UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Efficient Algorithms for Structured Motifs Extraction in DNA Sequences

Alexandra Sofia Martins de Carvalho

(Licenciada)

Dissertação para Obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Orientador:	Doutor Arlindo Manuel Limede de Oliveira
Co-orientador:	Doutora Ana Teresa Correia de Freitas

Júri

Presidente:	Doutora Maria Cristina Sales Viana Serôdio Sernadas
Vogais:	Doutor Arlindo Manuel Limede de Oliveira
	Doutora Marie-France Sagot
	Doutora Ana Teresa Correia de Freitas

Junho 2004

UNIVERSIDADE TÉCNICA DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

Efficient Algorithms for Structured Motifs Extraction in DNA Sequences

Alexandra Sofia Martins de Carvalho

(Licenciada)

Dissertação para Obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Orientador:	Doutor Arlindo Manuel Limede de Oliveira
Co-orientador:	Doutora Ana Teresa Correia de Freitas

Júri

Presidente:	Doutora Maria Cristina Sales Viana Serôdio Sernadas
Vogais:	Doutor Arlindo Manuel Limede de Oliveira
	Doutora Marie-France Sagot
	Doutora Ana Teresa Correia de Freitas

Junho 2004

Abstract

In this thesis we propose a new algorithm for the extraction of repeated motifs that may represent binding-site consensi in genomic sequences. In particular, the algorithm extracts structured motifs, which are defined as a collection of highly conserved segments of DNA with pre-specified sizes and spacings between them. This type of motifs is highly relevant in the search for gene regulatory mechanisms since promoter regions can be effectively modeled by structured motifs.

The new algorithm uses factor trees, a variation of suffix trees, and a new data structure, called box-links, to store the information about conserved regions that repeat often in the dataset sequences. The time and space complexity, in the worst case analyses, shows a gain over previous algorithms that is exponential on the spacings between the segments of the structured motifs. The application of a prototype implementation of this algorithm to biologically relevant datasets shows the ability of the method to extract relevant consensi. The experimental results also show that this algorithm is much faster than existing ones, sometimes by more than two orders of magnitude.

To deal with the enormous amount of data coming from this large-scale genome sequencing era, a parallel method is also proposed for the efficient extraction of structured motifs. By partitioning the structured motif searching space we divide the most demanding part of the algorithm by a number of processors that can be loosely coupled. In this way we obtain, under conditions that are easily met, a speedup that is linear on the number of available processing units. This speedup is verified by both theoretical and experimental analyses.

Keywords: Promoter prediction, Suffix tree, Factor tree, Structured motif, Box-link, Parallel algorithm

Resumo

Nesta tese é proposto um novo algoritmo para a extracção de motivos frequentes em sequências de ADN. Em particular, são extraídos motivos estruturados, definidos como uma colecção de segmentos de ADN fortemente conservados, onde o tamanho e a distância consecutiva entre os segmentos são parâmetros especificados à priori. Os motivos estruturados assumem um papel de extrema importância na compreensão dos mecanismos de regulação genética pois modelam eficazmente as regiões promotoras das sequências genómicas.

O novo algoritmo utiliza árvores de factores, uma variação das árvores de sufixos, e uma nova estrutura de dados, denominada ligações entre caixas, que armazena a informação sobre regiões fortemente conservadas nas sequências de ADN. A análise da complexidade temporal e espacial, no pior caso, apresenta um ganho exponencial na distância entre os segmentos dos motivos estruturados, relativamente às abordagens anteriores para a mesma extracção. A aplicação do protótipo desenvolvido a dados com significado biológico mostra a capacidade do método para extrair consensos relevantes. Os resultados experimentais mostram ainda que o algoritmo é muito mais rápido que os existentes, às vezes por mais de duas ordens de magnitude.

Para fazer frente ao enorme número de dados provenientes da sequenciação genómica em grande escala, é ainda proposto um método paralelo para a extracção de motivos estruturados. Através de uma correcta partição do espaço de pesquisa dos motivos o trabalho de extracção é dividido eficientemente pelas unidades de processamento disponíveis. Desta forma é obtida, sob condições facilmente atingíveis, uma aceleração linear no número de unidades de processamento. Esta aceleração é verificada por análise teórica e prática.

Palavras chave: Predição de promotor, Árvore de sufixos, Árvore de factores, Motivo estruturado, Ligação entre caixas, Algoritmo paralelo

Resumo alargado

Em termos moleculares, um *gene* pode ser definido como sendo um segmento de ADN, ou seja, uma sequência sobre o alfabeto de quatro *bases* (A, C, G, e T). Os genes codificam a estrutura das proteínas, sendo estas responsáveis por dirigir o metabolismo celular através da sua actividade como enzimas. O dogma central da biologia molecular assume um percurso para o fluxo de informação genética: ADN $\rightarrow$ ARN $\rightarrow$ proteína. Segundo este princípio, as moléculas de ARN são sintetizadas a partir de segmentos de ADN, um processo denominado *transcrição*, e as proteínas são sintetizadas a partir de moléculas de ARN, um processo denominado *tradução*. As moléculas de ARN surgem assim como intermediárias que transportam a informação contida no ADN para os locais de síntese das proteínas.

O constante avanço nas tecnologias de sequenciação de ADN permitiu identificar e guardar a sequência de bases que representa o conteúdo genético – *genoma* – de diversos organismos. A interpretação da informação codificada nestes genomas é uma das preocupações proeminentes na Bioinformática, sendo a compreensão da regulação genética um importante esforço nesse sentido. Uma parte importante desta regulação é mediada por proteínas específicas – *factores de transcrição* – que se ligam a determinados segmentos distribuídos sobre as próprias sequências de ADN – *locais de ligação* ou *região promotora*. Dependendo da complexidade do organismo em causa, estes locais de ligação podem ser constituídos por um ou mais segmentos de ADN, aos quais passaremos a chamar *caixas*. Neste ponto, torna-se óbvio que a compreensão da regulação genética passa primeiramente pela identificação de regiões promotoras, necessitando esta de um modelo que descreva a organização do promotor dadas as suas principais funções [Wer99].

Até há cinco anos, todos os métodos para detectar regiões promotoras em sequências de ADN analisavam cada caixa individualmente. Extraíam assim os chamados *motivos simples*, motivos compostos por uma única caixa, baseando-se na procura de padrões a menos de algumas substituições [BJVU98, Tom99, vHACV98] e mutações [Sag98]. Até então apenas um método muito conhecido, o algoritmo MEME [BE95b], lidava com *motivos múltiplos* através da identificação de conjuntos de motivos simples compatíveis (constrói iterativamente motivos múltiplos a partir de motivos simples cuja posição de ocorrência nas sequências de ADN não contradiz as posições de ocorrência dos motivos simples). Contudo, estas premissas são no mínimo questionáveis, dado que o conjunto de motivos produzido satisfaz apenas compatibilidade, não sendo aplicado qualquer constrangimento nas distâncias que os separam.

As primeiras soluções tendentes a resolver a extracção de motivos múltiplos consideravam os mesmos compostos apenas por duas caixas, separadas por uma distância tipicamente fixa. Um dos primeiros métodos propostos neste sentido é um método heurístico [CS92], tendo sido seguido por outras abordagens baseadas na procura de padrões [MS00, VMS99, VMLS00]. Para reflectir que a região promotora pode estar fragmentada em diversos locais de ligação Marsan e Sagot [MS00] introduziram o conceito de motivos múltiplos com várias caixas, denominados por *motivos estruturados*. Um motivo estruturado pode ser descrito como uma colecção ordenada de caixas, um número máximo de substituições para cada caixa, e um intervalo de distância para cada par consecutivo de caixas.

Recentemente, os métodos propostos [vHRCV00, EP02, EKGP03] continuam a considerar motivos complexos de uma forma muito limitada, mais especificamente, com duas caixas apenas. Além disso, são ainda muito ingénuos, pois enumeram exaustivamente todos os motivos possíveis ou extraem-nos através do cruzamento das listas de ocorrências de motivos simples. No primeiro caso, os métodos são em geral muito limitados no tamanho dos motivos que conseguem encontrar, assim como na distância entre as caixas que usualmente é fixa. No segundo caso, encontrar motivos múltiplos através do cruzamento de posições de motivos simples demora tempo pelo menos quadrático no número dos motivos simples e correspondentes ocorrências. Contudo, um motivo com várias caixas pode ser estatisticamente significativo ainda que as suas caixas individualmente não o sejam. Na realidade, esta é uma das razões que justifica a extracção de motivos múltiplos como um todo, em oposição à síntese de modelos múltiplos a partir de modelos simples.

Nesta tese, a extracção de motivos, fortemente influenciada pelos trabalhos realizados por Marsan e Sagot [MS00] para motivos estruturados, é abordada sob duas vertentes. Numa primeira vertente, um novo algoritmo exacto é proposto para a extracção de motivos estruturados. É mostrado através de análise de complexidade temporal e espacial, no pior caso, que o algoritmo apresenta um ganho exponencial, relativamente às abordagens anteriores para a mesma extracção. Para concretizar tal ganho, é proposta um nova estrutura de dados, denominada *ligações entre caixas*. Os resultados experimentais obtidos confirmam que o novo algoritmo é muito mais rápido que os anteriores, por vezes, em mais de duas ordens de magnitude. Numa segunda vertente, uma nova técnica de paralelização é proposta para os algoritmos que extraem motivos estruturados. Esta técnica introduz um novo problema NP-completo no sentido forte, denominado PARTIÇÃO A MENOS DE ε. Mostra-se que o algoritmo paralelo para extracção de motivos estruturados é de trabalho eficiente, no que diz respeito à mesma versão sequencial, sob condições facilmente atingíveis. São ainda obtidos resultados experimentais que confirmam os resultados teóricos de divisão linear do trabalho pelos número de unidades de processamento.

Para concluir, a principal contribuição desta tese é o desenvolvimento de novos métodos eficientes para a descoberta de regiões promotoras em sequências de ADN, dando assim um passo inicial na compreensão do processo de regulação genética. O impacto deste esforço pode ser enorme. As regiões promotoras podem desempenhar um papel fulcral na descoberta da função de determinados genes e ainda dar pistas na descoberta da função de proteínas completamente desconhecidas. A predição da funcionalidade de determinados promotores pode ainda levar a indicadores iniciais para terapias genéticas, enquanto que a análise da combinatória dos seus elementos é essencial para a compreensão do desenvolvimento celular.

Acknowlegements

I would like to thank my supervisors, Arlindo Oliveira and Ana Teresa Freitas, for their guidance during the elaboration of this thesis. Their constant support during my short scientific research life was essential for my development as a scientist. They also gave me the opportunity to meet several scientists, specially, Marie-France Sagot, who became a reference to me and to the work of this thesis.

Furthermore, I would like to thank all my colleagues at the ALGOS Group, having a special thank to João Silva for proofreading this thesis. I am profoundly grateful to the excellent working environment and access to an outstanding computer that I have in this Group that houses me at INESC-ID.

Finally, I would like to give special thanks to my family and friends, specially, João, Sara and Soledade, for all support and patience. But I owe more to Paulo than to anyone else. He has been a pillar of support I could rely on all times. He inspired me, encouraged me, and sustained me. Most of all, he has always believed in me.

The Project BIOGRID POSI/SRI/47778/2002 has sponsored the work contained in this dissertation, as well as my participation in conferences and workshops.

Contents

1 Introduction **1**

1.1 Context . . . 1

1.2 Aims . . . 1

1.3 Claim of contributions . . . 2

1.4 Layout of the thesis . . . 2

2 Extraction of motifs **5**

2.1 Problem description . . . 5

2.2 Main approaches . . . 7

3 Extraction of structured motifs **11**

3.1 Basic data structures . . . 11

3.1.1 Suffix trees . . . 11

3.2 Single motif extraction algorithm . . . 14

3.3 Structured motif extraction algorithms . . . 17

3.3.1 Jumping in the suffix tree . . . 19

3.3.2 Jumping in the suffix tree using suffix-links . . . 23

3.3.3 Extending the algorithms . . . 32

3.3.4 Measuring statistical significance . . . 32

4 Extraction of structured motifs using box-links **33**

4.1 Basic data structures . . . 33

4.1.1 Factor trees . . . 33

4.2 Structured motif extraction algorithm . . . 35

4.2.1 Box-link data structure 36
4.2.2 Box-link construction 37
4.2.3 Jumping in the factor tree using box-links 41
4.2.4 Extending the algorithm 44
4.2.5 Measuring statistical significance 45
4.3 Experimental results 45

5 Parallel extraction of structured motifs 49

5.1 Balanced partition 50
5.2 Parallel structured motif extraction algorithm 55
5.2.1 Tree partition 55
5.2.2 Parallel extraction 59
5.3 Experimental results 62

6 Conclusions 67

Notation

(m, d)	structured model
(v, e_v)	node-occurrence
B_l	box-link
c	number of gold bars obtained when a given gold bar is cut
$Colors$	Boolean array of size N indicating occurrence in the input sequences
d	distance between two consecutive boxes of a structured model
δ	number of virtual gold bars attributed to each person
Δ	range allowed in the distance between two consecutive boxes of a structured model
δ_i	allowed interval around the distances $[d_{min_i}, d_{max_i}]$ of a structured model
d_{max}	maximum distance between two consecutive boxes of a structured model
d_{max_i}	maximum distance between the i-th and $(i+1)$-th boxes of a structured model
d_{min}	minimum distance between two consecutive boxes of a structured model
d_{min_i}	minimum distance between the i-th and $(i+1)$-th boxes of a structured model
e	maximum number of substitutions allowed
e_{global}	maximum number of substitutions allowed in the whole structured model
e_i	maximum number of substitutions allowed in the i-th box of a structured model

ε	gold bar weight overload allowed for each person
γ	number of persons with at most ε more gold
Γ	range allowed in the size of the boxes of a structured model
I'_i	i-th interval of virtual gold bars
I_i	i-th partition set
k	length of a single model or a box of a structured model
k_i	size of the i-th box of a structured model
ℓ	number of initial gold bars
$L(i)$	array that stores the state of the nodes at level k for the $(i-1)$-th box of a structured model
λ	empty sequence
$list_{leaf}$	list of leaves as they are inserted in the factor tree
L_k	set of leaves at depth k
$Lptr_{v_k}(i)$	list of pointers to nodes at $[ik + (i-1)d_{min}, ik + (i-1)d_{max}]$ levels
$\mathcal{M}$	lexicographic tree
m	model
m_i	i-th box of a structured model
N	number of input sequences
n	average length of a string or input sequence S
n_k	number of tree nodes at depth k
$\nu(e,k)$	number of distinct words that are at a Hamming distance at most e from a k-long word
p	number of boxes in a structured model

q	quorum, minimum number of occurrences in the input sequences
r	number of persons
R	suffix or factor tree root
S	string or input sequence
S_i	i-th input sequence
Σ	alphabet of the string or input sequence S
S_l	suffix-link
t	number of times each gold bar is successively cut
$\mathcal{T}$	suffix or factor tree
V_j	j-th interval of virtual gold bars
w	total weight of gold bars
w_j	weight of the j-th gold bar
z	minimum number of final gold bars to share

List of Algorithms

3.1 SpellMotifs, single motif extraction 18
3.2 ExtractMotifs, structured motif extraction jumping in the suffix tree 20
3.3 ExtractMotifs, structured motif extraction using suffix-links 28

4.1 BoxLink, box-link construction 40
4.2 ExtractMotifs, structured motif extraction using box-links 42

5.1 SimpleCut, gold bar share among persons 52
5.2 SimpleCut, work balance among available processing units 56
5.3 PExtractMotifs, parallel version of Algorithm 3.2 60
5.4 PSMILE, parallel SMILEv1.45 algorithm (with frequencies estimation) 60
5.5 PSMILE, parallel SMILEv1.45 algorithm (without frequencies estimation) . . 61

List of Figures

3.1 Suffix tree for string AGACAGGAGGC\$. 12
3.2 Generalized suffix tree for strings TACTA\$ and CACTCA\$. 13
3.3 Generalized suffix tree with *Colors* for strings TACTA\$ and CACTCA\$. . . . 13
3.4 Suffix tree with suffix-links for string AGACAGGAGGC\$. 14
3.5 Extracting structured motifs jumping down in the tree. 20
3.6 Extracting structured motifs following suffix-links. 24
3.7 The $Lptr_{v_k}(2)$ arrays for the extraction of a third box. 25
3.8 Extracting a third box following suffix-links and $Lptr_{v_k}(2)$ array. 26

4.1 At most 5-deep factor tree for string AGACAGGAGGC\$. 34
4.2 At most 3-deep factor tree for string AGACAGGAGGC\$. 34
4.3 A general idea of box-links. 36
4.4 Extracting structured motifs following b) suffix-links and c) box-links. 37
4.5 Brute force construction of box-links starting at the same position. 38
4.6 Merge of equivalent box-links from the third box on. 38
4.7 Equivalent merge construction of box-links starting at the same position. . . 39
4.8 Box-links for boxes with a fixed size at variable distances. 40

5.1 Lexicographic trie cut at depth $t+1$=2. 57
5.2 Intervals V_j in the virtual tree space. 57
5.3 Intervals I'_i in the virtual tree space. 58
5.4 Intervals I_i in the tree node space. 58
5.5 Tree partition up to ε. 59
5.6 Estimation of the linear coefficient for work-efficiency. 64
5.7 PSMILE results with 3 processing units. 65

List of Tables

4.1 Extraction of $CGGn_{11}CCG$ and $CGGAn_{9}TCCG$ from the first dataset. . . . 46
4.2 Extraction of $TTGACAn_{17}TATAAT$ from the *E. coli* dataset. 47
4.3 Extraction of structured motifs with three boxes from synthetic data. 47

5.1 Extraction of structured motifs by PSMILE algorithm. 63
5.2 Extraction of structured motifs by SMILEv1.45 and PSMILE algorithms. . . 64

Chapter 1

Introduction

1.1 Context

The core of the work presented in this thesis was developed at the ALGorithms for Simulation and Optimization Group (ALGOS Group) of INESC-ID, Lisboa. The Bioinformatics effort in ALGOS Group aims at several goals. One of these goals is addressed in this thesis: the development of efficient algorithms for cis-regulatory region identification.

The techniques used throughout this thesis rely and are greatly influenced by previous work done by Marie-France Sagot, at the Helix research group of INRIA Rhône-Alpes, jointly with two of her PhD students, Laurent Marsan (in fact, an ex-student) and Julien Allali. There were other approaches, which will be briefly presented in the next chapter, that also influenced the work of this thesis.

This work was partially supported by the Project BIOGRID POSI/SRI/47778/2002.

1.2 Aims

In this large-scale genome sequencing era, the main bottleneck to progress in molecular biology is data analysis. The prime objective of this thesis is the search for one kind of biological information in sequenced data: the extraction of structured motifs from DNA sequences.

The main goal of this thesis is the proposal of new efficient algorithms, for the extraction of structured motifs, capable of dealing with the enormous amount of data coming from the Bioinformatics community. Two algorithms for structured motif extraction were first proposed by Marsan and Sagot [MS00]. This work aims at improving such algorithms in terms of their computational complexity. The improvements obtained should be documented by complexity analysis, as well as by experimental results over synthetic and biologically significant datasets.

1.3 Claim of contributions

Two main contributions were achieved within the scope of this thesis:

- A new algorithm for structured motif extraction is introduced [CFOS04a]. It is shown that the new algorithm has an exponential time and space gain, in the worst case analyses, relatively to previous existing approaches [MS00]. To achieve such gain a new data structure is proposed, called *box-links*. Experimental analysis shows that the new algorithm is much faster than previous ones, sometimes by more than two orders of magnitude.

- A parallel version of an algorithm for the extraction of structured motifs is proposed [CFOS04b]. The presented solution is a general method to parallelize algorithms where the search space is represented by a tree, which is the case of algorithms for structured motif extraction. The parallelization is based on the analysis of a new interesting strongly NP-complete problem, the PARTITION UP TO ε problem. The parallel algorithm is shown to be work-efficient, with respect to the sequential one, under easily achievable conditions. This speedup is also verified by experimental analysis.

