $k = 2$ is already an interesting one, and since as we will see most results are actually independent from $k$, from now on we will assume that $k \geq 2$. 

7.3 Finding tiling motifs

In [PRF+00], a first algorithm for finding tiling motifs has been suggested. Later, in [PRP01], the same idea has been extended to the discovery of what are called flexible motifs, which are motifs where the number of wild cards can vary from one occurrence to another. Consequently, what here we refer to here as tiling motifs, in [PRP01] are called rigid motifs. Although such flexible motifs are even more interesting than rigid ones for biological applications, we refer to rigid ones because, as we will see, we claim that their detection is already a problem deserving further work. The algorithm presented in [PRP01] takes $O(n^4 \log n)$ time to detect rigid motifs (which we will keep on calling tiling motifs from now on). It consists of two phases. First, an initialization phase which is executed only once and that generates suitable motifs whose only solid characters are the first and the last one. Second, a concatenation phase which is iteratively repeated until no new motifs are generated nor any is eliminated. In both cases some suitable new motifs are generated and others are consequently eliminated due to redundancy or non maximality. The concatenation phase is expected to be repeated $\log n$ times, and each iteration takes $O(n^4)$ time. We specify the algorithm below.

begin
1. /* Pattern initialization phase: */
1.1 for each $\sigma \in \Sigma$ do
   build $m = \sigma$ with $L_m = \{ i \mid s[i] = \sigma \}$
1.2 for $d = 0$ to $n - 2$ do
   for all $m_a$ and $m_b$ resulting from the previous step do
   build motif
   \[ m = m_a \cdots m_b \]
1.3a remove suffixes
1.3b remove redundancies
2. /* Pattern concatenation phase: */
repeat
2.1 for all $m_a, m_b$ resulting from phase 1 such that $m_a[\ell] = m_b[1]$ (with $\ell = |m_a|$) do
   build the motif $m$ such that
   \[ m[i] = \begin{cases} 
   m_a[i] & i \leq \ell \\
   m_b[i - \ell + 1] & i > \ell 
   \end{cases} \]
2.2a remove suffixes
2.2b remove redundancies
until done
end
According to [PRP01], steps 1.3a and 2.2a can be realized by offsetting every location list to zero and checking for identity of location lists. Steps 1.3b and 2.2b require solving what they call the Set Union Problem. For every location list $L_m$, first all lists $L_{m_i}$ such that $L_{m_i} \subset L_m$ are detected, and then it is checked whether $L_m = \cup_i L_{m_i}$. If so, then $m$ is removed and all $m_i$s are suitably updated. The updating is made by replacing $m_i$ with $m_i \otimes m$, and extending $m_i$ until the last letter of $m$ if $|m| > |m_i|$, where the $\otimes$ operator is defined as follows. The motif $m = m_1 \otimes m_2$ is such that $m_1, m_2 \leq m$ and there is no other motif $m' \leq m$. As an example, we have that $a \cdot a \cdots d \otimes ab \cdot c = abac \cdot d$.

A few imprecisions of the algorithm as is presented in [PRP01] can be solved in the following way.

- The quorum $k$ is never mentioned. This can be solved by checking for quorum directly when new motifs are generated, that is, at steps 1.1, 1.2, and 2.1. Any newly generated motif that does not satisfy the quorum is discarded immediately. In fact, it could never be a motif, nor could be useful in the concatenation phase for building longer motifs.

- In order to remove motifs that are suffixes of others, offsetting location lists to zero and checking for equality among these new lists could result in excessive pruning, because motifs that are not suffixes would also be eliminated. As an example, let us consider the sequence $s = abacdc$ and the motifs $a$ and $c$, whose offset lists $L'_a$ and $L'_c$ are identical and both equal to $\{0, 2\}$, while the motifs $a$ and $c$ do not even overlap. After equality of the offset location lists is detected, the condition of being a suffix has therefore to be further checked before erasing. The list offsetting is thus just a shortcut with respect to a brute force approach that would search for motifs that are suffixes that have the same amount of occurrences. Nevertheless, the identity of location lists which have been offset to zero gives interesting information which the algorithm in [PRP01] does not seem to use. For example, the identity of $L'_a$ and $L'_c$ above, both equal to $\{0, 2\}$, tells us that both $a$ and $c$ are not maximal because they can both be extended obtaining $a \cdot c$ with location list $\{1, 3\}$. In the literature of tiling motifs, there does not exist an operator that can generate such motifs by concatenation of two others whose occurrences do not overlap. We will define such an operator in section 7.6.

- When the attempt to remove suffixes is done at step 1.3a before irredundancy checking, we could at most erase motifs that are single letters, due to the way we have built motifs so far (that is after step 1.2), that would anyway be erased at the first suffix search of the second phase. In fact this is the only kind of suffixes we may have among motifs of the form $\sigma_1 \cdots \sigma_2$ with $\sigma_1, \sigma_2 \in \Sigma$. In addition, they would not have any effect on which motifs are generated or updated until that time, because, even if one of this single letter
motif would result redundant at step 1.3b, updating motifs that cause this redundancy would have no effect.\footnote{We have } Similarly, when concatenating at step 2.1, no single letter motif can contribute to a new motif generation (because, again, concatenating $m' = \alpha$ with any motif $m$ starting or ending with $\alpha \in \Sigma$ would result in $m$ itself). Therefore, the suffix removal of step 1.3a is actually of no use, except for possibly saving some useless motif generation that could be skipped in other cheaper ways. On the other hand, if suffix removal was performed \textit{after} redundancy checking, in general there would be several motifs that could be erased. Nevertheless, we do not assume that the right way is to just switch steps 1.3a and 1.3b because, as we will see, checking for redundancy before suffix removal can lead to incorrect results.

So far, we have made some assumptions concerning the algorithm given in \cite{PRP01}, for solving some imprecision that could lead to mistakes or to extra running time. We now show that there are other aspects of this algorithm that lead to incorrect results, and that we do not see for now how to solve. Therefore, we believe that this algorithm cannot identify tiling motifs correctly, nor that it can be adapted to do so with minor changes.

The second phase (the concatenation step) is repeated several times because it is iteratively executed until a fixed point is reached. Hence, it is in general unavoidable that a suffix removal takes place after a redundancy checking. It may therefore happen that some motif $x$ is removed because it results to be a suffix of another motif $x'$ without having additional occurrences with respect to it, that is, because $\mathcal{L}_x = \mathcal{L}_{x'}$. However, before being removed, motif $x$ may have contributed to the redundancy of another motif $y$ which has therefore been discarded. If without $x$, motif $y$ would have been irredundant and thus kept, then its removal would be a mistake, unless $y$ can be generated again. Therefore, it may happen that a tiling motif is eliminated because of redundancy at a certain step, and is never re-generated even after that the motifs that have caused its redundancy are no longer there. This happens because the only way new motifs are generated in the final loop is by means of the concatenation of two motifs such that the starting letter of one is the ending one of the other. It may be that a tiling motif is eliminated, as well as the only ones that could generate it by concatenation, thus disallowing their presence in the output solution.

\begin{example}\textbf{Example 3} Consider the sequence $s = aabaaacaaada$ with $|s| = n = 12$. Among its tiling motifs, there is $m = aa \cdot a$ with occurrence list $\mathcal{L}_{aa \cdot a} = \{1, 4, 5, 8\}$. It is generated at the first iteration of the concatenation phase, by concatenating $aa$ and $a \cdot a$, which both are generated at the initialization phase (none of them can be generated by concatenation), and there is no other way to generate $m$. At the\end{example}
redundancy checking phase of the same iteration, m is eliminated because its occurrence list results to be the union of the location lists of $m_1 = a aa$ ($\mathcal{L}_{aa} = \{4, 8\}$) and others (namely $a \cdots a$ with $\mathcal{L}_{a \cdots a} = \{1, 4, 5\}$). Motif $m_1$ has been generated as a concatenation of aa with itself. During the redundancy checking, $m_1$ is correctly extended to the right and it becomes equal to $aa \cdot a$. Afterwards, during the suffix removal phase, $m_1$ disappears as it is a suffix of $a \cdot aa \cdot a$ with the same number of occurrences. At this point, $m = aa \cdot a$ would no longer be redundant (and never will be, as it is indeed in the basis), but it has been eliminated and cannot be newly generated because $aa$ is also no longer there (as it has correctly been eliminated because of redundancy), and will not be re-generated either. The same would happen to the tiling motif $a \cdot aa$.

In the case of example 3, we say that a false negative error has taken place. As shown above removing motifs can lead to a false negative error when not all motifs are maximal yet. In particular, motifs that can be extended to the left, might play a dangerous role as their temporary occurrence list can lead to a wrong redundancy detection. This problem can be solved only if we could know in advance which are the occurrence lists of final maximal motifs, even if these still have to be filled with letters replacing wild cards or extended to the right. We thus need at least to complete the left extension before doing any redundancy check.

The problem of false positive can also happen with the algorithm presented in [PRP01]. That is, some motifs survive all selections and thus they belong to the output basis, without being tiling motifs. This can happen, for example, when two motifs have identical offset location lists, but neither is suffix of the other. The algorithm would keep them both, while the correct thing to do would be to extend them to a new motif which is a suitable combination of them.

**Example 4** Let $s = abaxbabaydacazbacate$ with $|s| = n = 20$. The first time that the concatenation phase takes place, both $m_1 = a \cdot ba$ and $m_2 = ba \cdot a$ are generated with location lists $\mathcal{L}_{m_1} = \{3, 13\}$ and $\mathcal{L}_{m_2} = \{5, 15\}$, respectively. For both of them, the offset location list is $\{0, 10\}$, but none of them is suffix of the other. The algorithm fails to notice that $m_1$ can be extended to the right (and $m_2$ to the left) obtaining the new motif $a \cdot ba \cdot a$ with location list $\{3, 13\}$. Instead, both $m_1$ and $m_2$ are kept even though they are not maximal. None of them will be deleted in successive steps, and thus they erroneously end up in the output.

In the next sections, we further explore the problem of finding tiling motifs. We introduce some properties of tiling motifs that might result useful in conceiving a new algorithm to find them.
Figure 7.1: The suffix tree for the sequence $s = baabbaa$. Bigger unfilled nodes are those of the compact suffix tree $\mathcal{T}$, and the smaller black ones are those that are the nodes of $\mathcal{U}$ only, and that actually do not branch. Nodes of the compact suffix tree that have crosses on them, are those from which edges labeled with ‘$S$’ should leave (and are omitted). The number labeling each node $v$ is $n_v$.

### 7.4 Some properties of tiling motifs

#### 7.4.1 Representing motifs on the suffix tree of $s$

We will now suggest a way of viewing tiling motifs on the suffix tree of $s$. This enables to better understand some of the properties that we state in this chapter. Let $\mathcal{T}$ be the compact suffix tree\(^4\) of the sequence $s$. It is known [Gus97] that $\mathcal{T}$ has $n$ leaves and at most $n - 1$ inner nodes. Each edge is labeled by a substring of $|s|$. Let the label of a path be the concatenation of the labels along all edges it meets. Each path from the root to a node of $\mathcal{T}$ is labeled by a substring of $|s|$ (prefixes of suffixes are indeed substrings of $s$). For each node $v$ of $\mathcal{T}$, let $n_v$ be the number of leaves in the subtree rooted at $v$. If the path from the root to a node $v$ spells a pattern $p$, then $n_v$ corresponds to the number of occurrences of $p$ in $s$. In the case of motifs, since in general there are wild cards, we have that a motif $m$ will be spelled by a set of paths from the root, as at each position where a wild card appears, several subpaths are taken into account. Let $\mathcal{U}$ be the uncompact suffix tree of the sequence $s$, that is the suffix tree having only one letter labeling each edge (see section 6.3). We name $V_m$ the set of nodes of the uncompact suffix tree $\mathcal{U}$ whose paths from the root spell $m$.

\(^4\)See section 6.3
Example 5 Let $s = baabaa$. The suffix tree of $s$ is shown in figure 7.1. If $k = 2$, then the tiling motifs are $m_1 = baa, m_2 = a \cdot b \cdot a, m_3 = a$ and $m_4 = b \cdot a$ with occurrences lists $L_{m_1} = \{1, 5\}, L_{m_2} = \{2, 3\}, L_{m_3} = \{2, 3, 6, 7\}$, and $L_{m_4} = \{1, 4, 5\}$. Motifs $m_1$ and $m_4$ are spelled by the paths that reach nodes $x$ and $v$, respectively. That is $V_{m_1} = \{x\}$ and $V_{m_4} = \{v\}$. Motif $m_2$ is spelled from the root to both node $y$ and node $z$, hence we have that $V_{m_2} = \{y, z\}$. Finally, $V_{m_4} = \{x, w\}$.

7.4.2 Some properties of maximal and redundant motifs

We have that $m$ is maximal if and only if

1. when we try to replace a wild card of $m$ with a letter, then we lose some occurrences; and

2. when we try to extend $m$ to the left or to the right, then we lose some occurrences.

The first property involves maximal motifs that contain at least one wild card. Let $m = m_1 \cdot m_2$ be one of them. We have that if the wild card between $m_1$ and $m_2$ is replaced, say, by the letter $\alpha \in \Sigma$, then we would get the motif $m' = m_1 \alpha m_2$ that has an occurrence list which is a proper subset of the list of $m$. This has to hold for all $\alpha \in \Sigma$. The second property concerns length extension. We will show it in terms of the suffix tree of the sequence. Let us consider right extension first, that is, we have motif $m$ with its set of paths along $U$ leading to all nodes in $V_m$. If another motif $m' = m\alpha$ (for some $\alpha \in \Sigma$) would result in the same set of occurrences as $m$, then we would have that in $U$ for all nodes in $V_m$ there is an edge leaving it that is labeled with (a word starting with) $\alpha$. Hence, in order to guarantee maximality with respect to right extension, we just have to make sure that, if the situation above holds, we can forget about $m$ and rather keep $m'$ instead (or further right extensions of it). Regarding left extension of a motif $m$, we need to check nodes whose paths in $U$ spell $\alpha m$ for each $\alpha \in \Sigma$. Such nodes can be reached by following backwards all suffix links leading to nodes of $V_m$. We recall that a suffix link in the uncompact suffix tree $U$ connects the node $v_{\alpha x}$ with $\alpha \in \Sigma$ and $x \in \Sigma^*$ (that is the one whose path from the root spells $\alpha x$), to the node $v_x$. If only one suffix link enters node $v_x$, then we have that every occurrence of $x$ is preceded by the same letter. If this is the case, then $x$ is not maximal because it can be extended to the left with at least the letter $\alpha$. The property to be checked for a motif $m$ which in general contains wild cards, is whether there exists an $\alpha \in \Sigma$ such that $\sum_{v \in V_m} n_{v\alpha} = \sum_{v' \in S_{\alpha}(m)} n_{v'}$ where $S_{\alpha}(m)$ is the set of all nodes reachable by following the suffix link backwards from a node in $V_m$, and that spell $\alpha$ as a first letter, that is $S_{\alpha}(m) = V_{\alpha m}$. This can be checked in polynomial time.

Actually, in order to spell all motifs in the basis, we may never spell all the maximal motifs because, as we know, there may be an exponential number of them. In
7.4. Some properties of tiling motifs

In fact, the algorithm suggested in the literature of tiling motifs aims to directly spell irredundant motifs, or at least to keep the invariant that at most as many as they are kept. We now list some properties of redundant motifs that could be useful in an algorithm that directly discards them as soon as they are generated. This is a necessary condition for a correct and polynomial solution to the problem of finding tiling motifs.

From the definition of redundancy, we have that a maximal motif $m$ is redundant if and only if $L_m = L_{n_1} \cup \cdots \cup L_{n_p}$ for some maximal motifs $n_1, \ldots, n_p$. We can already say that it must be $p > 1$. In fact, if we had a unique motif $n_1 \neq m$ such that $L_m = L_{n_1}$, $m$ and $n_1$ could not be both maximal. Hence, there must be more than one such maximal motifs $n_i$, and for each $i = 1, \ldots, p$, it must be that $L_{n_i} \subset L_m$. We now prove some properties of $n_1, \ldots, n_p$ that characterize them with respect to $m$ and thus, conversely, also characterize $m$ with respect to motifs that can cause it to be redundant. This characterization could be useful for the design of algorithms that find tiling motifs in a text and that have to check irredundancy of a maximal motif without having to generate all maximal motifs. Let $m$ have $h$ wild cards, that is $m = m_0 \cdot m_1 \cdot m_2 \cdots m_{h-1} \cdot m_h$, where $\forall 0 \leq i \leq h$, $m_i$ contains no wild card and is possibly empty. Any of the motifs $n_i$ may have a letter $\beta \in \Sigma$ at a position where $m$ has one of the $h$ wild cards. In fact, in such case, the number of occurrences can only decrease. And it is decreased, otherwise $m$ could not be maximal. In terms of the tree $U$, we have that paths spelling $n_i$ just choose one branch while for the motif $m$, the whole subtree is taken into account. On the other hand, the following restriction holds for the $n_i$s:

**Lemma 4** Let $m = m_0 \cdot m_1 \cdot m_2 \cdots m_{h-1} \cdot m_h$ be a maximal motif having $h$ wild cards ($\forall 0 \leq i \leq h$, $m_i$ contains no wild card and can be empty). Let $n_1, \ldots, n_p$ be maximal motifs such that $L_m = \cup L_{n_i}$. Then at each position where $m$ has a letter $\alpha \in \Sigma$, no $n_i$ can have a wild card or a letter $\beta \neq \alpha$.

**Proof.** By contradiction, assume that there is one $n_i$ such that there exists a position where $m$ presents a letter $\alpha \in \Sigma$ that is replaced in $n_i$ by, say, a wild card. Without loss of generality we can assume that this happens for the first time within the first fragment $m_0$ of $m$. That is, $m_0 = m' \alpha m''$, while motif $n_i$ has $m'$ as a prefix. Then, in the set $L_{n_i}$, there will be also occurrences of sequences starting with $m' \beta$ with $\beta \neq \alpha$ (such $\beta$ exists because $n_i$ is maximal). Such occurrences do not belong to $L_m$, which contradicts the hypothesis that $L_{n_i} \subset L_m$. With a similar argument, we can show that $\alpha$ could not be replaced by a different letter. \(\Box\)

So far, we have seen that, the motifs $n_i$ must conserve letters of $m$ over all their length, while they can replace a wild card with a letter. We now state another result on the length of $n_i$ with respect to $m$, and that also shows under which conditions the $n_i$s must have replaced wild cards of $m$.
Lemma 5 Let \( m = m_0 \cdot m_1 \cdot m_2 \ldots m_{h-1} \cdot m_h \) be a maximal motif having \( h \) wild cards with the \( m_i \)'s as above. Let \( n_1, \ldots, n_p \) be irredundant motifs such that \( \mathcal{L}_m = \cup \mathcal{L}_{n_i} \). Then we have that if \( |n_i| \leq |m| \), then \( n_i \) must have replaced a wild card of \( m \) with a letter.

Proof. By contradiction, let \( |n_i| \leq |m| \) and assume that \( n_i \) has conserved all wild cards of \( m \). By lemma 4, also letters of \( m \) are conserved. This implies that \( n_i \) is just a prefix of \( m \). However, if this is the case, we have that \( \mathcal{L}_m \subseteq \mathcal{L}_{n_i} \), which contradicts the hypothesis. \( \Box \)

As a consequence of lemma 5 we have that if \( n_i \) does not replace a wild card of \( m \) then it has to be longer than \( m \). Notice that nothing forbids the \( n_i \)'s to be longer than \( m \) even if some wild cards are replaced. Summing up, we have proved the following:

Theorem 9 Let \( m = m_0 \cdot m_1 \cdot m_2 \ldots m_{h-1} \cdot m_h \) be a maximal motif having \( h \) wild cards as above. Let \( n_1, \ldots, n_p \) be irredundant motifs such that \( \mathcal{L}_m = \cup \mathcal{L}_{n_i} \). Then each \( n_i \), \( 1 \leq i \leq p \), conserves all letters of \( m \), and may be shorter than \( m \), or of equal length than \( m \), only if a wild card of \( m \) has been replaced by a letter in \( n_i \). \( \Box \)

7.5 Characterization of solid parts of tiling motifs

Let \( T' \) be the part of the compact suffix tree where branches not satisfying the quorum are cut out. As an example, the subtree \( T' \) of the tree in figure 7.1 is the one above the dotted line. Since \( k \geq 2 \), then at least all leaves are cut out. Therefore we have that \( T' \) has at most \( n - 1 \) nodes, that is the number of internal nodes of a suffix tree [Gus97].

Let \( m \) be again a tiling motif with \( h \) wild cards, that is \( m = m_0 \cdot m_1 \cdot m_2 \ldots m_{h-1} \cdot m_h \). As above, the \( m_i \)'s are consecutive (possibly empty) solid characters without wild cards in between. We call the \( m_i \)'s factors of \( m \). We have the following:

Theorem 10 All factors of a tiling motif \( m \) are spelled by paths from the root to a node in \( T' \).