1.4 Layout of the thesis

In Chapter 2, we give some background and context to this thesis. In Section 2.1, we start by describing the problem of motif extraction and next, in Section 2.2, we shortly survey the main approaches to the problem.

In Chapter 3, we present in detail previous work that provides the foundations for this thesis. In Section 3.1, we start by presenting a basic data structure for string processing problems, called suffix tree, which is going to be used throughout this chapter. Then, in Section 3.2, we present an algorithm for single motif extraction, proposed by Sagot. Next, in

Section 3.3, we present the problem of structured motif extraction. Two algorithms, devised by Marsan and Sagot, are presented and analyzed in Section 3.3.1 and Section 3.3.2. Furthermore, in Section 3.3.3, we give some insight on how to extend the algorithms, and finally, in Section 3.3.4, we briefly explain how Marsan and Sagot dealt with the statistical significance of the extracted structured motifs.

In Chapter 4 we present the main contribution of this thesis, which is a new algorithm and a new data structure for the extraction of structured motifs. In Section 4.1, we present factor trees, proposed by Allali and Sagot, which appeared as a more compact version of suffix trees that indexes only the sufficient data required by problems such as the extraction of structured motifs. Next, in Section 4.2, we introduce the new approach for structured motif extraction. First, in Section 4.2.1, we present the new data structure, called box-links, and give some insight of its usage. Second, an efficient algorithm to build the box-link data structure is proposed in Section 4.2.2. Third, the new algorithm to extract structured motifs using box-links is presented and analyzed in Section 4.2.3. Finally, we briefly explain how to extend the algorithm and how to compute the statistical significance of the extracted motifs, in Section 4.2.4 and Section 4.2.5, respectively. We end this chapter with Section 4.3 by providing some experimental results and comparing the three algorithms for structured motif extraction.

In Chapter 5 we present a novel parallelization approach for the extraction of structured motifs, the second major contribution of this thesis. In Section 5.1, we start by presenting a general method to compute a balanced partition of gold bars among persons, introducing a new NP-complete problem, the PARTITION UP TO ε problem. Next, in Section 5.2, we introduce the parallelization of the structured motif extraction based on the PARTITION UP TO ε problem. First, in Section 5.2.1, we show how to distribute the extraction search space using the PARTITION UP TO ε problem. Second, in Section 5.2.2, we present and analyze the new algorithm for parallel extraction of structured motifs. We conclude this chapter with Section 5.3 where we present some experimental results.

In Chapter 6 we present conclusions and future work.

Chapter 2

Extraction of motifs

2.1 Problem description

The identification of DNA as the genetic material revealed that genetic information is represented by a sequence of four *bases* (A, C, G and T), also known as *nucleotides*. In molecular terms, a *gene* can be defined as a segment of DNA. Genes act by coding the structure of proteins, which are responsible for directing cell metabolism through their activity as enzymes. The central dogma of molecular biology assumes a pathway for the flow of genetic information: DNA $\rightarrow$ RNA $\rightarrow$ protein. According to this principle, RNA molecules are synthesized from DNA templates, a process called *transcription*, and proteins are synthesized from RNA templates, a process called *translation*. RNA appears therefore as an intermediate to convey information from DNA to the places of protein synthesis.

The complete genetic content, called *genome*, of most *eukaryotes* (cell or organism provided with a distinct nuclear envelop) is larger and more complex than the genetic content of *prokaryotes* (cell or organism that lack a nuclear envelope, also called *bacteria*). In fact, the genome of most eukaryotic organisms contains not only functional genes, but also large amounts of DNA sequences that do not code for proteins. Some of these non-coding DNA sequences lie between genes, in the so-called *intergenic regions*. However, large amounts of non-coding DNA are also found within the genes. Actually, genes of eukaryotic organisms are composed of segments of coding sequences, called *exons*, separated by segments of non-coding sequences, called *introns*.

An important part of gene regulation is mediated by specific proteins, called the *tran-*

scription factors, which influence the transcription of a particular gene by binding to specific sites on DNA sequences, called *transcription factor binding sites* or simply *binding sites*. Such binding sites are relatively short stretches of DNA, normally 5 to 25 nucleotides long, and are located in the so-called *promoter regions*, also known as *cis-regulatory regions*. Most of these regions are located in the non-coding sequences upstream of genes, but some are also found downstream, and even within the non-coding parts of a gene, the introns. In prokaryotic organisms, the binding sites are located predominantly in the immediate vicinity of the gene, which usually extends about 300 to 600 nucleotides upstream of the transcription start site. However, in eukaryotic organisms the binding site sequences are often shorter, and can be quite variable and distributed over very large distances. There is no clear-cut defined boundary for promoter regions which may extend further upstream to more than 2000 bases, as observed in some sea urchin promoters [KYD96].

Promoter prediction necessarily needs a model of promoter organization and its conspicuous features. In fact, strong and weak points of promoter prediction methods are determined to a large extent by the accuracy of the underlying promoter model with respect to the biological organization. A possible way to describe a promoter views it as being composed of three regions with different functions, each one having one or more transcription factor binding sites. The first one, the *core promoter*, is the region that suffices to determine the precise transcription start site. The second one, the *proximal promoter*, is the region that is capable of initiating basal transcription. Finally, the *distal promoter*, also called *enhancer*, is the transcription regulatory region that can be located farther from the core promoter and has for main function stimulating transcription. A detailed explanation on possible models for prediction and recognition of eukaryotic promoters can be found in the literature [Wer99].

The DNA sites involved in promoter function can be identified by searching for well conserved regions in a set of non-coding DNA sequences. Such well conserved regions, also known as consensus regions, are called *motifs* and can be found by comparison of non-coding sequences of a given organism, or by comparison of non-coding sequences of related genes in different organisms. In the first approach, frequently occurring patterns are likely to correspond to binding sites of a common transcription factor. The second approach is called *phylogenetic footprinting* [DB97] and requires careful identification of the appropriate genes to use. Only non-coding sequences of *orthologous genes*, which are genes evolutionarily related that perform the same biological function, are appropriate for phylogenetic footprinting. This technique uses the functional/non-functional dichotomy to identify regulatory elements by finding unusually well conserved regions in a set of orthologous non-coding DNA sequences from multiple species (for example, the non-coding sequence upstream the insulin gene in different species of vertebrates). Functional sequences tend to evolve much slower than non-functional sequences, as they are subject to selective pressure. Hence, it is a good conjecture that unusually well conserved regions in such sequences have some regulatory function.

There are two central problems concerning motifs in sequences: localization and extraction [CS04]. The goal of the *motif localization* problem is to find the positions in a sequence of the occurrences of a given motif [PVMZ04]. The task addressed in this thesis is the *motif extraction* problem and aims to identify *de novo* binding site consensi from a set of non-coding DNA sequences. The consensus extraction has been addressed in a variety of ways. Recent approaches have confronted the problem of extracting binding site consensi [BJVU98, Tom99, vHACV98, Sag98] or considered the fact that binding sites often come together in a well ordered and regularly spaced manner [CS92, MS00, VMS99, VMLS00, FMFM95, vHRCV00, BE95b, EP02, EKGP03, KFQW99]. Despite the existence of several proposals in the literature, the problem of detecting regulatory sites in DNA sequences is far from being solved. Given the flexibility of regulatory mechanisms, it remains essential to develop computer-assisted promoter recognition methods capable of detecting different kinds of regulatory signals and adapting to different promoter models. The impact of this task in the Bioinformatics community is enormous. Promoter regions can play an important role in gene function and may offer some clues to the function of completely anonymous proteins. Prediction of the functionality of a promoter may also yield initial indications for gene therapy approaches, while analysis of the combinatorics of their elements is essential for understanding cell development.

2.2 Main approaches

Identifying promoter sequences is notoriously difficult for both prokaryotic and eukaryotic organisms. There are two major limitations in this task. First, there is a constraint of algorithmic nature, meaning that in general the proposed methods can only be applied to sets of sequences restricted in their length and number. Second, there is a weakness in the models employed for promoter regions, leading to poor promoter predicting methods. Nevertheless, the subject has gained a renewed interest in the last few years, with the sequencing of the genome of vertebrates such as man and mouse. The literature on the topic of DNA binding site sequence detection is extensive, and there are two surveys on the subject [BJEG98, VMS99]. Herein, we concentrate on briefly surveying methods that try to extract conserved single binding sites, or multiple ones, possibly located at constrained distances from one another in a set of sequences which potentially contain a cis-regulatory region.

Up to five years ago, all methods for detecting DNA binding sites considered each such site individually. These methods therefore looked for *single motifs*, that is, motifs composed of a unique binding site. This includes pattern-based approaches which allowed for wildcards or a limited number of spacers but not for mutations [BJVU98, Tom99, vHACV98]. Apart from these, only an approach by Sagot [Sag98] based on a suffix tree allowed for mutations. Another well known single motif detection method is the MEME algorithm [BE94, BE95a, BE95b], an Expectation Maximization (EM) based algorithm that identifies motifs with high relative entropy. MEME also deals with *multiple motifs* by identifying significant sets of compatible motifs. It works by iteratively building such multiple motifs from single ones with high relative entropy, whose occurrence positions do not contradict the occurrence positions of the single motifs identified in a previous step. Furthermore, the set of motifs produced must only satisfy compatibility. No constraint, and therefore no statistical value, is put on the distances separating them.

At that time there were few exceptions to the single motif model. An exception was a heuristic approach by Cardon and Stormo [CS92] which looked for motifs composed of two parts separated by a distance which was estimated by the algorithm. Like MEME, the method is based on an EM approach to identify sets of words with high relative entropy. Moreover, there were pattern-based approaches to motif detection, where the pattern may be degenerated, that is, written on a physico-chemical alphabet, and mutations are allowed [MS00, VMS99, VMLS00]. To reflect the fact that a promoter is fragmented in several binding sites Marsan and Sagot [MS00] introduced the concept of *structured motifs*. A structured motif is described as an ordered collection of *boxes*, a maximum number of substitutions allowed for each box, and an interval of distance for each pair of consecutive boxes. Finally, there was a Fasta-inspired method which allows for spacers in general [FMFM95], seeking for exact short motifs occurring in conserved order along different sequences.

More recently, other methods have appeared which try to address the combinatorics of promoter regions. A number of algorithms have been developed that detect motifs composed of two parts, which we henceforward call *boxes*, separated by a spacer, often of fixed length [vHRCV00, EP02, EKGP03]. Besides considering more complex motifs of a limited type only, with two boxes at most, the algorithms for detecting such motifs are in general naive: they either exhaustively enumerate all possible motifs of two boxes separated by a distance [vHRCV00], or discover them by crossing the lists of occurrences of single motifs detected in a previous step [EP02, EKGP03]. In the first case, the method is severely limited in the length of the motifs it can identify and in the distance between them, which is usually fixed. In the second case, detecting motifs with two boxes by crossing the lists of single motifs takes time at least quadratic in the number of such single motifs and their occurrences. To address this problem, the lists for single motifs are trimmed by statistical significance before the crossing operation. However, a motif with two boxes may be statistically significant even though none of the boxes taken individually are. Indeed, one of the main interests in seeking for complex motifs directly lies in this fact.

Chapter 3

Extraction of structured motifs

In this chapter, we provide some background on the approaches used to solve the structured motif extraction problem. First, we present a relevant data structure widely used in string processing problems, called *suffix tree*. Next, we outline an algorithm devised by Sagot [Sag98] for the extraction of single motifs. Finally, we present two algorithms proposed by Marsan and Sagot [MS00] for structured motif extraction.

3.1 Basic data structures

3.1.1 Suffix trees

A suffix tree is a data structure built over all suffixes of a string. Such a data structure exposes the internal structure of a string and is often used to solve many string problems. The construction of a suffix tree in linear-time is a problem already addressed by Weiner [Wei73], by McCreight [McC76], and more recently by Ukkonen [Ukk95].

We define a suffix tree for an arbitrary string S of length n over an alphabet Σ as presented in [Gus97]. After that we generalize the suffix tree to handle a set of strings.

Definition 3.1.1 A *suffix tree* $\mathcal{T}$ of a n-character string S is a rooted directed tree with exactly n leaves, numbered 1 to n. Each internal node, other than the root R, has at least two children and each edge is labeled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf i, the label of the path from the root to the leaf i spells out exactly the suffix of S that starts at position i.

The previous definition of a suffix tree does not guarantee the existence of a suffix tree for any string S. The problem is that if a prefix of a suffix of S matches a suffix of S, the path for the later suffix would not end at a leaf. To avoid this problem we place at the end of S a special symbol that is not in the alphabet. Herein, we use the symbol \$ for the termination character. As an example, the suffix tree for string S =AGACAGGAGGC\$, over the DNA alphabet Σ={A,C,G,T}, is presented in Figure 3.1.

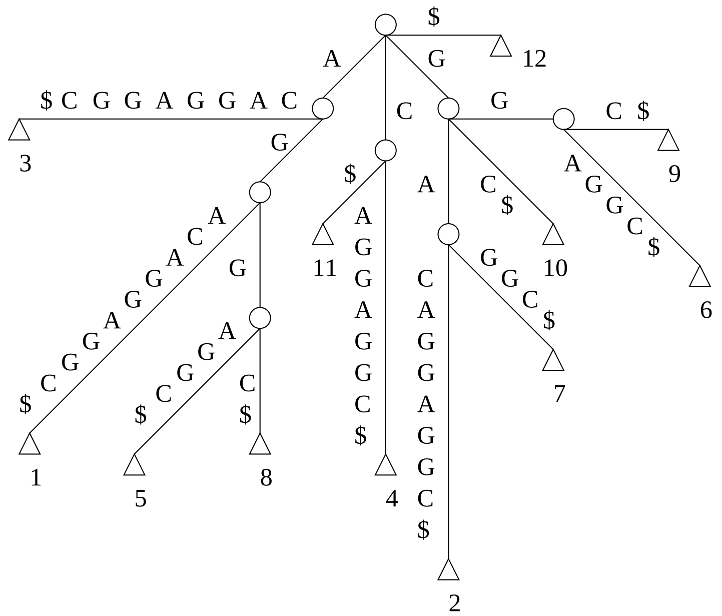

Figure 3.1: Suffix tree for string AGACAGGAGGC\$.

The suffix tree construction for a set of strings, called a *generalized suffix tree*, can be easily achieved by consecutively building the suffix tree for each string of the set. The resulting suffix tree is built in time proportional to the sum of all the string lengths. A way to distinguish the different strings in the generalized suffix tree is to convert the leaf number of the single string suffix tree to two numbers, one identifying the string and the other identifying the starting position in that string. As an example, the generalized suffix tree for strings S_1=TACTA\$ and S_2=CACTCA\$, over the DNA alphabet Σ={A,C,G,T}, is presented in Figure 3.2.

Contrarily to the motif localization problem, in the motif extraction problem the starting position of a suffix in a string is not relevant, only the identification of the string in the input set is required. A usual way to distinguish the input strings, in a motif extraction problem,

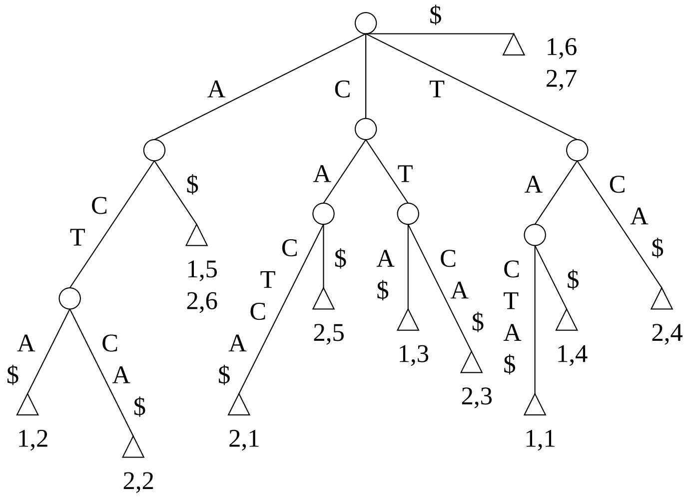

Figure 3.2: Generalized suffix tree for strings TACTA$ and CACTCA$.

is by storing at each tree node v a Boolean array, called the $Colors_v$ array [Sag98], usually implemented as a bit vector. Such array indicates the strings in the input set that contain the suffix labelling the path from the root to the tree node v. As an example, the generalized suffix tree with *Colors* for the same strings S_1=TACTA$ and S_2=CACTCA$, over the DNA alphabet Σ={A,C,G,T}, is presented in Figure 3.3.

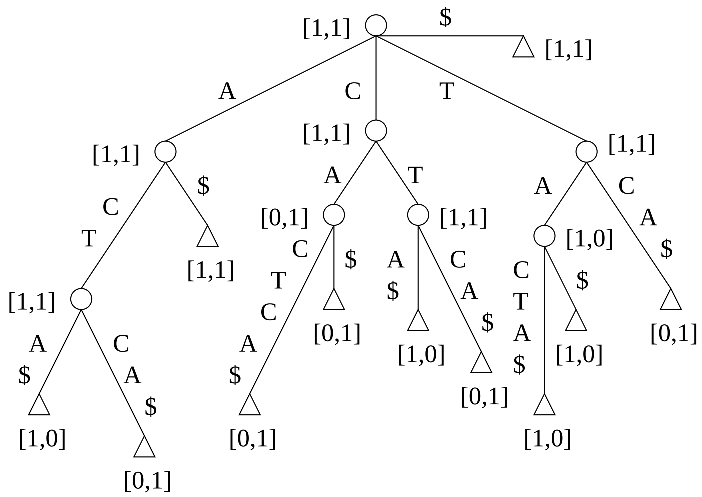

Figure 3.3: Generalized suffix tree with *Colors* for strings TACTA$ and CACTCA$.

We are not going to present any of the algorithms for the construction of suffix trees. However, it is worthwhile to notice that, by comparing all the linear-time construction algorithms [Wei73, McC76, Ukk95], Ukkonen's method [Ukk95] is one that uses less space in practice, therefore being the method of choice for most problems requiring the construction of a suffix tree. Besides being a space saving method, Ukkonen's algorithm uses a certain auxiliary structure that accounts for the most important acceleration element of the construction algorithm. This structure may be useful in some problems and it is defined as follows.

Definition 3.1.2 Let $x\alpha$ denote an arbitrary string, where x denotes a single character and α denotes a (possibly empty) substring. For an internal node v with path label $x\alpha$, if there is another node $S_l(v)$ with path label α, then a pointer from v to $S_l(v)$ is called a *suffix-link*.

As a special case, if α is empty, then the suffix-link from an internal node with path label $x\alpha$ goes to the root node. As an example, suffix-links are represented in the suffix tree for string S =AGACAGGAGGC$ in Figure 3.4.

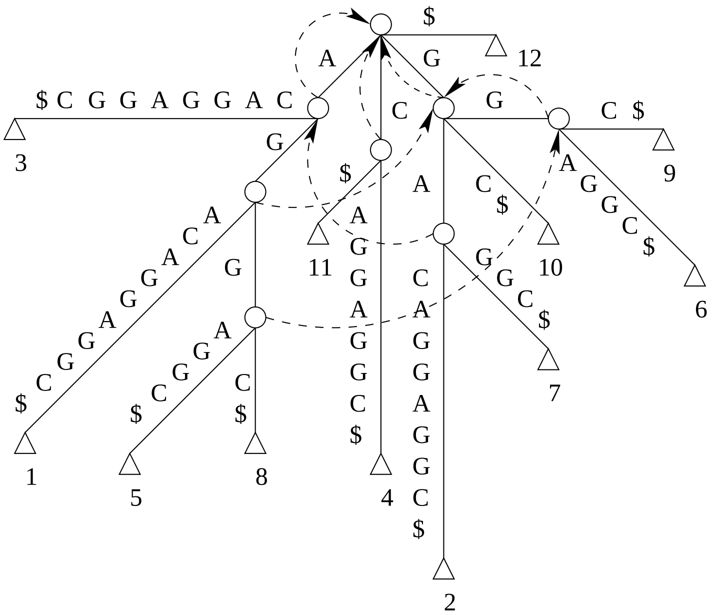

Figure 3.4: Suffix tree with suffix-links for string AGACAGGAGGC$.

3.2 Single motif extraction algorithm

Algorithms for single motif extraction address the extraction of consensus sequences that occur in a subset of the input sequences. In this section we present an algorithm, proposed by Sagot [Sag98], to extract single motifs. A suffix tree is used to find such motifs in a set of N input sequences over an alphabet Σ. We start by introducing some notation.

Definition 3.2.1 A *model* is a non-empty string over Σ, that is, it is an element in Σ^+.

A measure of the difference between two sequences of same length over Σ is given by the *Hamming distance*, which is defined as the minimum number of substitutions to transform one sequence into another.

Definition 3.2.2 A model m is said to have an *e-occurrence*, or simply an *occurrence*, in the input sequences, if there is one word u in the input sequences such that the Hamming distance between u and m is less than or equal to e.

Definition 3.2.3 A model is said to be a *valid model* if it has an occurrence in at least q input sequences, where q is called the *quorum*.

A valid model is also called a *motif*. In the literature it is common to abuse of notation by using model and motif interchangeably. In this thesis, motif is used when referring to a valid model only.

Definition 3.2.4 A *node-occurrence* of a model m is represented by a pair (v, e_v) where v is a tree node and $e_v \leq e$ is the Hamming distance between the label of the path from the root to v and m.

We are now able to describe the algorithm to extract single motifs [Sag98]. At first a suffix tree $\mathcal{T}$ is built for all input sequences. The suffix tree needs to be modified in order to store at each tree node v the $Colors_v$ Boolean array of size N. For the sake of exposition, a *suffix trie* (data structure similar to a suffix tree but with the tree arcs labeled by a single letter) is considered instead of a suffix tree. We shall refer to it as a suffix tree since adapting the algorithm to deal with a compact tree is straightforward. Given a maximum number e of substitutions allowed, it has been shown by Sagot [Sag98] that extracting all valid k-size models can be done by simultaneously and recursively traversing, in a depth-first way, the lexicographic trie $\mathcal{M}$ of all possible valid models and the suffix tree $\mathcal{T}$ of all input sequences. Observe that, when substitutions are allowed, models that are not represented in the suffix tree may be valid models. In this case, the models that need to be checked for validity are all sequences with Hamming distance at most e from the suffixes of the tree $\mathcal{T}$. Moreover, the lexicographic trie $\mathcal{M}$ is the trie of all these models pruned at the nodes where the quorum is no longer verified. In practice, $\mathcal{M}$ is never built but can be virtually traversed by a more complex traversal over $\mathcal{T}$. Moreover, if no substitutions are allowed and the quorum equals 1 then $\mathcal{M}$ and $\mathcal{T}$ present the same models.

We present the pseudo-code of the algorithm to spell k-size motifs with up to e mismatches in Algorithm 3.1, page 18. The algorithm makes use of the following variables and functions:

- the variable Ext_m, implemented as a bit vector, is a set of symbols by which the model m may be extended at the next step of the algorithm;
- the variable Occ_m is a set of node-occurrences of the model m;
- the variable $Colors_x$ is a Boolean array defined as

$$Colors_x[i] = \begin{cases} 1 & \text{if at least one leaf in the subtree rooted at } x \\ & \text{represents a suffix of the } i\text{-th input sequence } S_i \\ 0 & \text{otherwise} \end{cases} ;$$

- the variable $Colors_m$, similarly to $Colors_x$, is a Boolean array defined as

$$Colors_m[i] = \begin{cases} 1 & \text{if } m \text{ occurs in } S_i \\ 0 & \text{otherwise} \end{cases} ;$$

- the variable CSS_x, meaning the *color set size* of a node x [Hui92], is the number of different leaf colors in the subtree rooted at x, where a leaf is assigned a color i if it represents a suffix of S_i, that is, $CSS_x = \sum_{i=1}^{N} Colors_x[i]$;
- the variable CSS_m, similarly to CSS_x, is defined as $CSS_m = \sum_{i=1}^{N} Colors_m[i]$;
- the variable *minseq* indicates the minimum value of CSS_x for all node-occurrences x of the extended model;
- the variable *maxseq* indicates the sum of the values CSS_x for all node-occurrences x of the extended model;
- the function KeepMotif stores all information concerning valid models for printing later.

The Algorithm 3.1 is called with $(0, \lambda, Occ_\lambda = \{(R, 0)\}, Ext_\lambda)$, where λ is the empty sequence, R is the suffix tree root and

$$Ext_\lambda = \begin{cases} \Sigma & \text{if } e > 0 \\ \text{label of the branches leaving } R & \text{otherwise} \end{cases}.$$

Next, we present the time and space complexity for the Algorithm 3.1. These results have already been proved by Sagot [Sag98], and so we omit the proof here. Recall that, N is the number of input sequences and n is the average length of an input sequence. Moreover, n_k is the number of tree nodes at depth k and $\nu(e, k)$ is the number of distinct words that are at a Hamming distance at most e from a k-long word. It is easy to see that the following upper bound for $\nu(e, k)$ holds:

$$\nu(e, k) = \sum_{i=0}^{e} \begin{pmatrix} k \\ i \end{pmatrix} (|\Sigma| - 1)^i \leq k^e |\Sigma|^e.$$

Proposition 3.2.5 Algorithm 3.1 requires $O(Nn_k\nu(e, k))$ time and $O(N^2 n)$ space.

3.3 Structured motif extraction algorithms

Algorithms for structured motif extraction address the extraction of consensus motifs that appear together in a well-ordered and regularly spaced manner. Structured motifs were first introduced by Marsan and Sagot [MS00], and in this section we are going to present two algorithms proposed by the authors. A structured model can be described as an ordered collection of p boxes, a maximum number e of substitutions allowed for each box, and an interval of distance for each pair of consecutive boxes.

To set up the algorithms to extract structured motifs, we have to introduce some notation.

Definition 3.3.1 A *structured model* is a pair (m, d) where:

- $m = (m_i)_{1 \leq i \leq p}$ is a p-tuple of single models, denoting the p boxes;
- $d = (d_{min_i}, d_{max_i}, \delta_i)_{1 \leq i \leq p-1}$ is a $(p-1)$-tuple of triplets, denoting the $p-1$ intervals of distance.

Algorithm 3.1 SpellMotifs, single motif extraction

SpellMotifs(depth l, model m, occurrences Occ_m, extension Ext_m)

if ($l == k$) KeepMotif(m)

else

for each symbol α in Ext_m

$maxseq = 0$

$minseq = \infty$

$Colors_{m\alpha} = \overrightarrow{0}$

$Ext_{m\alpha} = \emptyset$

$Occ_{m\alpha} = \emptyset$

for each pair (x, x_{err}) in Occ_m

if there is a branch b leaving node x with a label starting with α

let x' be the node reached by following branch b from x

add to $Occ_{m\alpha}$ the pair (x', x_{err})

$maxseq = maxseq + CSS_{x'}$

if ($CSS_{x'} < minseq$) $minseq = CSS_{x'}$

$Colors_{m\alpha} = Colors_{m\alpha} + Colors_x$

$$Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup \text{label}_{b'} \text{ for } b' \text{ leaving } x' & \text{if } x_{err} = e \\ \Sigma & \text{otherwise} \end{cases}$$

if ($x_{err} < e$)

for each branch b leaving x except for the one labeled with α

let x' be the node reached by following branch b from x

add to $Occ_{m\alpha}$ the pair $(x', x_{err} + 1)$

$maxseq = maxseq + CSS_{x'}$

if ($CSS_{x'} < minseq$) $minseq = CSS_{x'}$

$Colors_{m\alpha} = Colors_{m\alpha} + Colors_x$

$$Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup \text{label}_{b'} \text{ for } b' \text{ leaving } x' & \text{if } x_{err} = e - 1 \\ \Sigma & \text{otherwise} \end{cases}$$

if ($maxseq < q$) return

else if ($minseq \geq q$ or $CSS_{m\alpha} \geq q$) SpellMotifs($l + 1$, $m\alpha$, $Occ_{m\alpha}$, $Ext_{m\alpha}$)

The terms $d_{min_i} \leq d_{max_i}$ represent a minimum and maximum allowed distance between consecutive boxes and δ_i an allowed neighbourhood within that distance. The δ_i is omitted when $\delta_i = (d_{max_i} - d_{min_i} + 1)/2$.

Definition 3.3.2 A structured model (m, d) is said to be a *valid structured model* if for all $1 \leq i \leq p-1$ and for all occurrences u_i of m_i, there exist occurrences $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_p$ of the single motifs $m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_p$ such that:

- $u_1, \ldots, u_p$ belong to the same input sequence;
- there exists d_i, with $d_{\mathrm{min}_i} + \delta_i \leq d_i \leq d_{\mathrm{max}_i} - \delta_i$, such that the distance between the end position of u_i and the start position of u_{i+1} in the sequence is in $[d_i - \delta_i, d_i + \delta_i]$;
- d_i is the same for p-tuples of occurrences present in at least q distinct sequences.

As for single motifs, a valid structured model is also called a *structured motif.*

3.3.1 Jumping in the suffix tree

In the following we describe the first published algorithm to extract structured motifs [MS00].

For the sake of exposition, we assume that all boxes of the structured motifs have the same size k, with distances between them over the interval $[d_{min}, d_{max}]$, and maximum allowed error e per box. Moreover, we consider only structured motifs with two boxes. The algorithm works in the following way. At first a suffix tree is built for all input sequences, storing at each tree node v the N size $Colors_v$ Boolean array. The extraction of structured motifs starts by extracting single motifs of length k, one at a time, as described in Section 3.2. Once a single valid model m_1 is obtained the extraction of all single models m_2 with which m_1 could form a valid structured model $((m_1, m_2), (d_{min}, d_{max}))$ starts. The extraction of the second box m_2 is done as follows. For each node-occurrence v of a first box m_1 (Figure 3.5a), a jump is made on the suffix tree to the descendants of v situated at distances $[d_{min}, d_{max}]$ from v. The nodes reached in this way are the potential start node-occurrences of the second boxes (Figure 3.5b). Single extractions of all possible second boxes proceed from these nodes (Figure 3.5c).

The ExtractMotifs algorithm for p boxes is presented in Algorithm 3.2, page 20. It makes use of the $PotentialStarts$ variable, which stores the potential start node-occurrences of the next box being extracted (Figure 3.5b).

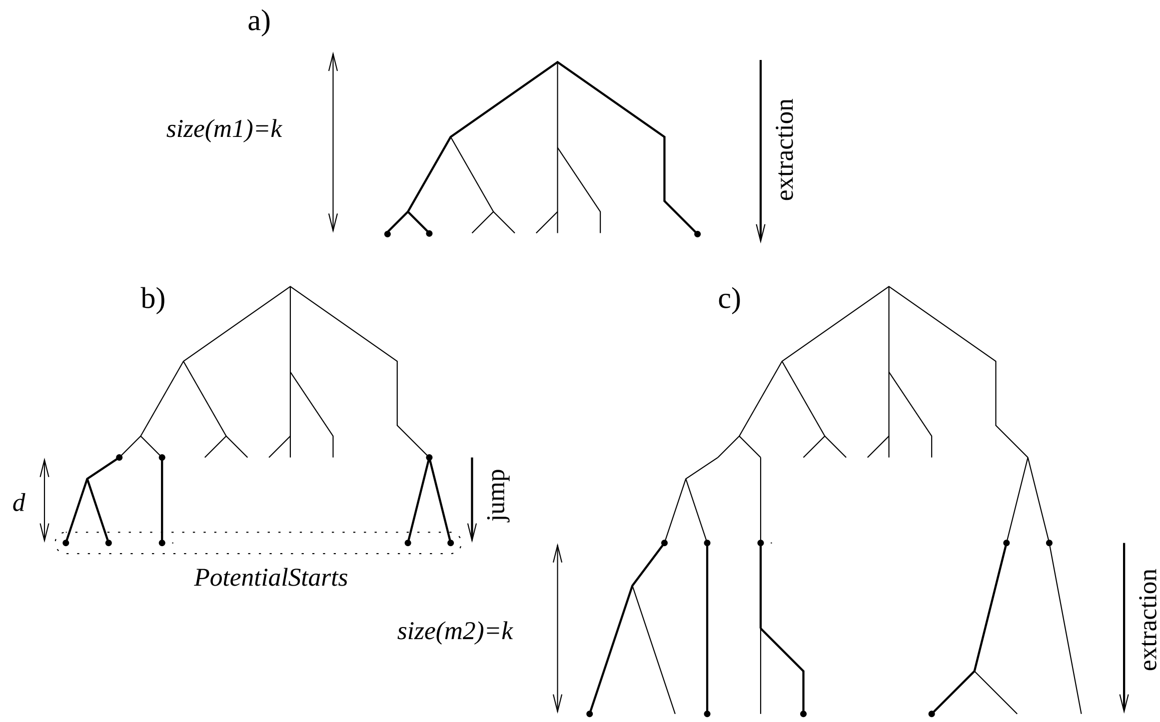

Figure 3.5: Extracting structured motifs jumping down in the tree.

Algorithm 3.2 ExtractMotifs, structured motif extraction jumping in the suffix tree

ExtractMotifs(model m, box i)

1. for each node-occurrence v of m
2. if $(i > 1)$
3. put in $PotentialStarts$ the children of v at levels $(i-1)k + (i-1)d_{min}$ to $(i-1)k + (i-1)d_{max}$
4. else put v in $PotentialStarts$
5. for each motif m_i obtained by traversing $\mathcal{T}$ from the node-occurrences in $PotentialStarts$
6. if $(i < p)$ ExtractMotifs$(m_1 \ldots m_i, i+1)$
7. else KeepModel$(\langle(m_1, \ldots, m_p), (d_{min}, d_{max})\rangle)$

Complexity analysis for constant distance between boxes

Next, we establish the time and space complexity for Algorithm 3.2. Even though such results have already been shown by Marsan and Sagot [MS00], we present a detailed proof here since it gives a deep insight over other results that will be presented further in this thesis. Moreover, since the complexity is hard to compute we start by considering the case when $d_{min} = d_{max} = d$.

Proposition 3.3.3 Algorithm 3.2 takes $O(Nn_{pk+(p-1)d}\nu^p(e,k))$ time.

Proof: To compute the complexity of Algorithm 3.2 we have to consider the total number of operations needed to build the p parts of all structured motifs. This means that we have to calculate the cost of all visits we may do to nodes between the root and level $pk+(p-1)d$ (the deepest level ever reached).

Start by noticing that when spelling all parts of a structured motif we are working with nodes between the root and level $pk+(p-1)d$, and because suffix trees are compact, being at least binary, there are at most $2n_{pk+(p-1)d}$ such nodes. Hence, the total number of visits we may do to nodes between the root and level $pk+(p-1)d$ is upper bounded by the total number of visits we may do to nodes at level $pk+(p-1)d$.

Moreover, when no substitutions are allowed, there are at most $n_{pk+(p-1)d}$ ways of spelling all structured motifs. In this case, the total number of visits to nodes at level $pk+(p-1)d$ is given by $n_{pk+(p-1)d}$. However, when up to e substitutions are allowed, a node at level $pk+(p-1)d$ may be visited $\nu^p(e,k)$ times. Hence, the total number of visits to nodes at level $pk+(p-1)d$ is upper bounded by $n_{pk+(p-1)d}\nu^p(e,k)$.

Finally, since each visit to a node requires the access to the *Colors* array, which takes $O(N)$ time, we can conclude that Algorithm 3.2 takes $O(Nn_{pk+(p-1)d}\nu^p(e,k))$ time. □

Proposition 3.3.4 Algorithm 3.2 takes $O(N^2n)$ space.

Proof: The construction of the suffix tree requires $O(Nn)$ and the storage of the *Colors* array in each node of the tree requires $O(N)$. □

Complexity analysis for variable distance between boxes

Herein we establish the time complexity for Algorithm 3.2, considering the general case where $d_{min} \leq d_{max}$. For the complexity analysis that follows we consider that $PotentialStarts$ does not have repetitions, even if this assumption is not explicit in the algorithm proposed by Marsan and Sagot [MS00]. This can be easily achieved, by marking the nodes reached at $[(i-1)k+(i-1)d_{min}, (i-1)k+(i-1)d_{max}]$ levels with a bit, without changing asymptotically time and space complexity. Furthermore, in the following we assume $\Delta = d_{min} - d_{max} + 1$.

Proposition 3.3.5 Algorithm 3.2 takes

$$O(p\Delta^2 n_{(p-1)k+(p-1)d_{max}} \nu^{p-1}(e,k) + Np\Delta n_{pk+(p-1)d_{max}} \nu^{p}(e,k))$$

time.

Proof: To compute the complexity of Algorithm 3.2 we have to consider the cost of all visits we may do to nodes between the root and level $pk + (p-1)d_{max}$. However, contrary to what happened in Proposition 3.3.3, this cost is not upper bounded by the cost of all visits we may do to nodes at level $pk + (p-1)d_{max}$. This happens because, by allowing a variable distance between consecutive boxes, more than one final visit may occur to the nodes at the lower levels of the tree. Taking this observation into account, we can upper bound the total cost of the algorithm using two parcels. First, we have to consider the cost of all visits to nodes at $[(p-1)k + (p-1)d_{min}, (p-1)k + (p-1)d_{max}]$ levels, corresponding to the computation of $PotentialStarts$, just before the extraction of the last box m_p. Second, we have to consider the cost of all visits we may do to nodes at $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ levels, corresponding to visits when spelling the last box m_p of the models being extracted.

As in Proposition 3.3.3, we start by analyzing the case where no substitutions are allowed.

To compute the first parcel concerning $PotentialStarts$, note that each visit to a node at a level within the interval $[(p-1)k + (p-1)d_{min}, (p-1)k + (p-1)d_{max}]$ is upper bounded by the number of ascendants at $[(p-1)k + (p-2)d_{min}, (p-1)k + (p-2)d_{max}]$ levels times the number of visits each ascendant may do to a descendant node. In the worst case and for each descendant node, the number of ascendants is $(p-2)(d_{max} - d_{min}) + 1 = (p-2)(\Delta - 1) + 1$ whereas the number of visits an ascendant may do to the descendant is Δ. Hence, the number of visits we may do to nodes at $[(p-1)k + (p-1)d_{min}, (p-1)k + (p-1)d_{max}]$ levels is $O(((p-2)(\Delta-1)+1)\Delta n_{(p-1)k+(p-1)d_{max}}) = O(p\Delta^2 n_{(p-1)k+(p-1)d_{max}})$.

To compute the second parcel concerning the last extraction step, note that each visit to a node at a level within the interval $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ is upper bounded by the number of ascendants at $[(p-1)k+(p-1)d_{min}, (p-1)k+(p-1)d_{max}]$ levels times the number of visits each ascendant may do to a descendant node. In the worst case and for each descendant node, the number of ascendants is $(p-1)(d_{max}-d_{min})+1 = (p-1)(\Delta-1)+1$ whereas the number of visits an ascendant may do to the descendant is 1 since we assume no repetitions in $PotentialStarts$. Hence, the number of visits to nodes at $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ levels is $O(((p-1)(\Delta-1)+1)n_{pk+(p-1)d_{max}}) = O(p\Delta n_{pk+(p-1)d_{max}})$.