Proof. Let \( m_i \) be a factor of the tiling motif \( m \). We have that if a wild card follows \( m_i \) in \( m \), hence it must be that at least one occurrence of \( m_i \) in \( s \) would be lost if a solid character was placed. Hence, there are at least two occurrences of \( m_i \) in \( s \) that are followed by different characters. Therefore, when spelling \( m_i \) from the root of the compact suffix tree, one ends up at a node followed by a branching. And, obviously, \( m_i \) satisfies the quorum. Therefore it is a node of \( T' \). \( \Box \)
Let $\mathcal{F}$ be the set of all possible factors of tiling motifs of a sequence $s$. We have seen that since $\mathcal{T}'$ has at most $n - 1$ nodes, we have that $|\mathcal{F}| \leq n - 1$. The compact suffix tree $\mathcal{T}$ can be built using, for instance, Ukkonen’s on-line algorithm (see the original paper [Ukk95], or the description given in [Gus97]). The construction of $\mathcal{T}$ takes $O(n)$ time and $O(n)$ space. The pruning that leaves only $\mathcal{T}'$, and the traversal of the tree to find all the factors, can also be done in linear time. Therefore, a consequence of theorem 10 is that we can find all factors of all tiling motifs in linear time. Each tiling motif is, on his turn, tiled over by these factors, in the sense that it is made of elements of $\mathcal{F}$ possibly concatenated and separated by a suitable number of wild cards. Theorem 10 could be useful for designing efficient heuristics to find tiling motifs. Actually, the set of the nodes of $\mathcal{T}'$ could result in a superset of $\mathcal{F}$. In fact, a possible pruning could be performed on the set of all paths of $\mathcal{T}'$ from the root to any node. This is because it may happen that a fragment occurs in the sequence always immediately after another one. In no tiling motif this fragment will therefore be present without the one that precedes it in all its occurrences. This can also be detected in the suffix tree in polynomial time by looking at suffix links of $\mathcal{T}'$. The property to be checked is whether the conditions shown in figure 7.2 hold. It suffices to check whether there is an unique suffix link entering a node. In figure 7.2 it is indeed the case of the node spelling $f'$. The condition of identity of subtrees does not have to be checked. Figure 7.2 shows a fragment of the suffix tree which results if the sequence is such that a fragment $f'$ only appears after another fragment $f$. When this happens, then keeping $f'$ as factor is of no use as it will never appear without $f$ before. The factor $ff'$ will anyway be considered as it corresponds to a path from the root to a node of the suffix tree. In this case, $f'$ should therefore be discarded and not be considered as belonging to the set $\mathcal{F}$ of factors of tiling motifs for $s$. Notice that we have instead that $f \in \mathcal{F}$ because the factor labeled by $f'$ is not the only branch leaving the node that spells $f$. 

Figure 7.2: Details of the suffix tree for a sequence where fragment $f'$ always follows fragment $f$.
7.6 Hints for a new algorithm: the operator $\otimes_q$

We introduce now a new operator that is an extension of the $\otimes$ of [PRP01]. We will name it $\otimes_q$ as it intuitively performs a $\otimes$ among motifs that are $q$-shifted between themselves.

**Definition 12** Let $m_1$ and $m_2$ be two motifs such that $\mathcal{L}_{m_2} \cap (\mathcal{L}_{m_1} + q) \neq \emptyset$. Let us assume that for any motif $m$ we have $m[i] = \cdot \forall i > |m|$. We define $m = m_1 \otimes_q m_2$ as follows

$$m[i] = \begin{cases} m_1[i] \otimes m_2[i] & q + 1 \leq i \leq \max\{|m_1|, |m_2| + q\} \\ m_1[i] & 1 \leq i \leq q \end{cases}$$

For $q = 0$ we have that $\otimes_q$ and $\otimes$ give the same result. In fact, in that case we have that the two occurrence lists are identical, and the resulting $m = m_1 \otimes m_2$ motif is built by aligning $m_1$ and $m_2$ at their left ends resulting in a motif $m$ such that $|m| = \max\{|m_1|, |m_2|\}$. In the more general case of $m = m_1 \otimes_q m_2$, we have that $m_1$ and $m_2$ are indeed shifted, possibly not even overlapping, and the resulting motif $m$ is such that $|m| = \max\{|m_1|, |m_2| + q\}$. After steps 1.1 and 1.2 of the algorithm suggested in [PRP01] (and reported in section 7.3), we have all single letter motifs that satisfy the quorum, and all motifs of the form $\sigma_1 \cdots \sigma_2$ with $\sigma_1, \sigma_2 \in \Sigma$ with all possible amounts of wild cards. As we have seen, step 1.3, as it is now, only removes trivial motifs as it seeks for suffixes. Actually, equality of occurrence lists offset to zero brings some interesting information that could be used at that step. In fact, if two motifs $m_1$ and $m_2$ have equal offset location lists, say $\mathcal{L}_{m_2} = \mathcal{L}_{m_1} + q$, then definitely none of them is maximal, while a suitable concatenation of them might still be. Such concatenation is indeed $m_1 \otimes_q m_2$. Let us assume we replace steps 1.3a (and hence also 2.2a) with the following 1.3d' (resp. 2.2a'):

**for all** motifs $m_1, m_2, \ldots, m_p$ such that $\forall 1 \leq i, j \leq p \exists q_{ij}$ such that $\mathcal{L}_t = \mathcal{L}_j + q_{ij}$ do

/* let $m_1$ (w.l.o.g.) be the one with the leftmost first occurrence */

Generate $m = (((m_1 \otimes_{q_1} m_2) \otimes_{q_3} m_3) \cdots \otimes_{q_p} m_p)$

Keep $m$ with $\mathcal{L}_m = \mathcal{L}_{m_1}$ and discard $m_1, m_2, \ldots, m_p$

In this way, not only suffixes of $m$ that have the same occurrence list as $m$ disappear, but also submotifs (that is, fragments that are not necessarily suffixes) or abstractions (that is, with wild card replacing one or more letter) that have the same occurrences as $m$. Therefore, using the $\otimes_q$ operator could avoid the false positive error shown in example 4, as shown in the following.

**Example 6** Let us consider again the instance of example 4 with its motifs $m_1$ and $m_2$ generated after the initialization phase of the algorithm of [PRP01]. That is,
s = abaxbabaydacabacate with |s| = n = 20, and \( m_1 = a \cdot ba \) and \( m_2 = ba \cdot a \) with locations lists \( \mathcal{L}_{m_1} = \{3,13\} \) and \( \mathcal{L}_{m_2} = \{5,15\} \), respectively. It could be detected in polynomial time that for both of them the offset location list is \{0, 10\}. Equality of location list is a subcase of the condition for the applicability of the \( \otimes_q \) operator. In fact, we have that for \( q = 2 \) it holds that \( 5,15 \cap (3,13 + 2) \neq \emptyset \). Hence, we can perform \( m_1 \otimes_2 m_2 = a \cdot ba \otimes_2 ba \cdot a = a \cdot ba \cdot a \). In this way, we have obtained a new motif \( m_3 \) that can be maximal while neither \( m_1 \) nor \( m_2 \) could because \( m_3 \) is an extension of both of them with location list equal \{3, 13\} which has the same size as the ones of \( m_1 \) and \( m_2 \).

Nevertheless, applying the \( \otimes_q \) operator when equality of offset location lists holds, solves only partially the false positive problem of the algorithm of section 7.3. In fact, it could be the case that a motif \( m_i \) can be extended by means of the application of \( \otimes_q \) with another motif \( m_j \) even if the offset location lists of \( m_i \) and \( m_j \) are not equal, but the equality could be obtained for example if only a subset of \( \mathcal{L}_{m_i} \) was taken into account. In other words, there is no \( q \) such that \( \mathcal{L}_{m_i} = \mathcal{L}_{m_j} + q \), but there is a \( q \) such that \( \mathcal{L}_{m_i} \cap (\mathcal{L}_{m_j} + q) \neq \emptyset \). If \( |\mathcal{L}_{m_i} \cap (\mathcal{L}_{m_j} + q)| \geq k \), then the motif \( m_i \otimes_q m_j \) can be maximal and should be generated. Notice that the concatenation step of the algorithm does not capture all these cases. If \( \mathcal{L}_{m_i} \cap (\mathcal{L}_{m_j} + q) \) results in a set smaller than \( \mathcal{L}_{m_i} \) (resp. \( \mathcal{L}_{m_j} \)), then we have that \( m_i \) (resp. \( m_j \)) can still be maximal. On the other hand, if we have that, e.g. \( |\mathcal{L}_{m_i} \cap (\mathcal{L}_{m_j} + q)| = |\mathcal{L}_{m_i}| \), then \( m_i \) has to be discarded and substituted by \( m_i \otimes_q m_j \) because \( m_i \) is not maximal.

**Example 7** Assume that \( m = a \cdot bb \cdot c \) is a tiling motif which we did not generate yet. Let the quorum be \( k = 2 \). If we know that, say, \( \mathcal{L}_{b \cdot c} = \{i, j\} \) with \( i < j \) and \( \mathcal{L}_{b \cdot c} = \{i - 1, j - 1\} \), then we can detect that \( b \cdot c \) has to be extended with \( b \) on the left even if \( b \cdot c \) is not a suffix of \( b \cdot c \cdot b \), because their offset location lists are equal, as they are both \( \{0, j - i\} \). Therefore, we can remove \( b \cdot c \) and \( b \cdot c \) and replace them with \( bb \cdot c \). On the other hand, if we had, say, \( \mathcal{L}_{a \cdot c} = \{i - 3, j - 3, h\} \), then equality of offset location lists would be a too strong condition. In fact, it is only the subset \( \{i - 3, j - 3\} \) of \( \mathcal{L}_{a \cdot c} \) whose offset location list equals \( \{0, j - i\} \). Hence, this is the condition to be checked and verified in order to finally apply \( a \cdots c \otimes_2 bb \cdot c \) and generate \( m \). If \( h > i - 3 \), then we would still have that the offset location lists of \( a \cdots c \) is a superset of \( \{0, j - i\} \). But if \( h < i - 3 \), then this would not even be the case.

The condition to be checked in order to extend motifs \( m_i \) and \( m_j \) in length (decreasing or not the number of occurrences, and thus discarding or not the initial \( m_i \) and \( m_j \)), is in general the following. For all possible \( q, q' \), verify whether \( |(\mathcal{L}_{m_i} + q) \cap (\mathcal{L}_{m_j} + q')| \geq k \). This should be checked for all pairs of motifs \( m_i, m_j \). In this way, we can find all possible length extensions (that satisfy the quorum) of all motifs held at a certain step. Checking this condition, although time consuming, can be done in polynomial time for a polynomial number of motifs held. Thus, this
could represent a (possibly only partial) solution to the false positive problem of the algorithm of [PRP01]. On the other hand, the problem of false negatives remains open.

7.7 Further work

A modified version of the notion of basis can be conceived. This could result in a possibly smaller size of $B$, and in a possibly computationally easier problem to find it. In particular, the notion of redundancy could be strengthened in order to avoid mentioning motifs that are repeated because of periods present in one of them. In this case the current definition of irredundancy is not sufficient to avoid an over representation of such motifs. Another possible way of further decreasing the size of the basis and, hopefully, the complexity of detecting it, is to bound the number of non consecutive wild cards in a motif. This would not disallow to build submotifs that can recover longer ones by concatenation. In fact, any motif with more wild cards could be possibly built by concatenation of these ones.