Reasoning as in Proposition 3.3.3, when errors are allowed, the total number of visits is $O(p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + p\Delta n_{pk+(p-1)d_{max}}\nu^{p}(e,k))$. Finally, since the extraction process requires the access to the $Colors$ array, which takes $O(N)$ time, Algorithm 3.2 takes $O(p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Np\Delta n_{pk+(p-1)d_{max}}\nu^{p}(e,k))$ time. □

Observe that when $\Delta = 1$ the complexity result of Proposition 3.3.5 agrees with the complexity result of Proposition 3.3.3 since the p term disappears. In fact, in the first parcel concerning the computation of $PotentialStarts$ for the last box, the number of visits to nodes at $[(p-1)k+(p-1)d_{min}, (p-1)k+(p-1)d_{max}]$ levels, which is upper bounded by $((p-2)(\Delta-1)+1)\Delta n_{(p-1)k+(p-1)d_{max}}$, becomes equal to $n_{(p-1)k+(p-1)d_{max}}$. Moreover, in the second parcel concerning the effort of extracting the last box of the structured models, the number of visits we may do to nodes at $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ levels, which is upper bounded by $((p-1)(\Delta-1)+1)n_{pk+(p-1)d_{max}}$, becomes equal to $n_{pk+(p-1)d_{max}}$. Hence, the complexity reduces to $O(n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Nn_{pk+(p-1)d_{max}}\nu^{p}(e,k)) = O(Nn_{pk+(p-1)d_{max}}\nu^{p}(e,k))$.

We do not address the space complexity since it is the same as for constant distances between boxes, presented in Proposition 3.3.4.

3.3.2 Jumping in the suffix tree using suffix-links

Herein, we present the second published algorithm to extract structured motifs [MS00], capitalizing on the suffix-link data structure of suffix trees. This algorithm achieves an exponential worst case time gain relatively to Algorithm 3.2 presented in Section 3.3.1.

For clarity of exposition, we consider only structured motifs with two boxes. Moreover, we assume that each box has the same size k, distanced by some value in the interval $[d_{min}, d_{max}]$, and the same maximum allowed error e per box. The algorithm works as follows. At first a suffix tree is built for all input sequences, storing at each tree node v the N size $Colors_v$ Boolean array. The structured motif extraction algorithm starts by extracting single motifs of length k, one at a time, as described in Section 3.2. For each node-occurrence v of a first box m_1 (Figure 3.6a), a jump is made on the suffix tree to the descendants of v situated at distances $[k+d_{min}, k+d_{max}]$ from v (Figure 3.6b). The content of the Boolean $Colors$ array of the nodes

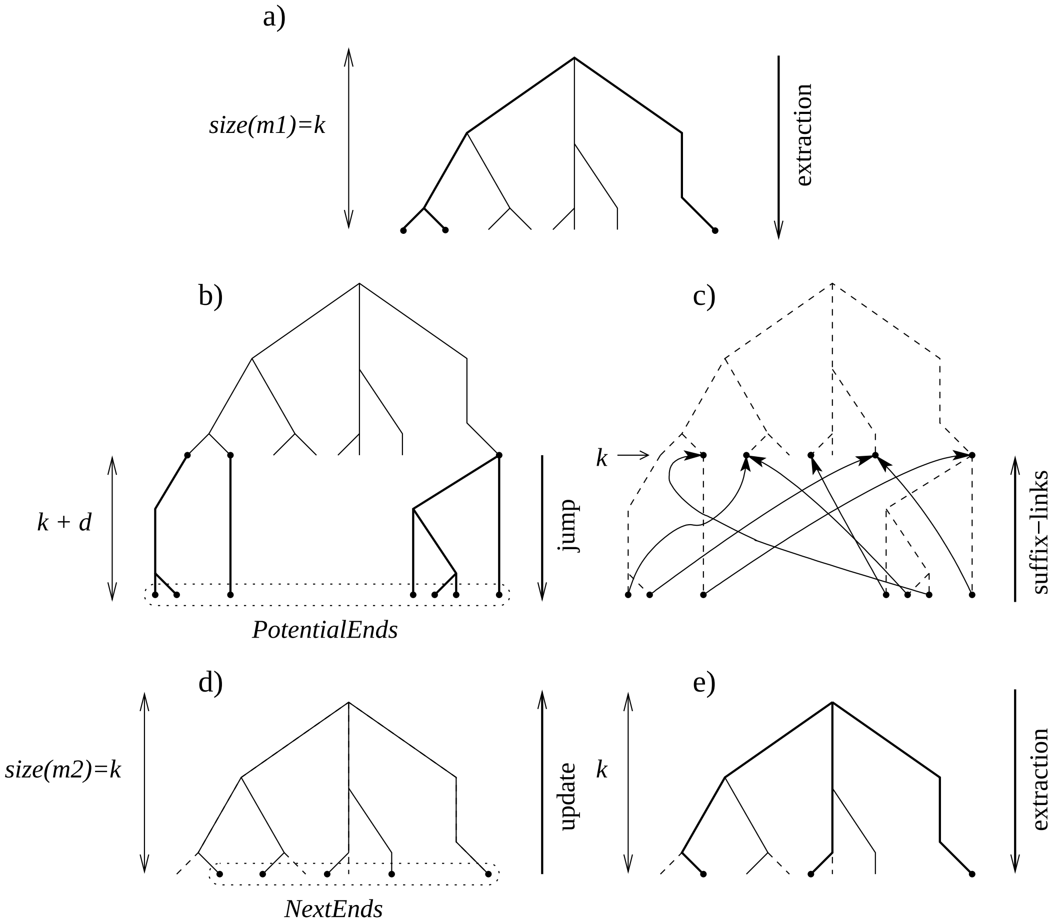

Figure 3.6: Extracting structured motifs following suffix-links.

reached at these lower levels is grabbed and carried along the unique path of suffix-links that leads back to a node at level k (Figure 3.6c). Back at level k, the grabbed information is used to temporarily and partially modify the suffix tree (Figure 3.6d). The extraction of the second box m_2 of a potentially valid structured model then proceeds in the same way (Figure 3.6e). Once the operation of extracting all possible valid motifs $\langle(m_1, m_2),(d_{min}, d_{max})\rangle$ has ended, the suffix tree is restored to its previous state. The construction of another single motif m_1

follows, and the whole process unwinds in a recursive way until all valid structured motifs are extracted.

This second algorithm to extract structured motifs restricts the single extraction of all boxes composing the structured models to the first k levels of the tree. Comparing with Algorithm 3.2 presented in Section 3.3.1, it merges in a higher level some branches of lower levels, easing the process of the numerous single extractions and leading to an exponential time gain in the worst case analysis, as we shall see later. On the other hand, more space is needed by the second algorithm to restore the tree after each modification it undergoes. Marsan and Sagot proposed a family of $L(i)$ arrays for that purpose. Notice that, in a general case of p boxes, up to $(p-1)$ arrays are needed. In fact, the $L(i)$ array, $1 < i \leq p$, stores the state of the nodes at level k for the $(i-1)$-th box, and there is no need to store it for the first box.

Another subtle detail shows up in this second algorithm when $p > 2$. Note that if we descend to lower levels directly from node-occurrences at level k for boxes $i > 2$, like we do for $i = 2$ (Figure 3.6b), we would not miss any potential end node-occurrence but we could get more (notice that the suffix tree gets sparser at lower levels). For that reason, we need to keep for each box i, with $1 < i < p$, and for each node v_k, at level k, reached from the following up of the suffix-links from nodes at $[ik + (i-1)d_{min}, ik + (i-1)d_{max}]$ levels to level k (Figure 3.6c), a list $Lptr_{v_k}(i)$ of pointers to the correspondent nodes at $[ik+(i-1)d_{min}, ik+(i-1)d_{max}]$ levels. As an example, the $Lptr_{v_k}(2)$ arrays for the extraction of a third box, where the first and second box were extracted as in Figure 3.6, are depicted in Figure 3.7: $Lptr_{v1_k}(2) = \{v_{11}\}$, $Lptr_{v2_k}(2) = \{v_6, v_{10}\}$, $Lptr_{v3_k}(2) = \{v_9\}$, $Lptr_{v4_k}(2) =$

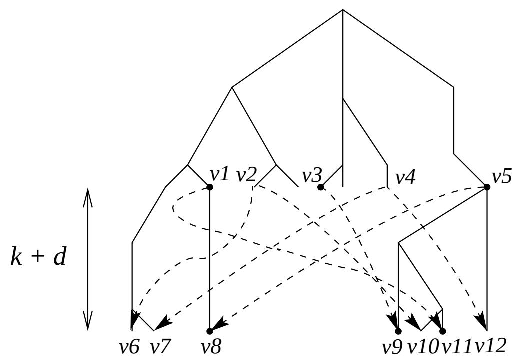

Figure 3.7: The $Lptr_{v_k}(2)$ arrays for the extraction of a third box.

$\{v_7, v_{12}\}$ and $Lptr_{v_{5_k}}(2) = \{v_8\}$ (notice the symmetry between suffix-links at Figure 3.6c and the $Lptr_{v_k}(2)$ arrays in Figure 3.7). In this example we only have represented $Lptr_{v_k}(2)$ since, in a general case of p boxes, we need at most $(p-2)$ $Lptr_{v_k}$ arrays for each node v_k at level k, one for each box of the structured motif, except for the first and the last box. In Figure 3.8 we present the extraction of a third box using the $Lptr_{v_k}(2)$ array, from Figure 3.7. For each

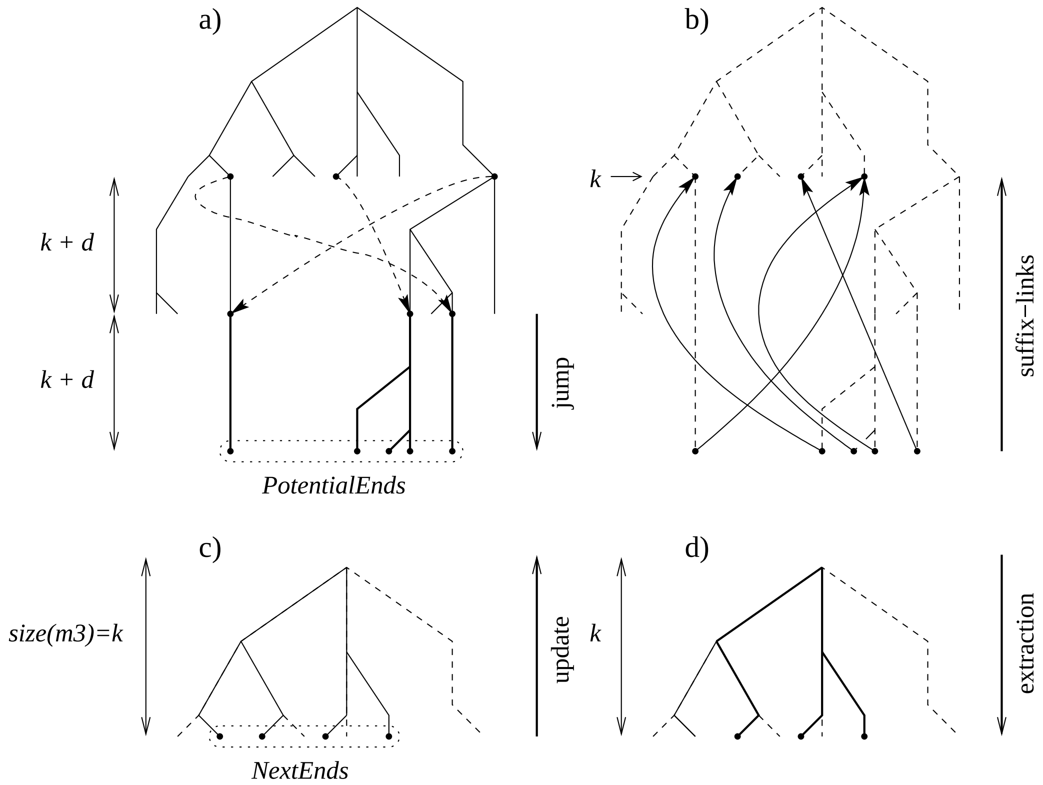

Figure 3.8: Extracting a third box following suffix-links and $Lptr_{v_k}(2)$ array.

node-occurrence of a second box found (Figure 3.6e), we follow the pointers stored in $Lptr_{v_k}(2)$ at the node-occurrences at level k, and then from the nodes reached at $[2k+d_{min}, 2k+d_{max}]$ levels descend to nodes at $[3k+2d_{min}, 3k+2d_{max}]$ levels (Figure 3.8a). In this case, only $Lptr_{v_{1_k}}(2)$, $Lptr_{v_{3_k}}(2)$ and $Lptr_{v_{5_k}}(2)$ are used. From that point on, the extration proceeds as for $i = 2$, until the process of extracting all possible valid models $\langle(m_1, m_2, m_3),(d_{min}, d_{max})\rangle$ has ended (Figure 3.8b to Figure 3.8d).

The ExtractMotifs algorithm for p boxes is presented in Algorithm 3.3, page 28, and it makes use of the following variables and functions:

– the variable *PotentialEnds* stores the potential end node-occurrences of the next box being extracted (Figure 3.6b);

- the variable $NextEnds$ stores the nodes at level k needed to update the suffix tree (Figure 3.6d);

- the variables $L(i)$, $1 < i \leq p$, store information to restore the suffix tree after each update it undergoes;

- the variables $Lptr_{v_k}(i)$, $1 < i < p$, store pointers from level k to lower levels from where to proceed with the descent (Figure 3.7);

- the function UpdateTree updates the Boolean arrays from the nodes in $NextEnds$ to the root in the following way: if nodes $\overline{z}$ and $\widehat{z}$ have the same parent z, then $Colors_z = Colors_{\overline{z}} + Colors_{\widehat{z}}$. Any arc from the root that does not have a node in $NextEnds$ is not part of the updated tree, nor are the subtrees rooted at its node in $NextEnds$;

- the function RestoreTree restores the Boolean arrays from the nodes in $L(i)$ to the root in the following way: if nodes $\overline{z}$ and $\widehat{z}$ have the same parent z, then $Colors_z = Colors_{\overline{z}}$ + $Colors_{\widehat{z}}$. Any arc from the root is part of the restored tree.

Complexity analysis for constant distance between boxes

Next, we establish the time and space complexity for Algorithm 3.3. The time complexity result is different from what is presented by Marsan and Sagot [MS00] due to an acknowledged imprecision in the later. For that reason, the detailed proof follows, for the case when we have $d_{max} = d_{min} = d$.

Proposition 3.3.6 Algorithm 3.3 takes $O(Ns_p(k,d)\nu^p(e,k) + Nn_{pk+(p-1)d}\nu^{p-1}(e,k))$ time, where $s_p(k,d) = \min\{n_k^p, n_{pk+(p-1)d}\}$.

Proof: We can parcel out the complexity of Algorithm 3.3 into three parts: first, the total number of operations needed to build the p parts of all structured motifs; second, the total number of operations needed to update $\mathcal{T}$, for all parts of a structured motif; third, the total number of operations needed to restore $\mathcal{T}$.

Algorithm 3.3 ExtractMotifs, structured motif extraction using suffix-links

ExtractMotifs(model m, box i)

for each node-occurrence v of m
    if $(i == 2)$
        put in $PotentialEnds$ the children w of v at levels $2k + d_{min}$ to $2k + d_{max}$
    else if $(i > 2)$
        for each pointer $v \mapsto z$ in $Lptr_{v_k}(i-1)$
            put in $PotentialEnds$ the children w of z
            at levels $ik + (i-1)d_{min}$ to $ik + (i-1)d_{max}$
    for each node-occurrence w in $PotentialEnds$
        follow suffix-link to node z at level k
        put node z in $L(i)$
        if $(1 < i < p)$ put pointer $z \mapsto w$ in $Lptr_{v_k}(i)$
        if (first time z is reached)
            $Colors_z = \overrightarrow{0}$
            put z in $NextEnds$
        $Colors_z = Colors_z + Colors_w$
UpdateTree($\mathcal{T}, NextEnds$)
for each motif m_i obtained by traversing $\mathcal{T}$ from the root
    if $(i < p)$ ExtractMotifs($m_1 \ldots m_i, i + 1$)
    else KeepMotif($\langle(m_1, \ldots, m_p), (d_{min}, d_{max})\rangle$)
RestoreTree($\mathcal{T}, L(i)$)

In order to compute the complexity of building the p parts of all structured motifs we have to calculate the cost of all visits we may do to nodes between the root and level k (the deeper level ever reached).

Start by noticing that when spelling all parts of a motif we are working with nodes between the root and level k only, and because suffix trees are compact, being at least binary, there are at most $2n_k$ such nodes. Hence, the total number of visits we may do to nodes between the root and level k is upper bounded by the total number of visits we may do to nodes at level k.

Moreover, when no substitutions are allowed, there are at most $s_p(k,d)$ ways of spelling all structured motifs. In this case, the number of visits to nodes at level k can be given by

$$\begin{aligned}\sum_{i=1}^{p} \min\left\{n_k^i, n_{ik+(i-1)d}\right\} &\leq \min\left\{\sum_{i=1}^{p} n_k^i, \sum_{i=1}^{p} n_{ik+(i-1)d}\right\} \\ &= O(\min\left\{2n_k^p, 2n_{pk+(p-1)d}\right\}) \\ &= O(s_p(k,d)).\end{aligned}$$

However, when up to e substitutions are allowed, a node at level k may be visited

$$\sum_{i=1}^{p} \nu^i(e,k) = O(\nu^p(e,k))$$

times more. Hence, the total number of visits to nodes at level k is in $O(s_p(k,d)\nu^p(e,k))$.

Finally, since each visit to a node requires the access to the *Colors* array, which takes $O(N)$ time, building the p parts of all structured motifs takes $O(Ns_p(k,d)\nu^p(e,k))$ time.

In order to compute the complexity of updating $\mathcal{T}$, for all parts of a structured motif, we need to count the total number of operations necessary to modify the first k levels of the suffix tree which is upper bounded by $Nn_{pk+(p-1)d}\nu^{p-1}(e,k)$. This corresponds to all visits done to nodes z coming from w for all motifs m_{p-1}. In addition, the propagation from node z to the root for all motifs m_{p-1} is also upper bounded by the same value.

To sum up, and since restoring the suffix tree is also upper bounded by the time to update $\mathcal{T}$, we conclude that Algorithm 3.3 takes $O(Ns_p(k,d)\nu^p(e,k)+Nn_{pk+(p-1)d}\nu^{p-1}(e,k))$ time. □

This algorithm exhibits an exponential worst case time gain relatively to Algorithm 3.2. Note that in the worst case scenario, the suffix tree is complete and we have $s_p(k,d) = \min\{|\Sigma|^{pk}, |\Sigma|^{pk+(p-1)d}\} = |\Sigma|^{pk} < n_{pk+(p-1)d} = |\Sigma|^{pk+(p-1)d}$ which reflects an exponential gain of the order of $|\Sigma|^{(p-1)d}$.

Proposition 3.3.7 Algorithm 3.3 takes $O(N^2n + Npn_k)$ space.

Proof: The first parcel is due to the construction of the suffix tree which requires $O(Nn)$ and the storage of the *Colors* array in each node of the tree which requires $O(N)$. The second parcel refers to the L arrays. There are at most n_k L arrays, one for each node of the suffix tree at level k, and we need up to $(p-1)$ $L(i)$ arrays for each node, one for each box of the structured motif except for the first one. Since each array $L(i)$ stores a *Colors* array which takes $O(N)$, the storage of the L arrays takes $O(N(p-1)n_k) = O(Npn_k)$. Finally, we need up to $(p-2)$ arrays $Lptr_{v_k}(i)$, one for each box of the structured motif, except for the first and the last one. Moreover, each $Lptr_{v_k}(i)$ array stores a set of pointers to nodes at a lower level in the suffix tree. Note that a $Lptr_{v_k}(i)$ array, stored at a node at level k, may store more than one pointer, but there is no repeated pointers over all $Lptr_{v_k}(i)$ arrays, stored at all nodes at level k. We can conclude that the total number of pointers in all $Lptr_{v_k}(i)$ arrays is upper bounded by the number of nodes at the deeper level ever stored, the level $(p-1)k + (p-2)d$. Hence, the storage of the $Lptr_{v_k}$ arrays takes $O((p-2)n_{(p-1)k+(p-2)d}) = O(pn_{(p-1)k+(p-2)d})$ space. Since $n_{(p-1)k+(p-2)d} < Nn$, we conclude that Algorithm 3.3 takes $O(N^2n + Npn_k)$ space. □

Complexity analysis for variable distance between boxes

Herein we establish the time complexity for Algorithm 3.3, considering the general case where $d_{min} \leq d_{max}$. In what follows we assume that the follow up of suffix-links to level k is never done twice for the same node, for each motif being found (see Figure 3.6c to recall the follow up operation), which is equivalent to saying that *PotentialEnds* has no repetitions. Even if this assumption is not explicit in the algorithm presented in the paper by Marsan and Sagot [MS00], it can be easily achieved without asymptotically changing time and space complexity.

Proposition 3.3.8 Algorithm 3.3 takes

$$O(Ns_p(k, d_{max})\nu^p(e, k) + (N + p\Delta^2)n_{pk+(p-1)d_{max}}\nu^{p-1}(e, k))$$

time, where $s_p(k, d_{max}) = \min\{n_k^p, n_{pk+(p-1)d_{max}}\}$.

Proof: In a similar way to Proposition 3.3.5, when we have variable distances between consecutive boxes, an extra parcel appears in this analysis. In fact, the parcel concerning the update of the suffix tree, presented in Proposition 3.3.6, is divided in two parcels. One corresponding to the computation of $PotentialEnds$ and another corresponding to the follow up of suffix-links to nodes at level k.

As in Proposition 3.3.6, we start by analyzing the case where no substitutions are allowed.

To compute the first parcel concerning $PotentialEnds$, note that each visit to a node at a level within the interval $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ is upper bounded by the number of ascendants at $[(p-1)k+(p-2)d_{min}, (p-1)k+(p-2)d_{max}]$ levels times the number of visits each ascendant may do to a descendant node. In the worst case and for each descendant node, the number of ascendants is $(p-2)(d_{max}-d_{min})+1 = (p-2)(\Delta-1)+1$ whereas the number of visits an ascendant may do to the descendant is Δ. Hence, the number of visits we may do to nodes at $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ levels is $O(((p-2)(\Delta-1)+1)\Delta n_{pk+(p-1)d_{max}}) = O(p\Delta^2 n_{pk+(p-1)d_{max}})$.

The second parcel concerning the follow up of suffix-links remains asymptotically the same due to two reasons. First, suffix trees are compact, being at least binary, and $2n_{pk+(p-1)d_{max}}$ is an upper bound for the total number of nodes at $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ levels. Second, $PotentialEnds$ has no repetitions and therefore each node within the levels $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ is followed up only once.

Finally, the parcels concerning the construction of the p parts of the structured models and the restoration of the suffix tree remain as in Proposition 3.3.6.

To sum up, reasoning as in Proposition 3.3.6, when errors are allowed, we have that Algorithm 3.3 takes $O(Ns_p(k, d_{max})\nu^p(e,k) + (N+p\Delta^2)n_{pk+(p-1)d_{max}}\nu^{p-1}(e,k))$ time. $\square$

Observe that, to compute $PotentialEnds$ when $\Delta = 1$, the number of visits to nodes at $[pk+(p-1)d_{min}, pk+(p-1)d_{max}]$ levels, upper bounded in the proof of Proposition 3.3.8 by $((p-2)(\Delta-1)+1)\Delta n_{pk+(p-1)d_{max}}$, becomes equal to $n_{pk+(p-1)d_{max}}$. So, the complexity result of Proposition 3.3.8 agrees with the complexity result of Proposition 3.3.6 since the p term disappears.

Space complexity is not presented since it is the same as for constant distances between boxes, presented in Proposition 3.3.7.

3.3.3 Extending the algorithms

The algorithms so far presented assumed that all single motifs m_i of a structured motif (m, d) have a unique fixed size k, the same substitution rate e and identical values for (d_{min}, d_{max}). The original paper that established both algorithms [MS00] presents extensions to handle boxes with variable length k_i, variable substitution rate e_i and variable intervals of distance (d_{min_i}, d_{max_i}). It also shows how to deal with restricted intervals of unknown limits $(d_{min_i}, d_{max_i}, \delta_i)$. Moreover, it emphasizes how local and global constraints can be introduced. In particular, besides fixing a maximum substitution rate for each box of a structured motif, it also establishes a maximum substitution rate e_{global} for the whole structured motif. Such a global rate allows to consider, in a limited way, possible correlations between boxes. Another presented local (or global) constraint imposes the frequency of one or more nucleotides in a box (or among all boxes) to be below or above a certain threshold.

3.3.4 Measuring statistical significance

Once all structured motifs have been extracted, they are classified according to their statistical significance, in an attempt to give them some biological pertinence. There is not in the literature a fully satisfactory method for evaluating such significance. Marsan and Sagot [MS00] used a data shuffling approach [KOB89] to evaluate the significance of a structured model. In order to obtain the statistical significance of the models found, a χ^2 test, with one degree of freedom, is performed on two contingency tables: one table expressing what was observed, and another corresponding to what is expected under the null hypothesis [PTVF93]. To derive the values in the contingency table for the null hypothesis several random shufflings are performed preserving the k-mer frequency distribution of the input sequences (a k-mer is a substring of length k). Both the number of sufflings and k are values given by the user (in general, 100 shufflings are considered conserving di- or tri- nucleotides). With this process, the probability of getting the models observed under the null hypothesis is estimated. Another type of statistics, based on a Z-score, was also tried by the authors [MS00].

Chapter 4

Extraction of structured motifs using box-links

In this chapter we propose a new exact algorithm to extract multiple binding site consensi from genomic sequences. Comparing with the algorithms presented in Section 3, which greatly influenced this new algorithm, the new one presents an exponential time and space gain in the worst case analysis.

We start by introducing a data structure devised by Allali and Sagot [AS03], called *factor tree*, which is roughly a suffix tree built only up to a certain level. The new algorithm uses a factor tree, unlike the previous approaches that make use of a suffix tree. The main advantage in the usage of a factor tree is that it requires less space in practice. Next, we propose a new data structure, called *box-links*, that accounts for the most important acceleration element of the new algorithm. Then, the new algorithm is presented, followed by a brief explanation on how to extend it and how to compute the statistical significance of the extracted motifs. We end this chapter with some experimental results that reinforce the theoretical ones.

4.1 Basic data structures

4.1.1 Factor trees

Factor trees are a new data structure to index strings, proposed by Allali and Sagot [AS03], very similar to suffix trees. This data structure, also called the *at most k-deep factor tree* or *k-factor tree*, indexes the factors of a string whose length does not exceed k, and only those.

Indeed, a factor tree is nothing more than a suffix tree pruned at the labels of depth k. We are not going to present the *k-factor tree* construction algorithm here, however, it is worthwhile to notice that it is based on Ukkonen's method [Ukk95]. As an example, consider the 5-deep factor tree for string S =AGACAGGAGGC$ presented in Figure 4.1. Note that the 5-deep

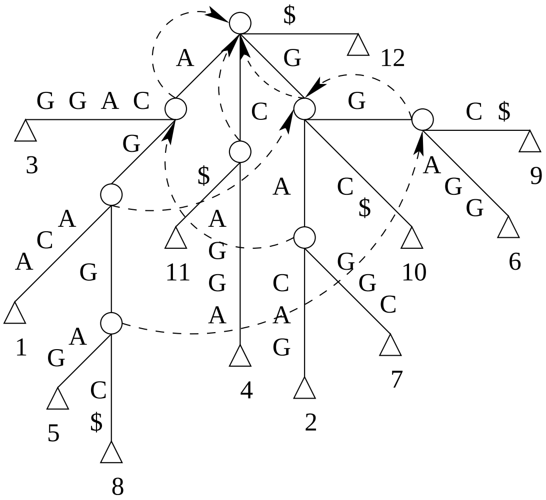

Figure 4.1: At most 5-deep factor tree for string AGACAGGAGGC$.

factor tree does not have any leaf with a collapsed start position, since there is no common substring of size 5 in the string S =AGACAGGAGGC$. However, if we consider $k = 3$, the substring AGG occurs twice in the string S, at positions 5 and 8, and we obtain a 3-deep factor tree with collapsed start positions, as depicted in Figure 4.2.

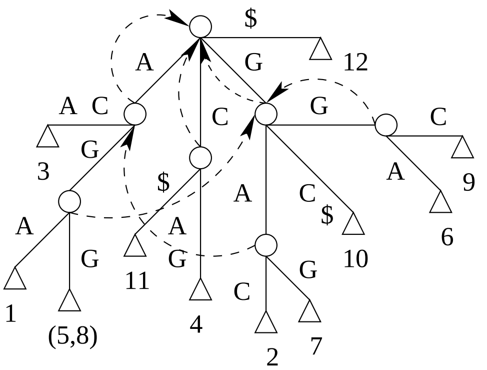

Figure 4.2: At most 3-deep factor tree for string AGACAGGAGGC$.

As for suffix trees, the time and space complexities for constructing factor trees are linear in the length of the string [AS03]. However, applications such as the extraction of single or structured motifs, where the length of the models to be searched in the suffix tree is limited, can obtain a considerable gain in terms of space and time by the use of factor trees. Compared with a suffix tree, the k-factor tree offers a substantial gain in terms of space complexity for small values of k, as well as a gain in time when used for enumerating all occurrences of a pattern in a text indexed by such a k-factor tree.

To implement the factor tree construction algorithm a new codification is used based on an *Improved Linked List Implementation*, proposed by Kurtz [Kur99] for suffix trees, called the *illi* coding. Fundamentally, the coding is changed so that it efficiently handles the fact that a leaf in the factor tree may store more than one position. In the factor tree coding, the first occurrence of a leaf added to the factor tree behaves exactly as a leaf added to the suffix tree. However, when trying to insert an already added leaf to the factor tree, a code of the leaf is also stored but now as a new occurrence of the first one. This new occurrence simply adds a new position to the already existing leaf (c.f. positions 5 and 8 in Figure 4.2). Whenever a new position is added to an already existing leaf we say that the leaf is being updated.

In Section 4.2.1, we are going to use $list_{leaf}$ to store the leaves of the factor tree as they are inserted or updated (recall Figure 4.2 and note that the $list_{leaf}$ of the depicted factor tree is $\{1,2,3,4,(5,8),6,7,(5,8),9,10,11,12\}$). It is obvious that the length of the $list_{leaf}$ is the length of the input sequence.

4.2 Structured motif extraction algorithm

In the next section we introduce the main contribution of this thesis, the box-link data structure. We will sketch, in a few strokes, the core ideas of the concept, hoping that it will convey a general idea of its usage. Its construction from the input sequences is straightforward and we shall examine an algorithm for that purpose quite carefully in Section 4.2.2. Then, in Section 4.2.3, we present the algorithm to extract structured motifs using the new data structure. Finally, in Section 4.2.4 and Section 4.2.5, we introduce some extensions to the proposed algorithm and explain how the extracted models are trimmed by statistical significance in order to deal with the enormous number of false positives.

4.2.1 Box-link data structure

A box-link stores the information needed to jump from box to box in a structured model. Its name comes from the fact that it links all p boxes of a possibly valid structured model. There are two main advantages in the use of this data structure. First and foremost, the information needed to jump from box to box when searching for structured models is memorized and can be quickly accessed. Naturally, we pay something with the use of box-links, but as we shall see, both theoretical and experimental analysis show that they accomplish a substantial improvement. Second, we capitalize on the use of factor trees, since there is no need to compute the suffix tree below the maximum size of the boxes of the structured models being extracted.

Informally, a box-link is a tuple of tree nodes, corresponding to jumps on the factor tree from box to box in a structured model. To illustrate the general idea behind box-links, suppose we have the input sequence AAACCCCCGGGGGT and we are extracting structured models with $p = 3$ boxes of the same size $k = 3$, and the same distance $d = 2$ between them. Under these conditions, there are only two box-links for the given input sequence, since there are at most two structured models. Box-links are illustrated in Figure 4.3. Note that only a

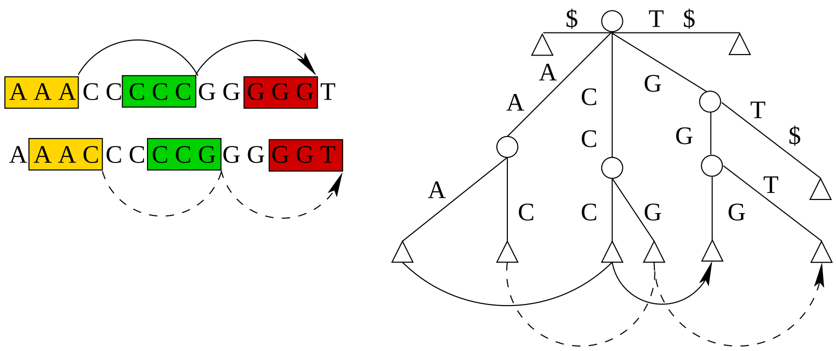

Figure 4.3: A general idea of box-links.

3-deep factor tree is needed to work out this problem.

For the sake of simplicity we assume that all p boxes of a structured model are of the same size k with distances between them ranging over the interval $[d_{min}, d_{max}]$. Formally, a box-link can be defined as follows:

Definition 4.2.1 Let L_k be the set of leaves at depth k and L_k^i denote all possible i-tuples over L_k. A *box-link of size* i, denoted simply by B_l, is a $(i+1)$-tuple in L_k^{i+1}, with $1 \leq i \leq p-1$.

The key idea towards the usage of a box-link data structure is that there is a box-link from a leaf l_1 to a leaf l_2, if when jumping down the suffix tree from l_1 at depth k (Figure 4.4a), and following the unique path of suffix-links from all children of l_1 at depth $2k+d$, we reach l_2 again at level k (Figure 4.4b). Clearly, there might be several leaves l_2 fulfilling the previous condition. In fact, Figure 4.4 depicts the differences between Algorithm 3.3 presented in

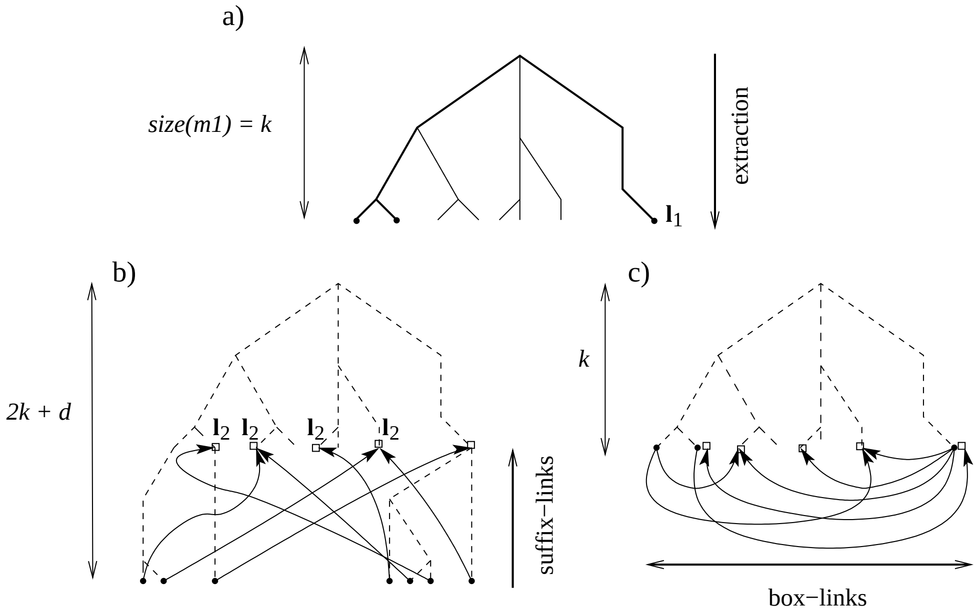

Figure 4.4: Extracting structured motifs following b) suffix-links and c) box-links.

Section 3.3.2, to extract structured motifs using suffix-links (Figure 4.4b), and the one using box-links (Figure 4.4c). The information taken from level $2k + d$ to level k of the suffix tree is the *Colors* of the nodes reached at depth $2k + d$. To store all this information we have to endow box-links with a Boolean array, similar to the one associated with tree nodes, defined as:

$$Colors_{B_l}[i] = \begin{cases} 1 & \text{if } B_l \text{ links boxes of the } i\text{-th input sequence} \\ 0 & \text{otherwise} \end{cases} .$$

4.2.2 Box-link construction

Herein we present an algorithm to build box-links which takes into account that some box-links may collapse from some box on, by placing them in a same equivalence class, leading to a time complexity that is negligible when compared to the time spent extracting structured motifs. Moreover, space requirements do not represent a problem, as we shall see.

Consider once again the case where we have all p boxes of the same size k with distances between them over the interval $[d_{min}, d_{max}]$. In a naive approach to build box-links we would create all possible p-tuples, corresponding to all possible box-links, for each position in the input sequences. However, and just for a single position of the input sequences, the number of box-links would explode exponentially in p, as roughly illustrated in Figure 4.5. Fortunately,

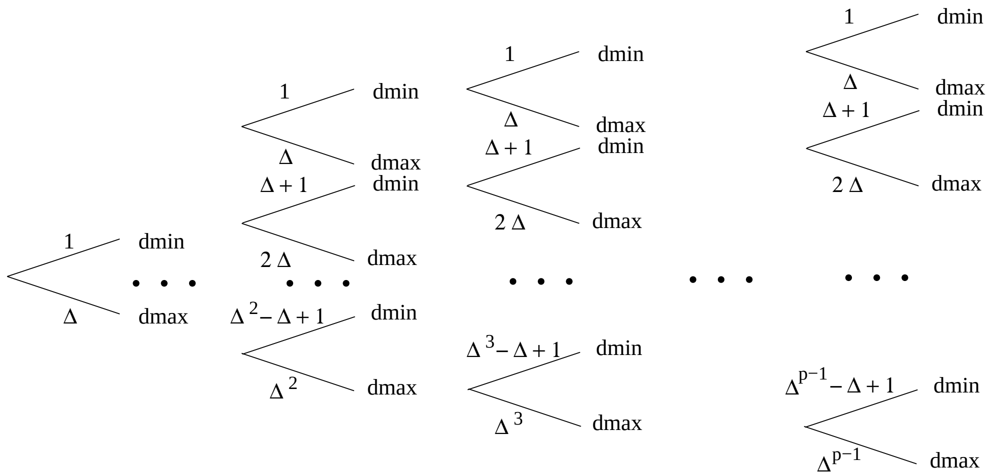

Figure 4.5: Brute force construction of box-links starting at the same position.

it is possible to reduce the number of box-links being constructed by taking into account the simple observation that after some point on, the information stored in different box-links is identical, and box-links become equivalent from that point on. It should be clear (see Figure 4.6 for an illustration) that a box-link that links boxes distanced by d_{min}, from first to second box, and $d_{min} + 1$, from second to third box, is equivalent, from the third box on, to a box-link that links boxes distanced by $d_{min} + 1$, from first to second box, and d_{min}, from second to third box. In fact, we consider that both box-links collapse in a same box-link from

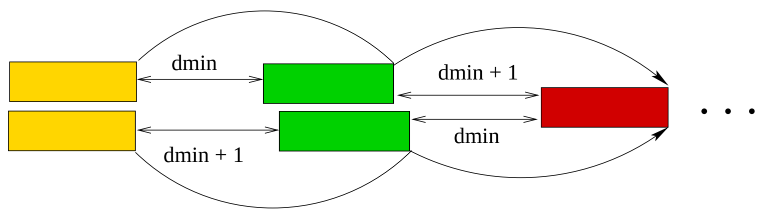

Figure 4.6: Merge of equivalent box-links from the third box on.

the third until the last box. The algorithm we propose takes this observation into account by placing box-links in equivalence classes. The gain obtained with such merging operations, and making an analogy with Figure 4.5, can be presented as depicted in Figure 4.7.