An important extension of the notion of tiling motifs is to allow different lengths for the occurrences of the same motifs, that is to allow insertions and gaps. The notion of flexible motifs of [PRP01] goes in this direction, and deserves attention provided that finding such motifs will result computationally feasible.
Chapter 8

Improving the extraction of structured motifs

8.1 Introduction

In this chapter we present a possible improvement for extracting structured motifs [MS00, MS01]. Given a text $s$, the problem is to find repeated patterns in $s$, that is motifs, according to some parameters that specify the frequency and the structure required for them. The problem is interesting for many text mining applications. The one we are concerned with is in molecular biology where the text is a DNA sequence. Interesting applications of structured motifs extractions in this domain have already been done using suffix trees [VMS99, VMLS00, MS00, MS01] with interesting performances. The problem still deserves attention from the algorithmic point of view because its computational complexity is exponential with respect to the number $e$ of mismatches allowed among different occurrences of the same motif. This imposes a limit to the length of the motifs themselves, as the values of $e$ depends on this length. We suggest an improvement that can be applied to a wide range of motif extraction algorithm that use suffix trees, such as those presented in [Sag98, MS00, MS01], and in particular to structured motifs extraction (i.e., motifs composed of several disjoint single motifs placed at given distances among each other). We will motivate the belief that its benefits should result particularly interesting for structured motifs and that the modifications we suggest can be applied to such instances. Nevertheless, this improvement applies already to single motive extraction [Sag98]. In fact, the algorithm for single motif extraction introduced in [Sag98] is the ancestor of the one presented in [MS01] for structured motifs.

In section 8.2 we present some variants of structured motifs extractions and the solution suggested for this problem. Section 8.3 describes the modifications we suggest to the existing algorithms in order to improve their performance. Finally, section 8.4.1 shows some preliminary experimental results that motivates our beliefs
that the modification is worthwhile.

8.2 Structured motif extraction

In [MS00, MS01], some algorithms for different variants of *structured motifs extraction* are presented. A structured motif is defined by the following parameters.

- An ordered collection of \( p \geq 1 \) “boxes”, each one corresponding to a single motif of length between a given minimal one \( k_{\text{min}} \) and a maximal one \( k_{\text{max}} \). Notice that we may have \( k_{\text{max}} = \infty \), which corresponds to the case in which only a minimal length is required for the motifs.

- \( p \) maximum error rates allowed \( e_1, \ldots, e_p \) (one for each of the \( p \) boxes), and a maximum total error rate \( e \).

- \( p - 1 \) intervals of distance, one for each pair of consecutive boxes.

- A quorum \( q \), that is the minimum amount of occurrences required to the motif.

Given these parameters, the algorithm searches for the contents of the boxes, that is the motifs, that have the structure defined by the parameters above and that satisfy the quorum. The variants addressed in [MS00, MS01] differ in the value allowed for \( p \) (\( p = 2 \) or \( p \geq 2 \)) and in the way the intervals are described. In this chapter we refer to the simplest case, that is with \( p = 2 \), and where the unique interval is given by a pair \((d_{\text{min}}, d_{\text{max}})\), respectively the minimal and the maximal distance between the two boxes. The modifications to the algorithm that we suggest in this chapter can be extended to all the more complex variants of the problem, as they do not depend on whether \( p \) equals 2 or is bigger, nor on the way the intervals are constrained. Moreover, we refer to the problem of finding such motifs in a single input sequence, as once again the extension to the case of \( N \) sequences as input, and thus to generalized suffix trees, is straightforward.

The motifs output by the algorithm of Marsan and Sagot are represented by words on the alphabet \( \Sigma \), which is that of the input sequence. Given an error rate \( e \), a motif is acknowledged as occurring at a certain position in the sequence if it occurs with at most \( e \) letters substitution. In the case of structured motifs, there is an error rate \( e_i \) for each box \( i \), as well as a global one \( e \). Given the quorum \( q \), a motif is *valid* if it occurs at least \( q \) times in the input sequence. The algorithm in [MS01] outputs all valid motifs that are structured according to the length of the boxes and their distance.

The structured motif extraction algorithm makes use of the concept of *model* of a motif that had been introduced in [Sag98]. A model is a word on the alphabet
\(\Sigma\) that represents a motif. That is, given a model \(m\) and an error rate \(e\), we say that \(m\) occurs at a certain position if it occurs with at most \(e\) substitutions. The model does not necessarily appear in the sequence in an exact way (that is, with no substitutions), as it is an external concept. The notion of model has been introduced to distinguish the model (indeed, the motif) and its occurrences. The definition of valid motif is extended to models. That is, when a model satisfies the quorum, we call it a valid model. In a few words, the algorithm\(^1\) first builds the suffix tree \(\mathcal{T}\) of the input sequence, then it searches for all valid models of length at least \(k_{\min}\) and up to \(k_{\max}\) and, for each one of them, it checks whether there is a second valid model with the required interval between them. Models are considered in lexicographical order starting from the empty word, and they are extended as long as the quorum is satisfied until either a valid model of maximal length is found (if the \(k_{\max}\) length is reached), or the quorum is no longer satisfied. In both cases, a new model is taken into account. This search makes use of the suffix tree \(\mathcal{T}\) where each node \(v\) also contains information concerning the amount of occurrences of the model spelled by the path from the root to \(v\). In particular, at each moment all nodes spelling occurrences of the current model are taken into account. More formally, the algorithm presented in \[MS01\] we refer to is described by the following procedure, where the model \(m\) is the one whose extension is being attempted, and the value \(i\) indicates whether we are dealing with the first or the second "box". Finally, \(\lambda\) denotes the empty word.

\[\text{procedure ExtractMotif(model } m, \text{ box } i)\]
\begin{align*}
\text{begin} & \\
& \text{for all } \alpha \in \Sigma \text{ do} \\
& \quad \text{if } ma \text{ is valid then} \\
& \quad \quad \text{if } |ma| = k_{\max} \\
& \quad \quad \text{then} \\
& \quad \quad \quad \text{if } i = 2 \\
& \quad \quad \quad \text{then return the complete valid motif} \\
& \quad \quad \text{else suitably moving in } \mathcal{T}, \text{ ExtractMotif}(\lambda, 2) \\
& \quad \text{else ExtractMotif}(ma, i) \\
\text{end}
\end{align*}

At the beginning \(\text{ExtractMotif}\) is called on the empty word and with \(i = 1\). Successively, the procedure recursively calls itself for longer models until possibly considering the second box, and it considers new ones when the extension fails. The new one is either chosen by replacing the last letter \(\alpha\) with a new one (according to the ‘for’ of line 1), or, when all letters have been tried, with a new shorter model where the penultimate letter is replaced (unless this was not again the last letter, and so on). The order in which models are generated actually equals a depth-first

\(^{1}\)The algorithm we refer to here is the first algorithm introduced in \[MS01\].
visit on a virtual complete trie \( \mathcal{M} \) of all strings of length \( k_{\text{max}} \) on the alphabet \( \Sigma \). We will refer to \( \mathcal{M} \) as the model tree. The algorithm of [MS01] does not need to allocate the model tree \( \mathcal{M} \). It is only mentioned as a virtual data structure whose depth-first visit describes the order in which motifs are considered. The only memory requirement is for the suffix tree \( \mathcal{T} \). Indeed, the conditions at lines 2 and 3 are checked on the tree \( \mathcal{T} \) which is traversed during the motif extraction as all nodes whose path from the root spell occurrences of the model that is being processed are taken into account. The jump over \( \mathcal{T} \) mentioned in line 7 basically consists in considering as potential starting positions for the motif that is the second box, all nodes from \( d_{\min} \) to \( d_{\max} \) deeper in all subtrees of \( \mathcal{T} \) rooted at nodes that have spelled occurrences of the valid model for the first box. The algorithm is actually optimized in such a way that only the upper \( 2k_{\text{max}} + d_{\max} \) deep part of \( \mathcal{T} \) is built and used. Therefore, it is actually only a factor tree (rather than a suffix tree) that is allocated. Again, we ignore this detail as our modification is independent from it.

Assuming that the required length of the motif is \( k \) for each box (that is \( k_{\min} = k_{\max} = k \)), and at most \( e \) mismatches are allowed, the algorithm has time complexity in \( O(n\nu(e,k)) \), where \( n \) is the size of the input sequence, and \( \nu(e,k) \) is the number of words of length \( k \) that differ of at most \( e \) letters from a word \( m \) of length \( k \). This value does not depend on \( m \), and it holds that \( \nu(e,k) \leq k^e|\Sigma|^e \). Therefore, the time complexity is possibly exponential in the number \( e \) of mismatches. Since reasonable values for \( e \) are proportional to the value of \( k \), this actually places a practical bound on the length required for the motifs. Finally, the space complexity is \( O(n) \), which is the space required by the suffix tree [Ukk95]. For any further detail concerning the algorithm for structured motifs extraction, we refer the interested reader to [MS01].

### 8.3 Using maximal extensibility of factors to speed up the computation

We initially assume that \( p = 1 \), that is the motif is not a structured one. In this case, the algorithm shown above still holds, provided lines 4, 5, and 7 are ignored as well as the second parameter of the procedure. When \( p = 1 \), we talk about single motif extraction, and the algorithm we refer to is basically the one presented in [Sag98].

### 8.3.1 The idea

Intuitively, the modification we suggest consists in storing information concerning maximal extensibility of factors in order to avoid trying to extend hopeless motifs. For example, let us assume that in our virtual depth-first visit of the model tree, we have found out that the model \( m \) can be further extended without losing the quorum up to a length of \( \text{MaxExt}(m) \) (maximal extensibility) only. If later on we
are processing a model $m'$ that has $m$ as a suffix, then the $MaxExt(m)$ information could result useful, as it applies to $m'$ as well. Indeed, this latter can also be extended at most $MaxExt(m)$ (and possibly less). In particular, we have that if $|m'| + MaxExt(m) < k_{\min}$, then we can avoid any further attempt to extend $m'$.

### 8.3.2 Storing the extensibility information

As we have seen in section 8.2, models are taken into account in lexicographical order by a depth-first (virtual) visit of the model tree. Every time we stop extending a model, that is, when we (virtually) backtrack in $\mathcal{M}$, it is either because we found a model of the required length (line 6), or because we reached a dead end where the quorum is no longer satisfied ($m_\alpha$ does not satisfy the condition at line 2, and we start to consider the next one in lexicographical order). More formally, the analysis of the model $m = \sigma_1, \ldots, \sigma_{|m|}$ with $\sigma_i \in \Sigma$, $\forall i = 1 \ldots |m|$ is abandoned when either

- $|m| = k_{\max}$, and then there is no reason to extend further, or
- $m$ does not satisfy the quorum.