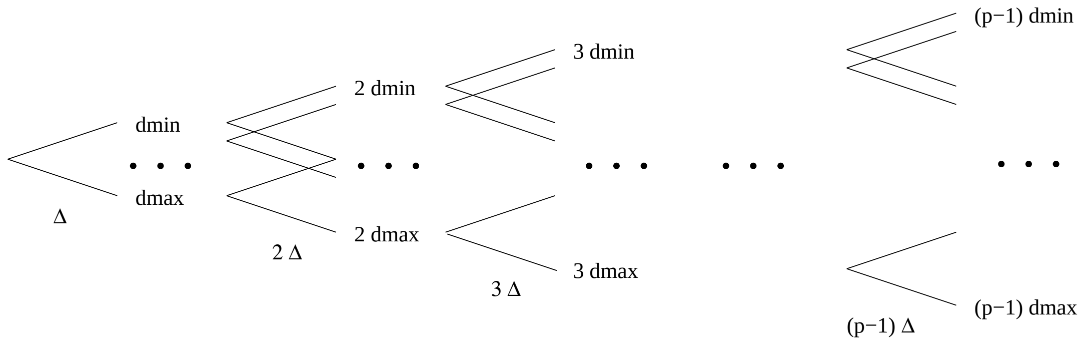

Figure 4.7: Equivalent merge construction of box-links starting at the same position.

In order to define the algorithm to construct box-links, presented in Algorithm 4.1, we have to make use of two variables. First, the variable $list_{leaf}$ has the list of all leaves as they were inserted or updated in the factor tree, which can be easily obtained during the factor tree construction as explained in Section 4.1.1. In fact, for the sake of exposition, $list_{leaf}$ can be seen as a family of variables $(list_{leaf_i})_{1 \leq i \leq N}$ (one for each input sequence), where each $list_{leaf_i}$ has length n (the average length of an input sequence). Observe that the factor labelling the path from the root to the j-th leaf of $list_{leaf_i}$ corresponds to the j-th at most k-length factor of the i-th input string. Second, the variable $[bl_j]_g$ represents the equivalence class of j-size box-links that have the sum of all j distances between boxes equal to g. Moreover, we have to define the function AddBoxLink. AddBoxLink(b,v,i) adds a box-link between an existing $(j-1)$-size box-link b and a leaf v for the i-th input sequence. However, it only creates a new box-link if there is not already a box-link between the box-link b and the node v (merging in this way equivalent box-links). In either way, creating or not a new box-link, the AddBoxLink function sets the Boolean array entry i to 1.

A proper appreciation of Algorithm 4.1 requires some explanation. The first step runs over all input sequences, while the second step runs over all positions of an input sequence, guaranteeing that there are box-links to build from that position on. Step 3 builds a dummy box-link to start the construction process and it could be seen as a 0-size box-link, that is, a 1-tuple which contains only the starting leaf. The fourth step runs over all $p-1$ distances between boxes of the structured model. Observe that to build all $(p-1)$-size box-links we need to build all i-size box-links, with $1 \leq i < p-1$. The fifth step runs over all possible

Algorithm 4.1 BoxLink, box-link construction

BoxLink(boxes p, box size k, box distance d, list of leaves $list_{leaf}$)

1. for (i from 1 to N)
2.   while (size of $list_{leaf_i} \geq pk + (p-1)d_{min}$)
3. $[bl_0]_0 = \text{AddBoxLink}(nil, list_{leaf_i}[0], i)$
4.   for (j from 1 to $p-1$)
5.   for (g from $(j-1)d_{min}$ to $(j-1)d_{max}$)
6.   for (h from d_{min} to d_{max})
7.   if (size of $list_{leaf_i} \geq pk + (g + h + (p-j-1)d_{min}))$
8. $[bl_j]_{g+h} += \text{AddBoxLink}([bl_{j-1}]_g, list_{leaf_i}[jk + g + h], i)$
9.   remove the first leaf of $list_{leaf_i}$

source positions for the next link to build. Notice that, when $j = 1$ there is only one source position, while when $j > 1$ there are several. See Figure 4.8 for a deeper insight. Step 6

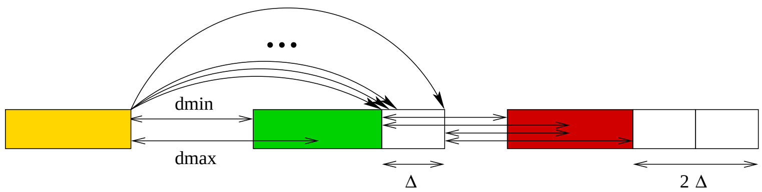

Figure 4.8: Box-links for boxes with a fixed size at variable distances.

runs over the possible distances to the target positions of the link to build. The seventh step guarantees the existence of such target and step 8 builds the box-link itself based on the previous data. The last step advances the algorithm one position in the input sequence.

Next, we establish the time and space complexity for Algorithm 4.1.

Proposition 4.2.2 Algorithm 4.1 takes $O(N^2 n \Delta^2 p^2)$ time.

Proof: The time complexity of Algorithm 4.1 is determined by steps 1, 2, 4, 5, 6 and 8. Step 1 requires $O(N)$ time. Step 2 requires $O(n)$, where n is the average number of leaves in $list_{leaf_i}$ (the average length of an input sequence). From step 4 to step 6 the time complexity of the box-link construction is given by

$$\sum_{j=1}^{p-1} \sum_{g=(j-1)d_{min}}^{(j-1)d_{max}} \sum_{h=d_{min}}^{d_{max}} O(1) = O(\Delta^2 p^2).$$

Step 8 requires $O(N)$ time, which corresponds to the creation and/or updating of the array *Colors* for the box-link being added. We conclude that Algorithm 4.1 takes $O(N^2 n \Delta^2 p^2)$ time. $\square$

Note that when $\Delta = 1$, the previous complexity becomes $O(N^2 np^2)$. However, building $(p-1)$-size box-links corresponds to linking p boxes, each one taking $O(N)$, for all n positions of an input sequence, and for all N input sequences, leading only to a $(N^2 np)$ time complexity. In fact, the extra p of the first complexity arises because we are counting the effect of distance variation on Step 5 of the algorithm, which does not exist when $\Delta = 1$. We conclude that when $\Delta = 1$ Algorithm 4.1 takes $O(N^2 np)$ time.

Proposition 4.2.3 Algorithm 4.1 takes $O(Nb_p(k, d_{max}))$ space, where an upper bound for the total number of $(p-1)$-size box-links is defined as $b_p(k, d_{max}) = \min\{n_k^p, p\Delta^2 n_{pk+(p-1)d_{max}}\}$.

Proof: By definition there are at most $b_p(k, d_{max})$ $(p-1)$-size box-links and each box-link stores the array *Colors*, which takes $O(N)$ space. We conclude that Algorithm 4.1 requires $O(Nb_p(k, d_{max}))$ space. $\square$

To compare complexity results for time and space, it is important to notice that $b_p(k, d_{max}) = \min\{n_k^p, p\Delta^2 n_{pk+(p-1)d_{max}}\} \leq p\Delta^2 Nn$. Hence, the space complexity of Algorithm 4.1 can also be given by $O(N^2 n\Delta^2 p)$, which is less than its time complexity, as expected.

Note that when $\Delta = 1$, Algorithm 4.1 takes $O(Nb_p(k, d))$ space, where the total number of $(p-1)$-size box-links can be upper bounded by $b_p(k, d) = \min\{n_k^p, n_{pk+(p-1)d}\}$.

4.2.3 Jumping in the factor tree using box-links

In this section, we describe the algorithm to extract structured motifs, using box-links as a fundamental data structure. We also present its complexity analysis with an exponential time gain in the worst case scenario. This improvement is attained because the time complexity does not depend on the distances between the boxes of the structured motif.

Suppose we have all p boxes of the same size k with distances between them over the interval $[d_{min}, d_{max}]$. The ExtractMotifs algorithm using box-links is very similar to the one proposed with the use of suffix-links [MS00]. At first, a factor tree $\mathcal{T}$ is built, up to the level k, for all input sequences. The factor tree is then modified to store at each node the $Colors$ array, and box-links are added to the leaves of the factor tree. After this pre-processing phase the extraction begins. The pseudo-code for the extraction is presented in Algorithm 4.2.

Algorithm 4.2 ExtractMotifs, structured motif extraction using box-links

ExtractMotifs(model m, box i)
for each node-occurrence v of m
for each box-link $B_l(v, z)$
put z in $L(i)$
if (first time z is reached)
$Colors_z = \overrightarrow{0}$
put z in $NextEnds$
$Colors_z = Colors_z + Colors_{B_l(v,z)}$
UpdateTree($\mathcal{T}$,$NextEnds$)
for each motif m_i obtained by SpellMotifs traversing $\mathcal{T}$
if $(i < p)$ ExtractMotifs($m = m_1 \ldots m_i$,$i + 1$)
else KeepMotif($m = \langle(m_1, \ldots, m_p),(d_{min}, d_{max})\rangle$)
RestoreTree($\mathcal{T}$,$L(i)$)

The ExtractMotifs algorithm makes use of three functions. First, UpdateTree updates the Boolean arrays from the nodes in $NextEnds$ to the root in the following way: if nodes $\overline{z}$ and $\widehat{z}$ have the same parent z, then $Colors_z = Colors_{\overline{z}} + Colors_{\widehat{z}}$. Any arc from the root that does not have a node in $NextEnds$ is not part of the updated tree, nor are the subtrees rooted at its node in $NextEnds$. Second, RestoreTree restores the Boolean arrays from the nodes in $L(i)$ to the root in the following way: if nodes $\overline{z}$ and $\widehat{z}$ have the same parent z, then $Colors_z = Colors_{\overline{z}} + Colors_{\widehat{z}}$. Any arc from the root is part of the restored tree. Third, KeepMotif stores all information concerning valid motifs.

The correctness of the Algorithm 4.2 follows straightforwardly from the correctness of

Algorithm 3.3, presented in the original paper by Marsan and Sagot [MS00], since box-links mimic the behaviour of jumping down the tree and following up the suffix-links.

Complexity analysis for constant distance between boxes

Next, we establish the time and space complexity for the algorithm at hand. We only consider the case when $d_{max} = d_{min} = d$, and as before we take an upper bound for the total number of $(p-1)$-size box-links defined by $b_p(k,d) = \min\{n_k^p, n_{pk+(p-1)d}\}$.

Proposition 4.2.4 Algorithm 4.2 takes $O(Ns_p(k,d)\nu^p(e,k) + Nb_p(k,d)\nu^{p-1}(e,k))$ time.

Proof: Given the similarity with the ExtractMotifs algorithm presented in Section 3.3.2, to compute the complexity of Algorithm 4.2 we just have to compute the total number of operations needed to update $\mathcal{T}$ (all other parcels are similar to Algorithm 3.3).

The total number of operations needed to modify the first k levels of the suffix tree $\mathcal{T}$, using box-links, is now upper bounded by $Nb_p(k,d)\nu^{p-1}(e,k)$. This corresponds to all visits made to nodes z coming from $B_l(v,z)$ for all models m_{p-1}. In addition, the propagation from z to the root R for all models m_{p-1} is upper bounded by the same value.

We conclude that Algorithm 4.2 takes

$$O(Ns_p(k,d)\nu^p(e,k) + Nb_p(k,d)\nu^{p-1}(e,k))$$

time. $\square$

This algorithm exhibits an exponential time gain, in the worst case analysis, relatively to the previous algorithms to extract structured motifs presented in Section 3.3. The only difference between complexity expressions appears in the second parcel, concerning the update and restoration of the suffix or factor tree, where the new algorithm presents the term $b_p(k,d)$ whereas the previous algorithm presents the term $n_{pk+(p-1)d}$. Observe that in the worst case scenario, the suffix or factor tree is complete and we have $b_p(k,d) = \min\{|\Sigma|^{pk}, |\Sigma|^{pk+(p-1)d}\} = |\Sigma|^{pk} < n_{pk+(p-1)d} = |\Sigma|^{pk+(p-1)d}$ which reflects an exponential gain of the order $|\Sigma|^{(p-1)d}$. The major gain of this new method, over previous approaches for extracting structured motifs, is that in the worst case scenario the extraction time of the motifs remains independent of the distances between them.

Moreover, it is important to notice that time spent building box-links, presented in Section 4.2.2, is negligible in comparison with time spent for the extraction.

Proposition 4.2.5 Algorithm 4.2 takes $O(Nb_p(k,d) + Npn_k)$ space.

Proof: To compute the space complexity of the Algorithm 4.2 note that the factor tree takes $O(Nn_k)$ space, the box-links take $O(Nb_p(k,d))$ space and restoring the tree $\mathcal{T}$ uses $O(N(p-1)n_k) = O(Npn_k)$ additional space, which is the size of the $L(i)$, with $1 < i \leq p$, each cell possibly pointing to a node at level k in $\mathcal{T}$ or to nil. Since $n_k \leq b_p(k,d)$, we conclude that the total space complexity is $O(Nb_p(k,d) + Npn_k)$. □

This algorithm exhibits an exponential space gain, in the worst case analysis, relatively to the previous algorithms to extract structured motifs presented in Section 3.3. To compare space complexity results of Algorithm 4.2 with Algorithm 3.2 and Algorithm 3.3, we have to rewrite the space complexity of the later two considering that they used a $(pk + (p-1)d)$-deep factor tree. In this case, the space complexity of Algorithm 3.2 becomes $O(Nn_{pk+(p-1)d})$ whereas the space complexity of Algorithm 3.3 becomes $O(Nn_{pk+(p-1)d} + Npn_k)$. Observe that in the worst case scenario, the factor trees are complete and we have

$$\begin{aligned}
b_p(k,d) + pn_k &= \min\{|\Sigma|^{pk}, |\Sigma|^{pk+(p-1)d}\} + p|\Sigma|^k \\
&= |\Sigma|^{pk} + p|\Sigma|^k \qquad \text{(which is } O(|\Sigma|^{pk}) \text{ when } |\Sigma| > 1) \\
&< |\Sigma|^{pk+(p-1)d} \\
&= n_{pk+(p-1)d}
\end{aligned}$$

which reflects an exponential gain of the order $|\Sigma|^{(p-1)d}$.

Complexity analysis for variable distance between boxes

Herein we discuss time and space complexity for Algorithm 4.2, considering the general case where $d_{min} \leq d_{max}$. In this case, both complexities analysis for time and space remain the same, but the upper bound for the total number of $(p-1)$-size box-links becomes defined by $b_p(k, d_{max}) = \min\{n_k^p, p\Delta^2 n_{pk+(p-1)d_{max}}\}$. Moreover, it is worthwhile to notice that the time and space exponential gains are also verified, since in the worst case $b_p(k, d_{max}) = n_k^p = |\Sigma|^{pk}$.

4.2.4 Extending the algorithm

All extensions presented in Section 3.3.3, to enhance the algorithms for extracting structured motifs, are easily translated into the algorithm based on box-links introduced in this chapter.

The only exception is the problem of handling boxes with variable length, which is briefly discussed in the following.

In order to deal with boxes of variable size one needs to slightly modify the notion of box-link. In fact, when dealing with boxes of variable length, the label of a tree arc between two nodes may have more than one symbol, since the factor tree is a compact tree, and so it may represent more that one box (of different sizes). This property of factor trees requires box-links to be endowed with some extra structure when dealing with boxes of variable length. A straightforward way to deal with this problem is to store information about the size of the boxes the box-link concerns aside from the input sequence where it occurs. This can be easily achieved by promoting the Boolean array $Colors$, of size N, to a Boolean matrix, of size $\Gamma \times N$, where $\Gamma = k_{max} - k_{min} + 1$.

The algorithm to build box-links requires few changes when we wish to consider structured motifs composed of boxes with different sizes. We only need to add two loops similar to the ones in steps 5 and 6 of the Algorithm 4.1, but now accounting for variations on the size of the boxes. Moreover, few changes are required to Algorithm 4.2 to extract structured motifs using box-links. The main one comes from the fact that the operations of following box-links to find a box i, once boxes 1 to $i-1$ have been found, is done for all possible allowed lengths for the previous boxes, as already proposed by Marsan and Sagot [MS00].

4.2.5 Measuring statistical significance

We do not address in this thesis the question of evaluating the statistical significance of the structured motifs found. Work on performing such evaluation for two boxes exists [RDR$^+$02]. However, this remains an open problem for more than two boxes and for more general statistical models. We used for such evaluation the same procedure as in previous algorithms [VMS99, VMLS00]. This produces reliable results but is not fast. However, it can be applied to datasets which most other methods, including well-known MEME, can not easily deal with.

4.3 Experimental results

In this section, we present preliminary results obtained using a prototype implementation of the algorithms described in this paper. The structured motif extraction original paper

[MS00] introduced two algorithms. A C implementation of the first algorithm, presented in Section 3.3.1, was made available by the authors (SMILEv1.45). For the second algorithm (SMILEv2), presented in Section 3.3.2, we used a C/C++ implementation. The new algorithm proposed in this paper was implemented on top of the SMILEv2 algorithm augmented with a new data structure to code box-links (RISO).

As test sets we used two groups of sequences. The first group corresponds to a collection of 68 genes that are known to be regulated by zinc cluster factors [vHRCV00]. The upstream sequences were retrieved from positions -1 to -1000 relative to the ORF (open-reading frame) start positions. We tested two models for the analysis of this data. A first model with two boxes with length 3 and distance 11 and a second one with two boxes with length 4 and distance 9. For the first model the algorithm detected, with a 10% quorum, the consensus $CGGn_{11}CCG$ that corresponds to the nucleotides that enter into direct contact with Gal4p. For the second model the consensus $CGGAn_9TCCG$ was detected with highest significance. These consensi, that were classified as statistically significant, are also biologically relevant. The results obtained for the three algorithms are presented in Table 4.1.

# Errors		CPU Times (in seconds)			# Extracted
Box 1	Box 2	SMILEv1.45	SMILEv2	RISO	models
1	1	44.72	7.4	0.12	4096
2	2	1612.68	60.71	12.12	65536

Table 4.1: Extraction of $CGGn_{11}CCG$ and $CGGAn_9TCCG$ from the first dataset.

The second group is composed of three sets of non-coding sequences located between two divergent genes and extracted from the whole genomes of *B. subtilis*, *E. coli* and *H. pylori*. The first of these sets contains 1,062 sequences for a total of 196,736 nucleotides. The second set contains 1,148 sequences and 226,928 nucleotides. The last set contains 308 sequences and 52,100 nucleotides. These three datasets were originally used as test cases for extracting promoter consensi and to test the sequential algorithm [MS00]. For these datasets we tested a model with two boxes of size 6 and spacing 17. For the *E. coli* organism, the algorithms were able to detect the consensus $TTGACAn_{17}TATAAT$ given in the literature for the σ^{70} promoter sequence. The results obtained for the three algorithms are presented in Table 4.2.

Besides biological significant data, we also compared SMILEv1.45 and RISO with synthetic

# Errors		CPU Times (in seconds)			# Extracted
Box 1	Box 2	SMILEv1.45	SMILEv2	RISO	models
1	2	1429.81	1983.41	942.42	11147160

Table 4.2: Extraction of $TTGACAn_{17}TATAAT$ from the *E. coli* dataset.

data. The results obtained revealed a similar speedup to the ones presented in Table 4.1 and Table 4.2. Just for an illustration purpose, an experiment for three boxes of length 4 separated by the intervals of distances 5..7 and 15..23 is presented in Table 4.3.

# Errors			CPU Times (in seconds)		# Extracted
Box 1	Box 2	Box 3	SMILEv1.45	RISO	models
1	1	2	2141.09	22.86	262131

Table 4.3: Extraction of structured motifs with three boxes from synthetic data.

By analyzing these results, together with others omitted here, we reached some conclusions about the three algorithms. On one hand, SMILEv2 and RISO perform better when the tree is dense, or when the first box of the structured model being extracted is small. In fact, in our experiments, we get to the conclusion that there is a level in the suffix or factor tree from which the tree becomes very sparse. If the size of the first box is smaller than this level, SMILEv2 and RISO achieve much better results than SMILEv1.45. Moreover, if the tree is dense, this level is deeper and we are closer to the worst case scenario where RISO presents an exponential gain over SMILEv2, which by itself presents an exponential gain over SMILEv1.45. This is the case illustrated in Table 4.1 and Table 4.3, where the tree becomes sparser at level 8, and we extracted boxes of length 3 and 4. On the other hand, SMILEv1.45 gets better results when we are extracting a structured model with a large first box, or when the input data sequences are poor, as illustrated in Table 4.2. However, in practice, cases where RISO achieves better results are much more relevant because the input data is richer and the problem is harder.