In the first case, no information on the maximal extension of $m$ nor of its prefixes can be of any use because all models having a prefix of $m$ as suffix can in general still be extended as much as necessary to reach again the length $k_{\max}$. In the second case, $m$ does not satisfy the quorum while all its prefixes did. This case can be subdivided into two subcases according to whether or not the model $m$ had reached the length $k_{\min} - 1$ before loosing the quorum. Since, for reasons that will appear clearer later, we chose to only use the maximum extensibility information of models of length up to $k_{\min} - 1$, then we need to distinguish the two subcases. When a model $m$ cannot be extended anymore and it has not reached the length $k_{\min} - 1$, we set $MaxExt(m) = 0$ (that is, we add an ‘else’ branch to the ‘if’ command of line 2, where we insert this command). If the model has reached a length $h$ between $k_{\min} - 1$ and $k_{\max}$ (this latter value corresponds to the first case above), then we set the $MaxExt$ value at its prefix $m_0$ of length $k_{\min} - 1$, which will mean $MaxExt(m_0) = h - (k_{\min} - 1)$. Since it can be that the values of $MaxExt(m_0)$ had already been set due to the interruption of the extension of another (lexicographically smaller) extension of $m_0$, then we change the value of $MaxExt(m_0)$ only if we are increasing it. In fact, the maximum extensibility of a model refers to the its extension that reaches the highest length.

**Example 8** Let us assume that we are dealing with a DNA sequence, that is the alphabet is $\Sigma = \{A, C, G, T\}$, and let $k_{\min} = 5$ and $k_{\max} = 10$. Assume that the algorithm has performed $ExtractMoti AntarCTAT$ which has terminated because $ATCTAT$ cannot be further extended without loosing the quorum. At this point, since we only store maximum extensibility information for models of length up to $k_{\min} - 1 = 4$, and since $|ATCTAT| = 6 > k_{\min} - 1$, we do not set $MaxExt(ATCTAT)$
(which would be equal to 0), but rather \( \text{MaxExt}(ATCT) := 6 - 4 = 2 \). Assume that we later process the model \( ATCTCA \) that also results no longer extendible. This cause a new assignment to \( \text{MaxExt}(ATCT) := 7 - 4 = 3 \) (because \( |ATCTCA| = 7 \)) which is performed as it increases the value of \( \text{MaxExt}(ATCT) \). Finally, assume that also \( ATCTGT \) cannot be extended further. In this case, the updating of \( \text{MaxExt}(ATCT) \) would be again \( 6 - 4 = 2 \), and then it is not performed because it would decrease its current value.

In all the cases above, the algorithm does not consider any further extension of \( m \), and backtracks. This backtracking consists in either replacing the last letter \( \sigma_{m|} \) of \( m \) (still in the ‘for’ command), or considering a shorter model which in general shares a prefix with \( m \), if \( \sigma_{m|} \) was the last letter of the alphabet \( \Sigma \). In this latter case, the whole subtree rooted at the node spelling \( \sigma_1 \ldots \sigma_{m|-1} \) has been (virtually) completely visited. Therefore, we have all the information necessary to set the value of \( \text{MaxExt}(\sigma_1 \ldots \sigma_{m|-1}) \). We can do it according to the following general formula, for all valid models \( x \) such that \( |x| < k_{\text{min}} \).

\[
\text{MaxExt}(x) = 1 + \max_{\alpha \in \Sigma} \text{MaxExt}(x\alpha)
\]

suitably adding a command at the end of the ‘for’ cycle in the \textit{ExtractMotif} procedure. If the letter \( \sigma_{m|-1} \) was the last of the alphabet, then the backtrack goes further. In that case, also the \text{MaxExt} information concerning the word \( \sigma_1 \ldots \sigma_{m|-2} \) can be filled in in the same way, and so on as long as we (virtually) climb up in the tree. Notice that maximum extensibility information for models that do not satisfy the quorum is never stored nor used.

**Example 9** Let us consider again a DNA sequence and an instance where \( k_{\text{min}} > 4 \). Assume that the execution of the algorithm has gone far enough to have set \( \text{MaxExt}(ATCA) = 3 \), and \( \text{MaxExt}(ATCC) = 2 \), as well as \( \text{MaxExt}(ATCG) = 4 \) and \( \text{MaxExt}(ATCT) = 1 \). Due to the order in which the models are taken into account, these values are set in the same order we mentioned them. When \( \text{MaxExt}(ATCT) \) is set, it must be that all the models in the subtree rooted at the node spelling \( ATCT \) have been processed, and the assignment of this value is the last operation done within the call of \textit{ExtractMotif}(ATCT). This latter had been called in turn by \textit{ExtractMotif}(ATC), which has thus terminated the ‘for’ cycle and can set the value \( \text{MaxExt}(ATC) = 1 + \max \{3, 2, 4, 1\} = 5 \).

We have seen in section 8.2 that the model tree \( \mathcal{M} \) is actually not allocated and that it is visited only virtually. Nevertheless, the most natural and efficient way to store the \( \text{MaxExt} \) information is exactly on a trie like \( \mathcal{M} \). Actually we need less than that because we will never store data concerning subtrees that correspond to models with no quorum. Let us keep on naming model tree the trie obtained from \( \mathcal{M} \) by pruning subtrees corresponding to models with no quorum. We observe that the
upper part of $T$ strongly resembles the model tree if the former was not compact. There are two possible types of differences between the model tree and the $k_{\text{min}}$ deep upper part of $T$. First, there are paths of $M$ that are also in $T$, but some of its nodes may be missing. Second, if all occurrences of a valid model appear with at least one mismatch, then the model itself never appears exactly in the sequence, and therefore there is no path that spells it in $T$. In this case, the suffix tree is missing not just nodes, but also branches with respect to the model tree. We claim that, since we are dealing with the upper part of $T$, then these missing nodes and edges are not many. We will motivate this claim in section 8.4.1. We therefore store the $\text{MaxExt}$ information in the nodes of $T$, where we possibly add missing nodes as well as missing arcs. We will name such extra nodes \textit{ghost} nodes, and we will talk about adding ghost nodes also when it is actually arcs that are added. Notice that ghost nodes are ignored for all uses of $T$ that do not involve maximum extensibility data. The modifications we mentioned above only involve the $k_{\text{min}}$ highest levels of $T$ and, among them, only arcs corresponding to valid models. The most time and space consuming aspects of this new information we add to the tree possibly involve those corresponding to the lowest levels of the model tree, that is level $k_{\text{min}}$, level $k_{\text{min}} - 1$, and so up. We will show in section 8.3.4 that we can actually avoid having to store maximum extensibility information in at least one of these lower levels.

**Example 10** Let us suppose that the input sequence is $s = \text{abbabc}$ whose suffix tree is shown in figure 6.1 in chapter 6. The alphabet is $\{a,b,c\}$. For the purposes of this example, we can ignore the $\$ \text{ character. Let the quorum be } q = 2, \text{ the error rate } e = 1, \text{ and } k_{\text{min}} = 3. \text{ The first maximum extensibility information to be stored is the value of } \text{MaxExt}(a) \text{ for which we already need to create a ghost arc leaving the root. If we had to store } \text{MaxExt}(b), \text{ no ghost node would be required. Finally, if we need to store } \text{MaxExt}(bb), \text{ then we have to add a ghost node inside the (already existing) arc labeled by } \text{abbac that leaves the node spelling } b.\text{\textit{}}$

### 8.3.3 Fetching and using extensibility information

We start by proving the following result.

**Lemma 6** Let $w \in \Sigma^*$. We have $\text{MaxExt}(w) \leq \text{MaxExt}(v)$ for each $v$ which is a suffix of $w$.

**Proof** Letting $\text{MaxExt}(w) = k$, there exists $s \in \Sigma^k$ such that the model $ws$ is valid, that is it appears in at least $q$ sequences, and no longer string in $\Sigma^*$ has the same property. Let us now assume that there is a suffix $v$ of $w$ such that $\text{MaxExt}(v) < k$. Then there exists $t \in \Sigma^j$ with $j < k$, and no longer $t$, such that the model $vt$ is valid. However, we know that there exists $s \in \Sigma^k$ such that $ws$ appears in at least $q$ sequences. Since $vs$ is a suffix of $ws$, and since it satisfies the quorum, then the hypothesis is contradicted. \qed
As mentioned in the last section, the MaxExt information can be used for models whose extension is being taken into account and for which this information could actually prevent some useless attempts. Namely, assume we are trying to extend the model \( m = \sigma_1, \sigma_2, \ldots, \sigma_{|m|} \). Since the models are considered by means of a depth-first search on the virtual model tree, we obviously do not know the value of \( \text{MaxExt}(m) \) yet. Moreover, we know \( \text{MaxExt}(\sigma_2, \ldots, \sigma_{|m|}) \) only if it lexicographically precedes \( m \) (that is, it has already been virtually visited in the model tree). If this is not the case, we check whether \( \text{MaxExt}(\sigma_3, \ldots, \sigma_{|m|}) \) is already known, and so on, possibly until the singleton \( \sigma_{|m|} \). If they are all lexicographically greater than \( m \), then no maximal extension information can be used for \( m \), but if for any of them \( \text{MaxExt} \) is known and it holds that the maximal possible extension is not enough to reach \( k_{\text{min}} \), then the information is useful as it guarantees that attempting to further extend \( m \) is useless. Lemma 6 has the consequence that longer suffixes of \( m \) can give us more tight bounds on the maximal extensibility information with respect to shorter ones. Therefore, since we start by checking the longest one, as soon as we find a suffix of \( m \) that enables us to state that \( m \) is not worth further attempts, then we can stop checking the other (shorter) suffixes. That is, if we find a suffix \( m' = \sigma_i, \ldots, \sigma_{|m|} \) of \( m \) with \( 1 < i \leq |m| \) such that \( \text{MaxExt}(m') \) is not enough for \( m \) to reach \( k_{\text{min}} \), then we can quit attempting \( m \) and all its extensions (and we consequently update \( \text{MaxExt}(m) \)). Therefore, we actually try all proper suffixes only when no maximal extensibility information of suffixes results useful. Notice that this is equivalent to search for all proper suffixes of \( m \) of decreasing length. These can be recovered just by following the suffix links starting from the (possibly ghost) node of \( T \) that spells \( m \). If no suffix \( m' \) of \( m \) is such that \( \text{MaxExt}(m') + |m| < k_{\text{min}} \), then the maximal extension does not disallow to reach \( k_{\text{min}} \). In this case we have to go on trying to extend \( m \) even if it might be the case that it will never reach the minimal length. Written in pseudo code, when we enter the \( \text{ExtractMotif}(m) \), before anything else, the following additional commands have to be executed. Let \( x \) be the (possibly ghost) node that spells \( m \) on \( T \). For simplicity, we identify a node \( x \) and the string spelled by the path from the root to \( x \). Finally, let \( a < b \) mean that \( a \) is lexicographically smaller than \( b \).

\[
\text{repeat } x' := \text{SuffixLink}(x') \text{ until } (x' = \text{root} \text{ or } x' < m);
\]
\[
\text{if } x' \neq \text{root} \text{ then }
\text{if } \text{MaxExt}(x') + |m| < k_{\text{min}} \text{ then }
\begin{align*}
\text{begin} \\
\text{MaxExt}(m) := \text{MaxExt}(x'); \\
\text{stop spelling } m; \\
\text{end;}
\end{align*}
\]

where ’stop spelling \( m \)’ will consist of exiting from the \( \text{ExtractMotif}(m, i) \). Lemma 6 guarantees us that stopping at the first suffix (that is, the longest) that gives useful
information is enough because no shorter suffix can give tighter maximal extensibility information. Otherwise, we perform the usual operations to try to extend \( m \).

**Example 11** Let us assume again that \( \Sigma = \{A, C, G, T\} \) and that we have stored on \( T \) the values \( \text{MaxExt}(AA) = 7 \), \( \text{MaxExt}(CAA) = 2 \), and \( \text{MaxExt}(GCAA) = 1 \). Suppose that \( k_{\min} = 8 \) and that the procedure \( \text{ExtractMotif}(GTGCAA) \) is called. Before trying to extend \( GTGCAA \) attempting to append to it each one of the four possible letters, we check whether maximum extensibility information can be useful. To this purpose, we first consider \( x' = TGCAA \) and, since \( x' \not\supset GTGCAA \), we know that we do not have any maximum extensibility information concerning \( x' = TGCAA \). We therefore try with \( x' = GCAA \) which results to be lexicographically smaller that \( GTGCAA \). That is, we notice that \( x' \prec GTGCAA \), and we go to fetch \( \text{MaxExt}(x') \) which is equal to 1. Since we have that \( 1 + |GTGCAA| = 7 < k_{\min} = 8 \), then we set \( \text{MaxExt}(GTGCAA) := 1 \) and we immediately exit \( \text{ExtractMotif}(GTGCAA) \). In fact, the maximum extension of \( GCAA \) is small enough to guarantee that \( GTGCAA \) would never reach \( k_{\min} \). Considering shorter prefixes such as \( CAA \) and \( AA \), would not result in more interesting bounds for the extensibility of \( GTGCAA \). In particular, both \( \text{MaxExt}(AA) = 7 \) and \( \text{MaxExt}(CAA) = 2 \) would not have been small enough to proof that \( GTGCAA \) could not reach the length \( k_{\min} \).

Let us now consider a different situation. Let \( TTAGC \) be the model being processed, and let \( k_{\min} = 7 \). Assume that we have \( \text{MaxExt}(TAGC) = 3 \), then we cannot know anything about whether or not \( TTAGC \) will reach \( k_{\min} \). In fact, it holds that \(|TTAGC| + \text{MaxExt}(TAGC) = 10 > k_{\min} \) and thus the maximum extensibility information can not be used for \( TTAGC \). Notice that it can still be the case that this latter will never be extended in such a way to reach \( k_{\min} \) because we have that \( \text{MaxExt}(TTAGC) \leq \text{MaxExt}(TAGC) \). Nevertheless, the only way we have in order to check whether \( TTAGC \) is a prefix of a valid model, is to try to extend it with at least one more letter. Finally, we remark that if the suffix \( TAGC \) has not given a tight enough \( \text{MaxExt} \) to stop extending \( TTAGC \), then also shorter suffixes would not. This is because shorter suffixes of \( TTAGC \) are also suffixes of \( TAGC \), and lemma 6 tells us that their \( \text{MaxExt} \) is not smaller that 10, that is the one of \( TAGC \).

As an example of a third different case, let us assume that we are processing the model \( AAAAA \). Independently from the value of \( k_{\min} \), no maximum extensibility information can be used because no backtracking on \( M \) has taken place yet, and thus no \( \text{MaxExt} \) value has been assigned.

### 8.3.4 Minimizing the amount of ghost nodes

We have seen in the last section that, due to the order in which models are taken into account, we have that only certain subwords of models can give useful information
concerning maximum extensibility, namely, those that are lexicographically smaller. Since no model is smaller than itself, we actually only use the $MaxExt$ information of models that are shorter than the current one, that is, they are proper suffixes. Therefore, since the condition to check is whether or not we can hope to reach $k_{\min}$ length, we then make use of the $MaxExt$ data only for strings of length at most $k_{\min} - 1$. Hence, it is not necessary to store this information for models that have length $k_{\min}$ or more for the purpose mentioned above. No ghost node is therefore created at level $k_{\min}$ or lower of the suffix tree $T$. Moreover, the $MaxExt$ information regarding a model $m'$ of length $k_{\min} - 1$ will only be possibly used (again, assuming that our goal is just to check whether a model cannot reach $k_{\min}$ length) by a model $m$ of length $k_{\min}$, and only if $MaxExt(m') = 0$, because any longer maximum extension still allows $m$ to reach $k_{\min}$. A maximal extensibility equal to zero can be useful for saving the $|\Sigma|$ calls of $ExtractMotifs$ corresponding to the different letters with which we try to attempt to extend. It cannot be used to save more than this. Nevertheless, these attempts are still quite time consuming because they involve all nodes of the tree that spell motifs that have at most $c$ substitutions with respect to the model. Moreover, it may happen that the suffix tree is such that at level $k_{\min} - 2$ there is no need for additional nodes in order to store maximal extensibility information. Our choice is therefore in general to store maximum extensibility information at level $k_{\min} - 1$ and above, that is for models of length $k_{\min} - 1$ or less. Nevertheless, which is the best choice in this sense actually depends from the specific applications where the advantages have to be weighted with the costs. For applications in which the space cost is more important, and in which the tree misses nodes at level $k_{\min} - 1$ (and thus ghost ones would have to be added), the $MaxExt$ data for strings of length $k_{\min} - 1$ can be avoided, and thus ghost nodes not created at that level. Otherwise, if the cost or the probability of having to add ghost nodes is negligible, then the information at level $k_{\min} - 1$ should rather be stored. In order to make such choices, it can result useful a suitable preprocessing of the input sequence(s) that also takes into account the value of the input parameters. We will find further motivations in this sense in sections 8.3.6 and 8.4.1.

### 8.3.5 The case of structured motifs

Let us now consider the case of $p = 2$, that is we are searching for motifs which are composed of two boxes. In this case the conditions for our optimization to be applicable may hold even more frequently with respect to the case of single motifs. In fact, since the search for a valid motif as second box is made after a valid motif for the first box is found (and for each one of these cases), maximum extensibility information may be known also for the whole model whose extension is attempted and not just for its prefixes. That it, it may happen that when the procedure $ExtractMotif$ with parameters $m$ and 2 is called, the value of $MaxExt(m)$
is already known. Proper suffixes are thus not the only candidates to give useful information when we are trying to find a motif for the second box. On the other hand, any failure in attempting to extend a model during the search of a second box cannot update any value of MaxExt because it refers only to parts of the text that follow a specific first box. The same observation leads to another difference in the modifications required by the algorithm with respect to the case of $p = 1$. When a model $m_2$ for the second box is being extended, a fixed valid motif $m_1$ is already assigned for the first box. Therefore, this time the condition for the suffix to give useful MaxExt information is to have that $x' \preceq m_1$ rather than $x' \preceq m_2$. Finally, when the first box $m_1$ of a structured model is fixed, then the maximum extensibility information that concerns the whole sequence are in general only upper bounds on those that concern the fragments of the sequence that are at the given distance from all occurrences of $m_1$. In terms of the suffix tree $T$, this can be seen by observing that at the top level of the tree there are many more paths than in all the subtrees rooted in nodes that are suitably below (that is, from $d_{\min}$ to $d_{\max}$ lower) nodes spelling occurrences of $m_1$. In particular, the former is a superset of the latter. Therefore, while attempting to find a model for the second box, we could find tighter maximum extensibility information which we can use as long as the first box is fixed. Of course, each time that the algorithm backtracks to a new first box, then the global values for MaxExt should be restored. Provided that the observations made in this section are taken into account, the modifications to the algorithm for the case of $p = 2$ (and similarly for $p > 2$) are straightforward with respect to those we described for the case $p = 1$.

**Example 12** Let us suppose that $p = 2$, $\Sigma = \{A, C, G, T\}$, and $k_{\min} = 5$. Moreover, let $d = d_{\min} = d_{\max}$ be the distance required between the two boxes. Let us assume that we have found a valid motif GCCCA for the first box. We then set $i = 2$ and we call ExtractMotif($\lambda, 2$) taking into account as starting positions all nodes of $T$ that are $d$ levels below nodes that spell occurrences of GCCCA. Let us name $T_{(GCCCA, d)}$ this portion of $T$. Figure 8.1 shows $T_{(GCCCA, d)}$ for $e = 1$. Hence, the paths where the occurrences of the possible second box are searched is $T_{(GCCCA, d)}$ which is only a proper subset of the paths that are considered when searching for a first box (that are all the paths of $T$). It is then intuitive that in general, for all motifs $m$, we have that $m$ occurs in $T$ at least as much as it occurs in $T_{(GCCCA, d)}$. Moreover, MaxExt($m$) is also in general more tight when $m$ is searched in a subset $T_{(GCCCA, d)}$ of $T$ only, which corresponds to the subsequences of the input sequence that appear $d$ bases after an occurrence of GCCCA. Therefore, the MaxExt information concerning the whole sequence (which we name global) still applies to the model of the second box. Moreover, such information can be available already for motifs that are lexicographically greater than the one being processed for the second box, provided they are smaller than the fixed first box. Nevertheless, the actual local maximum extensibility information concerning the fraction of the sequence represented in $T_{(GCCCA, d)}$ could result more interesting because it would be more tight. If
Figure 8.1: Area $\mathcal{T}_{(GCCCA,1,d)}$ where the second box has to be searched once the first one is GCCCA. Nodes $x$ and $y$ are the occurrences of GCCCA. The subtrees rooted at nodes that are $d$ levels below $x$ and $y$ and that are their descendants compose $\mathcal{T}_{(GCCCA,1,d)}$.

For example, the global value of MaxExt(AAA) is, say, 4, then this means that motifs that start with AAA are at most at least 7. If none of the occurrences of such motifs is placed $d$ letters after an occurrence of GCCCA, then the value of MaxExt(AAA) local to $\mathcal{T}_{(GCCCA,e,d)}$ is strictly lower than 7. Therefore, for each fixed first box such as the GCCCA mentioned in this example, it could be worth storing the local maximum extensibility information. Of course, there is a local maximum extensibility information for each different possible first box. Finally, when a new first box is searched, then the global values of MaxExt must be restored.