As concerns memory, all algorithms performed similarly in our experiments, using about 20MB. Box-links did not cause memory problems and in cases where boxes were small the total memory usage of the factor tree with box-links was less than the memory required by the full suffix tree.

Chapter 5

Parallel extraction of structured motifs

This chapter presents a general method concerning the parallelization of algorithms where the search space is represented by a tree. That is the case of the algorithms, presented in Chapter 3 and Chapter 4, for structured motif extraction. Work-efficiency of the parallel version is only achieved if the extraction is balanced over the available processing units. Therefore, the main issue of the parallelization is to define a balanced partition of the structured motif searching space. This space can be represented by the lexicographic trie $\mathcal{M}$ (Section 3.2) and, in some special cases, by the suffix or factor tree of the input sequences.

Finding a balanced partition of a tree, and in particular of a suffix or factor tree, is an issue of the utmost importance. This follows from the fact that many search algorithms are based on suffix trees and a parallel balanced approach can help solve the problem of an increasing amount of data coming from the Bioinformatics community. A considerable effort on the development of algorithms for building suffix trees using distributed approaches has already been done [Har97, AIL$^+$88]. However, the main goal of these algorithms is to reduce the time complexity of the suffix tree construction, and this step is by far the fastest of the sequential algorithms to extract single or structured motifs. Other approaches have been taken to solve the problem of suffix tree construction for a whole genome [SS02, HAI01]. The purpose of these approaches is to build suffix trees on disk in order to amortize the construction cost over many thousand searches. This question is addressed with an algorithm to build consecutive partitions of the suffix tree, where the partition size depends on the available memory, and compose them together on disk to proceed with searches. Once again, this is not the goal of the parallel extraction of structured motifs. Since the extraction step of the structured motifs algorithm is the most time consuming, the main goal of the parallelization is to divide the tree search space among the available processing units and to proceed in each one with a separate extraction. Moreover, we want a balanced partition of the tree search space over all the available processors that is not necessarily imposed by memory constraints.

In our approach, we abstract the problem of finding a balanced partition of a tree to a generalization of the 3-PARTITION problem [GJ79], which we call PARTITION UP TO ε problem. We show that this new problem is NP-complete in the strong sense and can not be straightforwardly reduced to 3-PARTITION. This indicates that the PARTITION UP TO ε problem is interesting *per se*. Furthermore, unlike 3-PARTITION, the PARTITION UP TO ε problem has an optimization version for which we propose an approximation algorithm. This approximation algorithm is applied to obtain a partition of the search tree space among the available processing units and it is the core of the proposed parallelization.

The parallel version of the algorithm was implemented using grid technology. Nevertheless, it is worthwhile to notice that the algorithm is designed in the CREW-PRAM (concurrent read, exclusive write, parallel random access machine) computational model [CLR90] and therefore any realization of this model will support the algorithm. Moreover, our solution does not require the grid nodes to communicate, which boosts the overall performance. We obtain as a simple corollary that our algorithm can be made work-efficient, with respect to the sequential algorithm, under easily achievable conditions.

In Section 5.1 we establish the PARTITION UP TO ε problem and present an algorithm to approximate the optimization version of this problem. In Section 5.2 we apply this algorithm to the practical case of parallel structured motif extraction and we present some experimental results in Section 4.3.

5.1 Balanced partition

The problem of determining a balanced partition of the structured motif extraction search space can be abstracted to the following general problem.

PARTITION UP TO ε problem: Suppose we are given a set of ℓ gold bars, where the weight of the j-th gold bar is a non negative integer w_j. Additionally, assume that we can cut any gold bar with weight w in c equal parts, obtaining in this way c new gold bars with weight $\frac{w}{c}$. Note that each of the resulting gold bars can then be cut again in c equal parts and that this process can proceed successively.

- **Optimization version**: The problem is how to share the gold between r persons, with the minimum number of gold bars z, in such a way that each person gets the same share of gold up to some weight $\varepsilon > 0$.

- **Decision version**: The problem is to decide whether it is possible to share the gold between r persons, ending up with z gold bars, in such a way that each person gets the same share of gold up to some weight $\varepsilon \geq 0$.

Before presenting an approximation algorithm to solve the optimization version of this problem we show that the decision version is NP-complete in the strong sense.

Theorem 5.1.1 The PARTITION UP TO ε problem is NP-complete in the strong sense.

Proof: To show that this problem is in NP, take as witness space the set of all r-partitions made with z gold bars. Consider as the polynomial time verifier the program that checks whether a witness is a partition up to ε. Then, if there is one partition up to ε, with z gold bars, take it as witness. Moreover, if there is no partition up to ε, with z gold bars, then there is no witness for which the polynomial time verifier will answer yes. Finally, to prove the strong NP-completeness we transform the 3-PARTITION problem [GJ79], which is NP-complete in the strong sense, to the PARTITION UP TO ε problem. Consider the set $A = \{a_1, \ldots, a_{3m}\}$ as an arbitrary instance of the 3-PARTITION and make an instance of the PARTITION UP TO ε where $r = m$, $\ell = 3m$, $w_j = a_j$ with $1 \leq i \leq 3m$, $\varepsilon = 0$, $z = 3m$ and $c = 1$. This transformation can clearly be performed in time polynomial in the input length alone. Furthermore, the length and the largest number of the constructed instance remain the same as in the given 3-PARTITION instance. Finally, there is a solution for the given 3-PARTITION problem instance iff there is a solution for the obtained instance of the PARTITION UP TO ε problem. Moreover, the transformation is a pseudo-polynomial transformation and so the PARTITION UP TO ε problem is strongly NP-complete. $\square$

Note that the previous result about strong NP-completeness guarantees that the PARTITION UP TO ε problem cannot be solved by a pseudo-polynomial time algorithm, unless P=NP. Next we propose an approximation algorithm to solve this problem. The main goal is to find an optimal balanced r-partition of the gold bars up to some positive weight ε. Since there is no hope to find an efficient optimal solution we consider a trade-off between optimal solution presentation and optimal number of cuts. In our algorithm the balanced partition is defined as a family of intervals $(I_i)_{1\leq i\leq r}$. This feature has the main advantage that it is straightforward to check whether a gold bar belongs to a certain person. On the other hand, our solution cuts successively all gold bars, which is not necessarily optimal. The algorithm to determine the i-th partition set I_i is presented in Algorithm 5.1.

Algorithm 5.1 SimpleCut, gold bar share among persons

SimpleCut(partition set i, gold bars ℓ, persons r, weights $(w_j)_{1\leq j\leq \ell}$, cut factor c, overload ε)

1. find the smallest t such that $\frac{1+2\max\{w_j\ :\ 1\leq j\leq \ell\}}{c^t} \leq \varepsilon$
2. for each $j \in \{1, ..., \ell\}$
3. let $V_j = \left[\sum_{k=1}^{j-1} w_k \times c^t, \sum_{k=1}^{j} w_k \times c^t\right)$
4. let $w = \sum_{j=1}^{\ell} w_j$
5. let $\gamma = w \times c^t \mod r$
6. let $\delta = \left\lfloor \frac{w\times c^t}{r} \right\rfloor$
7. let $I_i' = \begin{cases} [(i-1)(\delta+1),\ i(\delta+1)\,) & \text{for all } i \leq \gamma \\ [\gamma(\delta+1)+(i-(\gamma+1))\delta,\ \gamma(\delta+1)+(i-\gamma)\delta\,) & \text{otherwise} \end{cases}$
8. transform $I_i' = [a, b)$ into $I_i = [f(a), f(b))$ with $f : w\times c^t \to \ell \times c^t$ defined as

$$f(x) = \begin{cases} (j-1)\times c^t + \left\lfloor \frac{x-\inf(V_j)}{w_j} \right\rfloor & \text{for all } x \in V_j \\ \ell \times c^t & \text{if } x = w\times c^t \end{cases}$$

The first step of the algorithm finds the number t of times each gold bar is successively cut such that each one weights less than $\frac{\varepsilon}{2}$. Note that the total number of cuts obtained is $\sum_{i=1}^{t} \ell \times c^{i-1}$. At this point we have

$$z = \ell \times c^t \tag{5.1}$$

gold bars and we represent the set of these final gold bars by the interval of natural numbers $[0, \ell \times c^t)$. Hence, the original j-th gold bar lies on the interval $[(j-1) \times c^t, j \times c^t)$ and each final gold bar in this interval weights $\frac{w_j}{c^t} \leq \frac{\varepsilon}{2} - \frac{1}{2c^t}$. These final gold bars are enough to define a balanced r-partition up to the weight ε. However, it is not straightforward to define the partition since the final gold bars weight differently. A way out of this problem is to create a set of virtual gold bars where all virtual gold bars weight the same. To achieve this we divide the original j-th gold bar interval $[(j-1) \times c^t, j \times c^t)$ by w_j and obtain a new j-th virtual gold bar interval V_j where each virtual gold bar weights $\frac{1}{c^t}$. Note that we have $w \times c^t$ virtual gold bars and that the set of these virtual gold bars can be represented by the interval of natural numbers $[0, w \times c^t)$. The computation of the intervals $V_j \subseteq [0, w \times c^t)$ is exactly what we do on the second step of the algorithm.

From step 3 to 6 we define a balanced r-partition set I'_i up to weight ε over the virtual gold bars set $[0, w \times c^t)$. This is straightforward to do since all virtual gold bars weight the same. Finally, in step 7 we map the partition set I'_i over the virtual gold bars set $[0, w \times c^t)$ into a partition set I_i over the original gold bars set $[0, \ell \times c^t)$.

In order to check that the weight of the intervals obtained for the balanced partition differ at most ε, we next establish the correctness of Algorithm 5.1.

Proposition 5.1.2 Algorithm 5.1 is correct.

Proof: In order to show that the algorithm is correct we consider the worst possible imbalance. This happens when we obtain, in step 7, intervals I'_i with weight $\frac{\delta+1}{c^t}$ and $I'_{i'}$ with weight $\frac{\delta}{c^t}$. Due to the truncation made in Step 8, namely by the floor operation on the definition of f, it is possible that the weight of the obtained interval I_i is augmented at most to $\frac{\delta+1}{c^t} + \frac{\max\{w_j \,:\, 1 \leq j \leq \ell\}}{c^t}$ while the weight of $I_{i'}$ is reduced at most to $\frac{\delta}{c^t} - \frac{\max\{w_j \,:\, 1 \leq j \leq \ell\}}{c^t}$. This happens because the maximum weight of each final gold bar is $\frac{\max\{w_j \,:\, 1 \leq j \leq \ell\}}{c^t}$ and the truncation can only add or remove at most the weight of one final gold bar to the weight of the respective interval I'. Hence, the difference between the weights of these two intervals is $\frac{1+2\max\{w_j \,:\, 1 \leq j \leq \ell\}}{c^t}$ which we know to be less than or equal to ε by Step 1. $\square$

Next we establish the time complexity for the algorithm at hand, but first notice that the value t can be computed as

$$t = \left\lceil \frac{\log(1+2\max\{w_j \,:\, 1 \leq j \leq \ell\}) + \log\left(\frac{1}{\varepsilon}\right)}{\log(c)} \right\rceil. \tag{5.2}$$

Proposition 5.1.3 Algorithm 5.1 requires $O(\ell)$ time.

Proof: In the first step of the SimpleCut algorithm the value t can be computed as given by (5.2) in constant time. In the second and third steps we determine ℓ intervals, but notice that the upper limit of one interval is the lower limit of the next, so in the end we only have to compute ℓ values, each one taking $O(1)$ time. Hence, the time complexity of these steps are $O(\ell)$. In the fourth step the ℓ summations are done in $O(\ell)$ time. In the seventh step we only have to compute the interval I'_i for the correspondent partition set i, in constant time. In the last step we have to determine to which interval V_j the variable x belongs, which takes $O(\log \ell)$ time. All the other operations of this step are $O(1)$. Hence, we can conclude that the overall time complexity of the algorithm SimpleCut is $O(\ell + \log(\ell)) = O(\ell)$. $\square$

Recall that the *ratio bound* upper bounds the ratio between the cost of the solution produced by the algorithm and the cost of the optimal solution, for all possible inputs [CLR90]. This ratio measures how close the proposed solution is from the optimal solution. Clearly, a proposed solution is optimal when its ratio bound is 1.

Proposition 5.1.4 Algorithm 5.1 has a ratio bound

$$\rho(\ell, r, (w_j)_{1\le j\le \ell}, c, \varepsilon) = O\left(\frac{\max\{w_j \ : \ 1 \le j \le \ell\}}{\varepsilon}\right).$$

Proof: Algorithm 5.1 returns a total number of gold bars given by

$$\begin{aligned}
z &= O\left(\ell \times c^{\frac{\log(1+2\max\{w_j \ : \ 1\le j\le \ell\})+\log\left(\frac{1}{\varepsilon}\right)}{\log(c)}}\right) && \text{by (5.1) and (5.2)}\\
&= O\left(\ell \times c^{\frac{\log\left(\frac{1+2\max\{w_j \ : \ 1\le j\le \ell\}}{\varepsilon}\right)}{\log(c)}}\right)\\
&= O\left(\ell \times 2^{\log\left(\frac{1+2\max\{w_j \ : \ 1\le j\le \ell\}}{\varepsilon}\right)}\right) && \text{since } c = 2^{\log(c)}\\
&= O\left(\ell \times \frac{(1+2\max\{w_j \ : \ 1\le j\le \ell\})}{\varepsilon}\right)\\
&= O\left(\frac{\ell\max\{w_j \ : \ 1\le j\le \ell\}}{\varepsilon}\right).
\end{aligned}$$

In the worst case scenario, there is no need to cut any gold bar. In this case the optimal solution returns ℓ gold bars and Algorithm 5.1 returns $O\left(\frac{\ell\max\{w_j \ : \ 1\le j\le \ell\}}{\varepsilon}\right)$ gold bars, which leads to the ratio bound $O\left(\frac{\ell\max\{w_j \ : \ 1\le j\le \ell\}}{\ell\varepsilon}\right) = O\left(\frac{\max\{w_j \ : \ 1\le j\le \ell\}}{\varepsilon}\right)$. $\square$

5.2 Parallel structured motif extraction algorithm

We apply the previous algorithm to establish a balanced partition of the lexicographic trie $\mathcal{M}$ of models. As mentioned in Section 3.2, the trie $\mathcal{M}$ is never built, but is virtually used to define a partition for the extraction of structured motifs from the suffix tree $\mathcal{T}$ of the input sequences.

In Section 5.2.1 we describe and illustrate how to use the Algorithm 5.1 to establish a balanced partition of the trie $\mathcal{M}$. In Section 5.2.2 we present the parallel algorithm that partitions the extraction of structured motifs over the suffix tree $\mathcal{T}$ capitalizing on a balanced partition over the lexicographic trie $\mathcal{M}$.

5.2.1 Tree partition

Herein we establish a balanced partition of the lexicographic trie $\mathcal{M}$ of models by taking advantage of the Algorithm 5.1 defined in the previous section. Observe that sequences with the same prefix belong to the same partition set, since the Algorithm 5.1 returns a family of intervals. To determine the prefixes of each partition set we compute the weight of each symbol of the alphabet Σ. To obtain these weights we first scan the input sequences to get the frequency of each symbol, and afterwards assign to each symbol a weight that reflects these frequencies. Our approach computes only weights of prefixes of size one. Clearly, we can compute weights of prefixes with greater sizes deriving in this way a more precise load balanced partition. However, experiments have shown that computing the weights for prefixes of size one is enough for deriving good results.

After this set up it is important to reduce the tree partition problem to the PARTITION UP TO ε problem. Note that the original gold bars are the prefixes for which we computed the weights. In our case the original gold bars correspond to the symbols of Σ since we only compute weights for prefixes of size one. Hence, we have $\ell = |\Sigma|$. Moreover, the number of cuts we can do to each gold bar is precisely $|\Sigma|$, corresponding to the spanning of the tree. Therefore, we have $c = |\Sigma|$. The weight w_j of each symbol of the alphabet is obtained by scanning the input sequences. Finally, the number of persons r matches the number of grid nodes and the allowed imbalance ε is a user parameter. At this point we have defined the inputs required to call the SimpleCut algorithm for the i-th grid node. The algorithm returns two outputs. First, the number t of cuts gives the depth $t+1$ of the tree where the partition is defined. Observe that the total number of cuts is ℓ^t. Second, the interval I_i corresponds to tree nodes at depth $t+1$ that are assigned to the i-th grid node.

Before adapting Algorithm 5.1 to set up a partition of the lexicographic trie $\mathcal{M}$ note that the depth of $\mathcal{M}$ is finite and upper bounded by the sum of the length of all boxes. Hence, the depth $t+1$ to which we have to descend to define the partition might be larger than the depth of $\mathcal{M}$. Therefore, we have to ensure that $t+1$ does not exceed the depth of $\mathcal{M}$. In order to fulfill this condition we replace the first step of the Algorithm 5.1 by the first two steps presented in Algorithm 5.2.

Algorithm 5.2 SimpleCut, work balance among available processing units

SimpleCut(partition set i, alphabet size ℓ, grid nodes r, weights $(w_j)_{1\leq j\leq \ell}$, alphabet size c, overload ε)

1. find the smallest t' such that $\frac{1+2\max\{w_j\ :\ 1\leq j\leq \ell\}}{c^{t'}} \leq \varepsilon$
2. let $t = \min(\mathrm{depth}(\mathcal{M}) - 1, t')$
3. for each $j \in \{1, ..., \ell\}$
4. let $V_j = \left[\sum_{k=1}^{j-1} w_k \times c^t, \sum_{k=1}^{j} w_k \times c^t\right)$
5. let $w = \sum_{j=1}^{\ell} w_j$
6. let $\gamma = w \times c^t \mod r$
7. let $\delta = \left\lfloor \frac{w\times c^t}{r} \right\rfloor$
8. let $I'_i = \begin{cases} [(i-1)(\delta+1),\ i(\delta+1)\,) & \text{for all } i \leq \gamma \\ [\gamma(\delta+1)+(i-(\gamma+1))\delta,\ \gamma(\delta+1)+(i-\gamma)\delta\,) & \text{otherwise} \end{cases}$
9. transform $I'_i = [a,b)$ into $I_i = [f(a), f(b))$ with $f : w \times c^t \to \ell \times c^t$ defined as

$$f(x) = \begin{cases} (j-1)\times c^t + \left\lfloor \frac{x-\inf(V_j)}{w_j} \right\rfloor & \text{for all } x \in V_j \\ \ell \times c^t & \text{if } x = w \times c^t \end{cases}$$

The following example illustrates the use of this algorithm. Suppose we have $r = 5$ grid nodes and $\Sigma = \{\mathrm{A,C,G,T}\}$. Moreover, assume $\sigma_1 =$A, $\sigma_2 =$C, $\sigma_3 =$G and $\sigma_4 =$T with the following weights $w_1 = 2$, $w_2 = 1$, $w_3 = 1$ and $w_4 = 2$, resulting in a total weight $w = 6$. Finally, consider $\varepsilon = \frac{3}{2}$, which means that the user allows an imbalance of $\frac{1}{4}$ of the total weight.

In the first step of the algorithm we obtain $t = 1$. The value of $t+1$ is the depth where the tree partition is made. In Figure 5.1 we can see a tree with nodes at depth $t+1$ as targets to determine a balanced distribution among the r grid nodes.