8.3.6 A further optimization

We have seen in section 8.3.3 how to use maximum extensibility information. So far, we have been using such information only in order to avoid extensions that cannot lead to the minimum required length. In this section, we show that the same information (that is, concerning only models of length up to $k_{\min} - 1$) can also be used to save a possibly useless attempt to extend further a model that has reached or passed length $k_{\min}$, but whose processing continues since it has not reached the length $k_{\max}$ yet. For this purpose, the following conditions have to be verified. Assume that a model $|m|$ has a suffix $|m'|$ whose MaxExt is already known, and such that $k_{\min} \leq |m| + \text{MaxExt}(m') < k_{\max}$. In this case, we have that the informa-
tion concerning \( m' \) does not disallow \( m \) to reach \( k_{min} \), but it also tells that \( k_{max} \)
will never be reached. In section 8.3.3, we had just included this case in the more
general one where \( |m| + \text{MaxExt}(m') \not< k_{min} \), and we ignored this information.
Assume, instead, that if it holds that \( k_{min} \leq |m| + \text{MaxExt}(m') < k_{max} \), then, we
set a variable \textit{countdown} with the value of \( \text{MaxExt}(m') \), and we decrease the value
of this variable at each successive call of \textit{ExtractMotifs} on extensions of \( m \). The
value of \textit{countdown} can result useful if and when the extension of \( m \) reaches length
\( |m| + \text{MaxExt}(m') \) (that is when \textit{countdown} = 0), as we can avoid to try to extend
further. We “only” save a one letter attempt to extend, but this is not negligible because checking the quorum has to be done for all the \( |\Sigma| \) possible next letters, and
for each one of these, all occurrences of the model (corresponding to as many nodes
in the suffix tree) have to be handled at the same time.

What we just described is a possible way to use maximum extensibility information
also for models that are longer than \( k_{min} \). Nevertheless, we still store and use \( \text{MaxExt} \)
data only on models up to length \( k_{min} \). Notice that reasonable choices of the
parameters include the case of \( k_{max} = \infty \), which makes the further improvement
presented in this section particularly interesting. Moreover, this further improvement
does not require any additional cost in terms of ghost nodes to be added.

\textbf{Example 13} Let us consider the situation of the example 8. That is, \( k_{min} = 5 \)
and \( k_{max} = 10 \) and \( \text{MaxExt}(ATCT) = 3 \). Assume now that we are processing
the motif CATCT which has reached the length \( k_{min} \) but is still being processed
in order to be extended possibly up to length \( k_{max} = 10 \). When we enter the
\textit{ExtractMotif}(CATCT), we detect that \( |CATCT| + \text{MaxExt}(ATCT) = 8 \), which is a value smaller than \( k_{max} \). We therefore set a variable \textit{countdown}
equal to 3. Successively, if and when the extension of CATCT will cause the call
of \textit{ExtractMotif}(CATCTxyz) with suitable \( x, y, z \in \Sigma \), it will be detected that
\textit{countdown} = 0, and thus that CATCTxyz has reached length 8 which is its maximum extension.

When \( k_{min} \neq k_{max} \), and especially when \( k_{max} = \infty \), one could theoretically
use also maximum extensibility information concerning models that are longer than
\( k_{min} - 1 \) bases. In particular, avoiding a useless attempt to extend by one more
letter as shown earlier in this section, is something that could also be done making
use of \( \text{MaxExt}(m) \) for models \( m \) such that \( |m| \geq k_{min} \). In practice, this can lead to
too many ghost nodes and paths to be allocated, and thus to excessive extra space
costs. Nevertheless, there can be applications for which the cost of storing maximum
extensibility data remains negligible at least for a few levels of \( T \) that correspond
to models of length higher than \( k_{min} \). We will see in the next section how, given the
size of the input sequence, the cost of adding ghost nodes can be quickly predicted
by means of statistical considerations.
8.4 Experimental results

8.4.1 Preliminary results

Before implementing the modification introduced in this chapter, we have performed some preliminary studies that led us to predict a sensible improvement in both single and structured motif extraction when using maximal extensibility information. Maximal extensibility can definitely save some hopeless visits, but it might cost extra space which is a delicate issue for such large data sets. Therefore, the questions to be answered are:

1. how much extra space do we need?

2. how many visits do we save?, and finally

3. considering the answers to the previous two questions, will the modification improve the algorithm?

In order to answer to the first question, we need to evaluate how many ghost nodes we expect to have to add to the already allocated tree $T$. We say that a tree is uncompact complete if it is a trie where all possible nodes are present. There is thus no arc whose label contains more than one letter. A previous result [All00] makes use of some statistical analysis for stating that a suffix tree of a text of length $n$ is expected to be uncompact complete at the $\log_\Sigma(n)$ top levels, where $\Sigma$ is the alphabet of the text. Since we are interested in the DNA alphabet (composed of the four nucleotides $A, C, G, \text{ and } T$), then we have that our suffix tree $T$ is uncompact complete at the top $\log_4(n)$ levels where $n$ is the size of the input sequence $s$.

The function $\log_4(n)$ reaches 10 for $n \approx 10^6$, it is greater than 11 for $n = 10^7$, it is more than 13 for $n = 10^8$, and nearly 15 for $n = 10^9$. These values correspond to reasonable values for the minimal length $k_{\text{min}}$ of the motif, and they are reached for values $n$ of the text size corresponding to quite big data sets. In fact, we remind that, due to the potentially exponential complexity in the number $e$ of allowed mismatches among different occurrences of the same model, and due to the fact that for biological applications $e$ is in general proportional to $k$, larger values of $k$ are hard to be tractable in practice.

In order to answer to the second question (how many visits we save using the maximal extensibility information), we have run a test on some real biological data. This test counts the number of models whose suffixes are lexicographically smaller and for which the value of $\text{Max Ext}$ does not suffice to reach $k_{\text{min}}$. We thus count how many attempts to extend a model we would save with our modifications. We have done these tests for $p = 1$, and for different values of $k = k_{\text{min}} = k_{\text{max}}$, $e$, and $q$. The text is approximately $400Kb$ long and it represents a non coding fragment of the genome sequence of the bacterium *Bacillus subtilis*. Notice that, since we are
considering cases in which $k_{\text{min}}$ and $k_{\text{max}}$ coincide, then the possible improvements such as those mentioned in section 8.3.6 are not counted. The results that follow are therefore in general only a lower bound of the number of visits that can be saved. The table below shows, for different values of $q, e$, and $k$, the number of valid models output by the algorithm (column valid), the number of attempted models (column visited), the number of those among the visited models that would be avoided thanks to the maximum extensibility information (column avoidable), and finally the fraction between avoidable and visited given as a percentage. We remind that the time complexity of the algorithm of Marsan and Sagot is linear with respect to the number of visited models.

<table>
<thead>
<tr>
<th>q</th>
<th>e</th>
<th>k</th>
<th>valid</th>
<th>visited</th>
<th>avoidable</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1</td>
<td>8</td>
<td>4472</td>
<td>55553</td>
<td>795</td>
<td>1.5%</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>6</td>
<td>103</td>
<td>2853</td>
<td>56</td>
<td>2%</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>9</td>
<td>534</td>
<td>73441</td>
<td>14644</td>
<td>20%</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>3265</td>
<td>896</td>
<td>27%</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>10</td>
<td>569</td>
<td>180473</td>
<td>54727</td>
<td>30%</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>7125</td>
<td>2683</td>
<td>38%</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>11</td>
<td>484</td>
<td>419255</td>
<td>168339</td>
<td>40%</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>75577</td>
<td>36203</td>
<td>48%</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>3281</td>
<td>1753</td>
<td>54%</td>
</tr>
</tbody>
</table>

Not surprisingly, the number of saved attempts is considerably bigger when the values of the parameters is such that very few motifs are found. Indeed, in that case the algorithm without the maximum extensibility information would have attempted many hopeless extensions. Notice that for $p = 2$, and even more for $p > 2$, it is reasonable to expect even higher values for the percentage of saved visits.

Given these answers to the first two questions, we claim that the answer to the third question is affirmative. The amount of saved visits and their impact on the performance of motif extraction depends on the instances and the values of the parameters. Some further studies on real data or some possible preprocessing could be helpful in detecting which are the ranges of values of the parameters for which this modification is more suitable.

### 8.4.2 A first implementation: results

In [PS00a], a challenging problem has been launched as the one of finding all single motifs of length 15 with at most 4 mismatches in a 20 texts of size 600. The quorum is 20, that is the motifs have to appear in all sequences. We ran the algorithm of [MS01] on such instance, as well as the new version of the algorithm that includes the improvement suggested in this chapter. In this first implementation, we chose to
store maximum extensibility information if and only if this does not require adding ghost nodes. Therefore, the space cost is the same as in the previous version of the algorithm. Figure 8.2 shows the changes in time costs which results up to 40% smaller than with the original algorithm. We actually believe that a true challenge should involve texts of larger size. Therefore, we have ran tests with the same parameters (length 15 and at most 4 mismatches) on larger input sequences. The results confirm the 40% time costs reduction for sequences of length 700 and 800, 22% for length 900, and then the percentage decreases, but the time required by the improved version of the algorithm always results lower than for the original one. These results are shown in figure 8.2, where we plotted the time (in minutes) required by the algorithm before and after the improvement, with respect to the length of the 20 input sequences.

8.5 Conclusions and further work

Section 8.4.1 has shown how, once the size of the input sequence is known, it can be predicted until which level of the suffix tree \( T \) adding ghost nodes is expected not to add any extra space cost. We have also seen that for many reasonable values of the minimal length required for the motif, this data motivates the belief that storing maximum extensibility information for models up to \( k_{\min} - 1 \) (which is our general choice, and the one for which we have computed the number of saved visits in our tests) leads to little extra space required. Nevertheless, depending on the specific
applications, and in particular on the length \( n \) of the input sequence, it may happen that storing such data for models of higher length can result in no extra space cost (because \( \log_{2} n > k_{\min} \)) and useful (because \( k_{\max} > k_{\min} \)). On the other hand, it may also be that \( n \) and \( k_{\min} \) are such that they do not allow to store the data for models up to \( k_{\min} - 1 \) bases long without excessive space costs. We therefore claim that a preliminary preprocessing that suitably evaluates and compares the values of \( \log_{2} n \) and \( k_{\min} \) should be performed and should drive the choice of which levels of the suffix tree could store maximum extensibility information.

As we have seen, the structured motif extraction is made considering models according to a (virtual) depth-first visit to the model tree. It is intuitive to see that in the case of a breadth-first search visit [SVS95b], the maximal extensibility information could be available for all shorter submodels of the one being extended. This could result in a possible increase of the number of saved visits. It could therefore be interesting to perform the modification presented in this chapter for motif extraction algorithms that consider strings in an order corresponding to a breadth-first search visit of the tree of the models.
8. Improving the Extraction of Structured Motifs
Chapter 9

Conclusions

9.1 Concluding remarks

With the increasing number of complete genome sequences available, a new phase of studies in molecular biology has started. This has led to new challenges in computational biology. Comparative genomics and the detection of common and repeated motifs in biological sequences are among them. The latter can also be meant as a starting point for the former. Within these fields, this thesis aimed to contribute to some of the above mentioned challenges.

Studies of genomic sequences and structures at a large scale, as those described in chapters 3, 4, and 5 (part I), lead to new tools for the molecular biologist with the purpose of both investigating the properties of certain genomes and gene families, and comparing them with respect to those of other genomes. The syntenic distance (investigated in chapter 3) is a way to quantify the distance between two genomes according to a suitable measure. Both the PaTre tool and the investigations on the cardinalities of gene families (presented respectively in chapters 4 and 5) have a twofold use. First, they serve to investigate suitable properties (regarding gene duplication) of gene families belonging to a specific genome, as well as to compare these properties with those of other families of the same genome with the aim of analysing the evolutionary history of the whole genome (and not just one gene family). Second, they can be used for comparative genomics, as the properties of gene families that they detect can be compared with those of corresponding families in other organisms, giving information on the different evolutions of two organisms since the speciation events that separated them.

Motif extraction represents also a tool for comparative genomics, as it can be used to search for common motifs among $N$ different sequences possibly representing different genomes. These common motifs in the genomic sequence are good candidates (to be evaluated by the biologists) to represent biologically significant
similarities in the functions of the fragment of DNA (or amino acids sequence) they involve. Moreover, the detection of common and repeated motifs leads to a possible more abstract view of sequences (that represents them as sequences of labels that in turn represent segments, and no longer as single nucleotides or amino acids), which is the starting point of most of the problems in the genome rearrangement literature. Nevertheless, many applications of motif extraction, besides those in fields different from molecular biology, are also for investigations within the same sequence. For example, structured motif extraction can be used to find regulatory sites, and single motif extraction for finding duplicated genes. This last application represents another meeting point for the contents of part I and part II of this thesis. Our results go in the direction of improving the speed of exact algorithms for motif extraction. Tiling motifs are promising in this sense because they may lead to an algorithm for motif extraction whose time complexity is not exponential with respect to the number of mismatches allowed between the occurrences of a motif. In fact, we believe that the complexity of finding the tiling motifs of a sequence is independent from the number of wild cards allowed. The use of maximum extensibility information that we showed to be promising for many variants of motif extraction, can lead to a better performance of the algorithm of Marsan and Sagot, and thus to the possibility of extending the length of the output motifs. In particular, we expect interesting results with respect to the challenge launched by P.Pevzner [PS00a]. This consists in finding a \( k = 15 \) nucleotides long model with \( e = 4 \) within 20 sequences of length 600. Preliminaries tests on randomly generated texts of this size promise a 20\% of the visited models to be saved using maximum extensibility information. The input has a total size of 12000, and we have that \( \log_2(12000) = 6.7 \), which implies that for models up to length 6, no ghost nodes and paths should be created.

### 9.2 Future work

Referring to the problems addressed in this thesis, further intensive work may be envisioned. We consider each subject separately.

#### 9.2.1 Syntenic distance

We have shown in chapter 3, that finding the optimal pairing for multicomponent synteny is \( \mathcal{NP} – \text{hard} \). It could be interesting to conceive a heuristic that approximates this problem which we have shown to be equivalent to solving the optimal pairing problem defined in section 3.4.3. Given a multicomponent instance of synteny, we do not know how to find the optimal canonical move sequence for solving the single components because connected synteny is also \( \mathcal{NP} \)-hard ([IDK+97]). We therefore do not know a priori how many fusions have to be done overall in a component, nor we know how many fissions are required at the end of the hypothetical canonical move sequence of a single component. Hence, it seems impossible to an-
ticipate the inter-component moves to be done, and thus a greedy approach could be suitable.

9.2.2 PaTre: a tool for building paralogy trees

We plan to investigate further, by means of both simulations and experiments on real data, the possibility of guessing which arcs of the output tree should contain a steiner node in order to give a result more faithful to the truth. This could be done by looking at the cost of arcs.

As a further step in this project, we aim to compare paralogy trees of different families, both in the same genome and in different ones. In particular, as a new way of performing genome comparisons and phylogenetic reconstruction, it could be interesting to compare the shape (i.e., some topological properties) of the paralogy tree of the same families in different organisms. For example, we could check whether certain organisms tend to have deeper trees, with respect to others, for the same family. With respect to traditional phylogenetic studies, instead of comparing single genes belonging to different organisms, we could compare entire paralogy trees for a given family of paralogous genes in different species. This approach could avoid many problems related to the choice of the gene and the relative cross-similarity between orthologs and paralogs. Moreover, it could give information about the strategy of new genes creation. It could also address the question posed by the simulator structure, that is, whether the template for a new copy should be chosen among all the existing paralogs with uniform probability, or, on the contrary, the more recent genes have a higher probability to be chosen. Finally, within the same genome, it could be interesting to compare the shapes of different trees related to different families. This could help in current studies of the different duplication rates of genes.

9.2.3 Clustering gene families by using random graphs

The main goal in this area is to improve the random graph model, that is to find a better one. It seems that a somehow high local connectivity is still missing in the model $\mathcal{G}(n, m, \alpha + 1)$. To this purpose, some recent work caught our attention [Kle00, KKR+99, Kle99]. Some studies performed mainly by J. Kleinberg at Cornell University attempted to characterize the graph that represents the web. Nodes are sites and edges are links. It has been observed that lengths of minimal paths connecting nodes are surprising low. It seems also [Mar01] that gene duplication in evolution could present some analogies with the way new web sites are created and modified. These similarities could be worth further investigations.

Another possible improvement concerns the different rates of duplications in different families of genes. It is documented ([Coi96]) that some genes duplicate more
often that others. In our model, this could be represented by a larger probability of the presence of certain edges: genes of the same and quickly duplicating family should correspond, in the random graphs, to nodes that have a larger probability of being connected by an edge.

9.2.4 Tiling motifs

A modified version of the notion of basis can be conceived. This could result in a possibly smaller size of $B$, and in a possibly computationally easier problem to find them. In particular, the notion of redundancy could be strengthened in order to avoid mentioning motifs that are repeated because of periods present in one of them. In this case, the current definition of irredundancy is not enough to avoid an over representation of such motifs. Another possible way of further decreasing the size of the basis and, hopefully, the complexity of detecting it, is to bound the number of non consecutive wild cards in a motif. This would result in a smaller set of motifs in the base, and motifs with more wild cards could still be built by concatenation of these.

An important extension of the notion of tiling motifs is to allow different lengths for the occurrences of the same motifs, that is to allow insertions and gaps. The notion of flexible motifs [PRP01] goes in this direction and deserves attention, in case the problem of finding such motifs results computationally feasible.

9.2.5 Improving structured motif extraction

In section 8.4.1 it is shown how, once the size of the input sequence is known, it can be predicted until which level of the suffix tree $T$ adding ghost nodes is expected not to add any extra space cost. We have also seen that for many reasonable values of the minimal length required for the motif, this information motivates the belief that storing maximum extensibility information for models up to $k_{\text{min}} - 1$ (which is our general choice, and the one for which we have computed the number of saved visits in our tests) leads to little extra space required. Nevertheless, depending on the specific applications, and in particular on the length $n$ of the input sequence, it may happen that storing such information for models of higher length can result in no extra space cost and be useful. On the other hand, we have also seen that $n$ and the size required to the motif are such that they do not allow to store the maximum extensibility information for all valid models of required length without excessive space costs. We therefore claim that a preliminary preprocessing should suitably evaluate up to which length of the motifs the maximum extensibility information should be stored.

As we have seen, the structured motif extraction is made considering models according to a (virtual) depth-first visit to the model tree. It is intuitive to see
that in the case of a breadth-first search visit [SVS95b], the maximal extensibility information could be available for all shorter submodels of the one being extended. This could result in a possible increase of the number of saved visits. It could therefore be interesting to perform the modification presented in this chapter for motif extraction algorithms that consider strings in an order corresponding to a breadth-first search visit of the tree of the models.
Bibliography


Index

amino acid, 7, 8, 99
*Arabidopsis thaliana*, 59–63

*Bacillus subtilis*, 18, 93, 132
distance
  best fit, 94
  block interchange, 17
  breakpoint, 18, 23
  exemplar, 18
  Levenshtein, 68
  metric, 17, 25, 77
  reversal, 16
  segment-based, 13
  signed reversal, 16
  syntenic, 15, 18, 20, 23–28, 30–48
  transformation, 17, 50–53, 98
  translocation, 17
  transposition, 16
  ultrametric, 77, 78

*Escherichia coli*, 18
eukaryotes, 9, 13, 67, 99
gene
  duplication, 9, 17, 18, 20, 49, 50, 55, 56, 58–61, 65, 66
  recent, 67
  families, 49, 50, 56, 58–63, 65, 67
  homologous, 9
  orthologous, 9, 16
  paralogous, 9, 19, 20, 49, 50, 56, 58–63, 65–67
  pseudo, 19, 67
  syntenic, 24

genetic code, 7
genome rearrangement, 6, 13–16, 23
lightest spanning arborescence, 53

motifs
  extraction, 1, 97–99, 101–123, 125, 128, 129, 132–135, 140
  structured, 99, 102, 119–123, 125, 128, 129, 132–135
  tiling, 101, 103–118, 140

*Mycoplasma*, 83
  *genitalium*, 18, 21, 66, 70, 71, 83, 85, 89–93
  *pneumoniae*, 18, 58, 59, 70, 83, 93
nucleotide, 7, 99
parsimony principle, 16
*PaTre*, 20, 49–63, 139
phylogeny reconstruction, 6, 14, 17, 19, 62, 139
prokaryotes, 8
protein folding, 8
random graph, 65, 72–74, 76, 78–83, 88
  model, 65, 74, 76, 78–83, 88
  \( G(n, m) \), 74
  \( G(n, m, f) \), 76, 78–80
  \( G(n, m, \alpha + 1) \), 81–83, 88, 93
RNA, 8
  messenger, 8, 9, 99
  ribosomal, 8
  transfer, 7, 8

*Saccharomyces cerevisiae*, 58
syntenic diameter, 27, 28, 30–32
  bidimensional, 31, 32
synteny
  connected, 26
  exact, 26
  general, 28, 32, 46–48
  instance of, 25
  linear, 26
  linear exact, 27
  multi-component, 32–36
  nested, 26, 36–45
    exact, 37–40
    general, 41–45
    linear, 41

tree
  consensus, 19
  gene, 19, 49
  paralogy, 49, 50, 53–63, 139
  species, 19
  suffix, 99, 100, 104, 111, 112, 114,
    115, 121–123, 125, 128, 129,
    132
    compact, 100, 125, 132, 133
    generalized, 100, 120