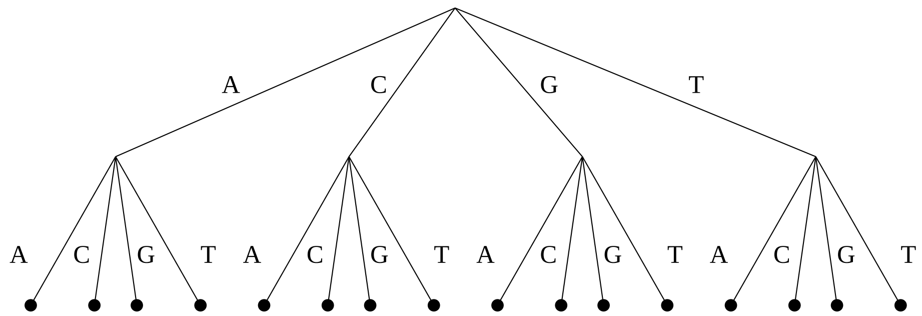

Figure 5.1: Lexicographic trie cut at depth $t+1$=2.

The second step of the algorithm computes the virtual tree node space. The computation of the intervals V_j establishes the set of virtual tree nodes with the same weight. As seen before we transform $|\Sigma| \times |\Sigma|^t = 4 \times 4^1 = 16$ tree nodes into $w \times |\Sigma|^t = 6 \times 4^1 = 24$ virtual tree nodes. Note that in this example, tree nodes are represented as filled circles and virtual tree nodes are represented as simple circles. The intervals V_j are represented in Figure 5.2.

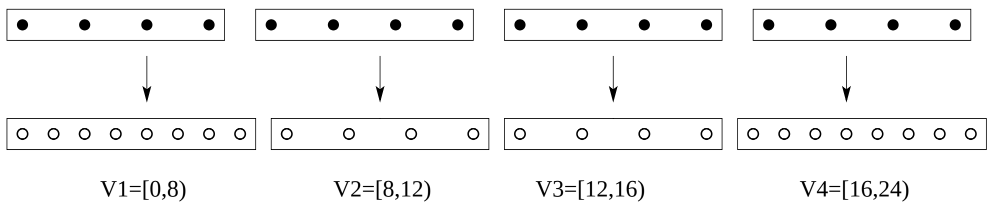

Figure 5.2: Intervals V_j in the virtual tree space.

In the next four steps, from 3 to 6, we compute the auxiliary set I'_i that corresponds to the partition set on the virtual tree node space for the i-th grid node. Observe that to determine the width of each interval I' in this virtual space we compute $\delta = (6 \times 4^1)/5 = 4$ and to determine the number of grid nodes that are going to be overloaded with at most $\varepsilon = \frac{3}{2}$ weight we compute $\gamma = (6 \times 4^1) \mod 5 = 4$. Since $\gamma = 4 > 0$ there is an imbalance among the grid nodes, and in this particular case, there are $\gamma = 4$ grid nodes with at most an overload $\varepsilon = \frac{3}{2}$ over the remaining grid node. In Figure 5.3 we see that each grid node has the following interval $I'_1 = [0, 5)$, $I'_2 = [5, 10)$, $I'_3 = [10, 15)$, $I'_4 = [15, 20)$ and $I'_5 = [20, 24)$ of virtual nodes.

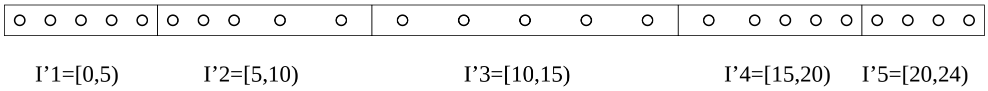

Figure 5.3: Intervals I'_i in the virtual tree space.

Finally, in step 7 of the algorithm, we map the interval I'_i in the virtual tree node space into the interval I_i in the tree node space. Observe that to compute the function f we need to use the intervals V_j defined above. For this case f is defined as follows:

$$f(x) = \begin{cases} 0 \times 4 + \frac{x-0}{2} & \text{for all } 0 \leq x < 8 \\ 1 \times 4 + \frac{x-8}{1} & \text{for all } 8 \leq x < 12 \\ 2 \times 4 + \frac{x-12}{1} & \text{for all } 12 \leq x < 16 \\ 3 \times 4 + \frac{x-16}{2} & \text{for all } 16 \leq x < 24 \\ 4 \times 4 & \text{if } x = 24 \end{cases}$$

The intervals I_i computed directly from the function f defined above and the intervals I'_i are represented in Figure 5.4.

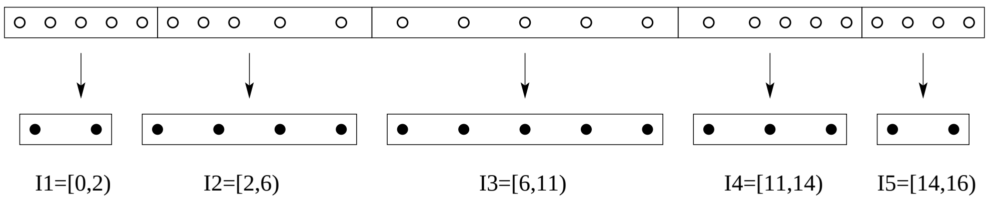

Figure 5.4: Intervals I_i in the tree node space.

From interval I_i it is straightforward to determine the tree partition assigned to the i-th grid node. For instance, grid node number 3 is going to extract models with prefixes CG, CT, GA, GC and GG. The tree that is going to be considered by each grid node is represented in Figure 5.5.

Note that the first to the fifth grid nodes are going to extract structured motifs from trees with weight $0.5 \times 2 = 1$, $0.5 \times 2 + 0.25 \times 2 = 1.5$, $0.25 \times 5 = 1.25$, $0.25 + 0.5 \times 2 = 1.25$ and $0.5 \times 2 = 1$, respectively. Hence, the maximum overload among the grid nodes is $1.5 - 1 = 0.5$ which is less than the imposed maximum overload $\varepsilon = \frac{3}{2}$.

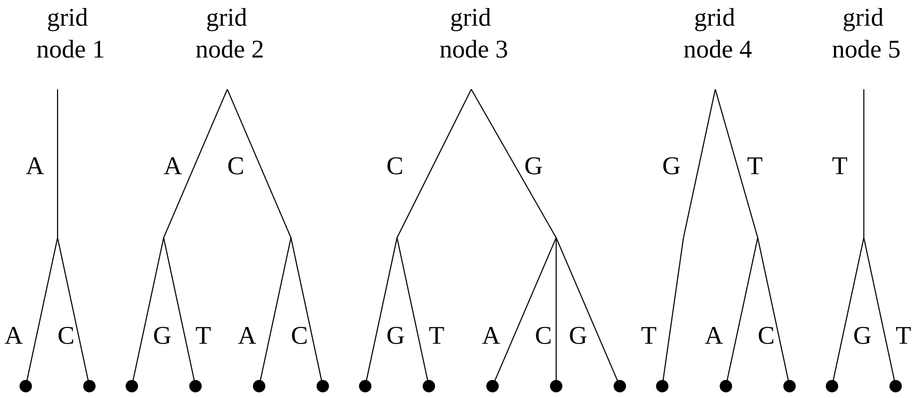

Figure 5.5: Tree partition up to ε.

5.2.2 Parallel extraction

Herein we describe how a partition of the lexicographic trie $\mathcal{M}$ of models, obtained as described in Section 5.2.1, can be used to define a partition for the extraction of structured motifs over the suffix tree $\mathcal{T}$ of the input sequences.

The structured motifs that are going to be extracted in each grid node are dictated by the correspondent partition set of the trie $\mathcal{M}$. The extraction itself is going to be made over the full suffix tree $\mathcal{T}$. It is important to notice that the full suffix tree $\mathcal{T}$ is going to be built in all grid nodes. This follows from the fact that the algorithm to extract single motifs [Sag98] might need to traverse all the tree $\mathcal{T}$ to check whether a model is valid. From a memory requirement point of view, a better solution would be to build only the part of $\mathcal{T}$ necessary to check the validity of the models assigned to each grid node. However, in this first approach to parallelize the structured motif extraction, we do not address the optimal space parallel complexity problem. As concerns time, the complexity of the extraction is by far larger than the complexity of building $\mathcal{T}$. Therefore, we do not waste too much time by forcing each grid node to compute the full suffix tree.

Although it is possible to define a parallel algorithm for any of the structured motif extraction algorithms presented in Chapter 3 and Chapter 4, we present only the parallel version of the Algorithm 3.2. The parallel algorithm to extract structured motifs in a grid node for a given partition set I is presented in Algorithm 5.3.

Note that in the fifth step of Algorithm 5.3 it is necessary to check whether a model belongs

Algorithm 5.3 PExtractMotifs, parallel version of Algorithm 3.2

PExtractMotifs(partition set I of $\mathcal{M}$, model m, box i)

1. for each node-occurrence v of m
2. if $(i > 1)$
3. put in $PotentialStarts$ the children of v at levels $(i-1)k + (i-1)d_{min}$ to $(i-1)k + (i-1)d_{max}$
4. else put v in $PotentialStarts$
5. for each motif $m_i \in I$ obtained by traversing $\mathcal{T}$ from the node-occurrences in $PotentialStarts$
6. if $(i < p)$ PExtractMotifs$(I, m_1 \ldots m_i, i+1)$
7. else KeepModel$(\langle(m_1, \ldots, m_p), (d_{min}, d_{max})\rangle)$

to I. If the partition set I was not an interval it might be quite hard to check the conditions on those steps. This is why we decided that the Algorithm 5.2 should return intervals as partition sets. Once again we stress that this algorithm was designed having in mind a trade-off between finding an optimal number of cuts and finding optimal representations of the partition sets.

We are now able to present the parallel algorithm that runs in each grid node to extract the structured motifs. The pseudo-code of the parallel algorithm is presented in Algorithm 5.4.

Algorithm 5.4 PSMILE, parallel SMILEv1.45 algorithm (with frequencies estimation)

PSMILE(grid node i, overload ε)

1. compute weights $(w_j)_{1\leq j\leq|\Sigma|}$
2. build suffix tree $\mathcal{T}$
3. create colors on $\mathcal{T}$
4. let I_i= SimpleCut$(i, |\Sigma|, r, (w_j)_{1\leq j\leq|\Sigma|}, |\Sigma|, \varepsilon)$
5. call PExtractMotifs(I_i, λ, 1)

Next, we show that PSMILE is work-efficient with respect to the sequential version. Recall that the *work* of a parallel algorithm is computed by the product of the number of processing units and the time complexity of the most demanding unit. An algorithm is said to be *work-efficient* with respect to the sequential version whenever its work is asymptotically upper bounded by the time complexity of the sequential algorithm.

Before showing that PSMILE is work-efficient we have to make some modifications to the algorithm. Observe that in the worst case scenario all tree leaves extract structured motifs. For this reason, the estimation of the frequencies and the computation of the weights done at the first step of Algorithm 5.4 may lead to a partition that produces worse time complexity results than a uniform partition. Hence, in order to achieve work-efficiency, we replace the first step of Algorithm 5.4 by the step presented in Algorithm 5.5.

Algorithm 5.5 PSMILE, parallel SMILEv1.45 algorithm (without frequencies estimation)

PSMILE(grid node i, overload ε)

1. let $w_j = 1$ for $1 \leq j \leq |\Sigma|$
2. build suffix tree $\mathcal{T}$
3. create colors on $\mathcal{T}$
4. let I_i= SimpleCut$(i, |\Sigma|, r, (w_j)_{1 \leq j \leq |\Sigma|}, |\Sigma|, \varepsilon)$
5. call PExtractMotifs(I_i, λ, 1)

Moreover, we assume that the alphabet Σ is fixed, a common assumption when measuring time complexity of algorithms involving suffix trees. Then, by Proposition 5.1.3, the time complexity of Algorithm 5.2 can be expressed as $O(1)$. Furthermore, since both Σ and w_j for $1 \leq j \leq |\Sigma|$ are constants, w is also constant. We obtain the following result for the general case of $d_{min} \leq d_{max}$.

Proposition 5.2.1 Algorithm 5.5 is work-efficient with respect to the sequential version when $r = O\left(\nu^p(e, k)\right)$ and $\frac{\varepsilon}{w} \leq \frac{1}{r}$.

Proof: We start by computing the time complexity of Algorithm 5.5 for each grid node. The first step is $O(1)$ since $w_j = 1$, for all j, and the alphabet is fixed. The second step requires $O(Nn)$ time, where n is the average length of the input sequences, using any linear time suffix tree construction algorithm [Wei73, McC76, Ukk95]. The time complexity of the third step is $O(Nn_{pk+(p-1)d_{\max}})$, as shown in the original paper by Marsan and Sagot [MS00]. The fourth step requires $O(1)$ time. Finally, to determine the time complexity of the fifth step, notice that in the worst case scenario $|\Sigma|^{pk}$ is the total number of models. Moreover, the partition of the models assigns to any grid node

$$\begin{aligned} O\left(\frac{|\Sigma|^{pk}}{r} + |\Sigma|^{pk}\frac{\varepsilon}{w}\right) &= O\left(2 \times \frac{|\Sigma|^{pk}}{r}\right) \quad \text{by hypothesis } \frac{\varepsilon}{w} \leq \frac{1}{r} \\ &= O\left(\frac{|\Sigma|^{pk}}{r}\right) \end{aligned}$$

models. Furthermore, in the worst case where all possible models are valid, the time complexity associated with the extraction of structured motifs is directly proportional to the number of models extracted, and so by Proposition 3.3.5, each model takes

$$O\left(\frac{p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{|\Sigma|^{pk}}\right)$$

time and each grid node takes

$$\begin{aligned} &O\left(\frac{p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{|\Sigma|^{pk}}\right) \times O\left(\frac{|\Sigma|^{pk}}{r}\right) = \\ &O\left(\frac{p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{r}\right) \end{aligned}$$

time to extract structured motifs. The complexity of the last step upper bounds the complexity of the rest of the algorithm because $r = O\left(\nu^p(e,k)\right)$.

We conclude, by comparing with Proposition 3.3.5, that the overall work complexity is

$$\begin{aligned} &O\left(\frac{p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{r}\right) \times r = \\ &O\left(p\Delta^2 n_{(p-1)k+(p-1)d_{max}}\nu^{p-1}(e,k) + Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)\right) \end{aligned}$$

and so Algorithm 5.5 achieves work-efficiency when $r = O\left(\nu^p(e,k)\right)$ and $\frac{\varepsilon}{w} \leq \frac{1}{r}$. □

The above result asserts that to achieve work-efficiency the number of processing units should not grow faster than the most demanding parcel of the extraction process, which is the exponential parcel $\nu^p(e,k)$. Moreover, the imbalance should be inversely proportional to the number of processing units.

5.3 Experimental results

The aim of the present section is to describe the set up of the parallelized algorithm as well as a comparison between time results of the sequential (Algorithm 3.2) and the parallel (Algorithm 5.3) version of the algorithm, SMILEv1.45 and PSMILE (Algorithm 5.4), respectively.

For the computational grid infrastructure we used the open source Globus Toolkit 2.4. Globus is a research project and its primary goal is to provide basic technology that enables entirely new classes of applications, one of which is distributed supercomputing. The Globus version 2.4 was installed in four machines, three inside the same LAN (Pentium IV 2.4GHz with 1GB of RAM, Pentium IV Xeon 2.4GHz with 4GB of RAM and Pentium III 1.2GHz with 1GB of RAM) and one in the WAN (Pentium IV 2.5GHz with 512MB of RAM).

As test sets we used two groups of sequences. The first group contains only one test set with gene sequences of *S. cerevisiae*. This set is composed by 23 gene sequences, for a total of 23,000 nucleotides, encoding proteins that are up-regulated in yeast cells exposed to the herbicide 2,4-D, as assessed by quantitative proteomic analysis. Since this dataset belongs to an eukaryote the definition of the promoter or regulatory site models was more complex. In this work two different models, with two and three boxes, were tested with the objective of recognizing specific promoters, relying on combinations of individual elements. The second group is composed of three sets of non-coding sequences located between two divergent genes and extracted from the whole genomes of *B. subtilis*, *E. coli* and *H. pylori*. The first of these sets contains 1,062 sequences for a total of 196,736 nucleotides. The second set contains 1,1148 sequences and 226,928 nucleotides. The last set contains 308 sequences and 52,100 nucleotides. These three datasets were originally used as test cases for extracting promoter consensi and to test the sequential algorithm [MS00].

The SMILEv1.45 and PSMILE algorithms were tested with both group of sequences. The results obtained by PSMILE in two experiments, one with each group of sequences, are presented in Table 5.1. We can conclude by comparing these results with results for

	# Extracted models	CPU Times (in seconds)	# Extracted models	CPU Times (in seconds)
Grid node 1	2	155.83	9987	444.50
Grid node 2	1	168.11	6178	385.28
Grid node 3	2	245.35	3108	473.70
Grid node 4	16	<u>262.51</u>	15884	<u>581.64</u>
Total	21	831.80	35157	1885.18

Table 5.1: Extraction of structured motifs by PSMILE algorithm.

SMILEv1.45, presented in Table 5.2, that the speed-up is almost linear for both tests.

	CPU Times (in seconds)	
SMILEv1.45	757.97	1790.70
PSMILE	262.51	581.64
Speed-up	2.9	3.1

Table 5.2: Extraction of structured motifs by SMILEv1.45 and PSMILE algorithms.

The second group of sequences were also tested to obtain an estimation of the linear work-efficiency coefficient. We set up several extractions where

$$r = \frac{\nu^{\frac{p}{2}}(e,k)}{2} \leq \nu^{p}(e,k) \text{ and } \varepsilon = \frac{1}{r} \leq \frac{w}{r}$$

and computed the quotient

$$\text{wef} = \frac{\text{parallel work}}{\text{sequential time}} = \frac{r \times \text{parallel time}}{\text{sequential time}}$$

between the parallel work and the sequential time results. The results are summarized in Figure 5.6. We conclude that the linear coefficient for work-efficiency is between 1 and 1.6.

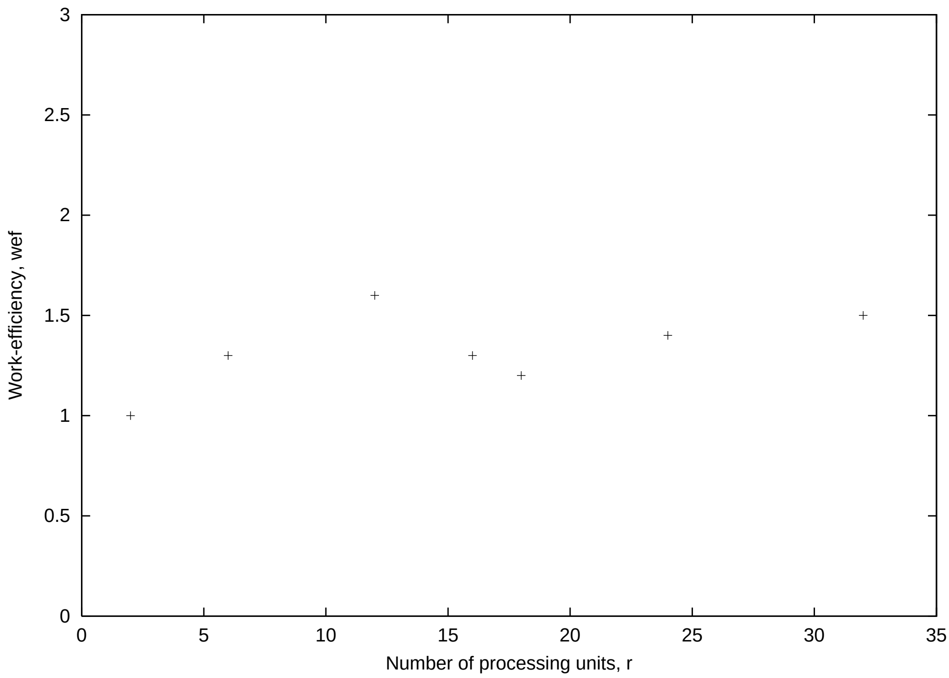

Figure 5.6: Estimation of the linear coefficient for work-efficiency.

Other tests were performed to study the exponential behaviour of the sequential algorithm. We present a result, in Figure 5.7, where an incremental increase on the factor e of box errors produces an exponential time result. Note that in this case, a speed-up in the order of the

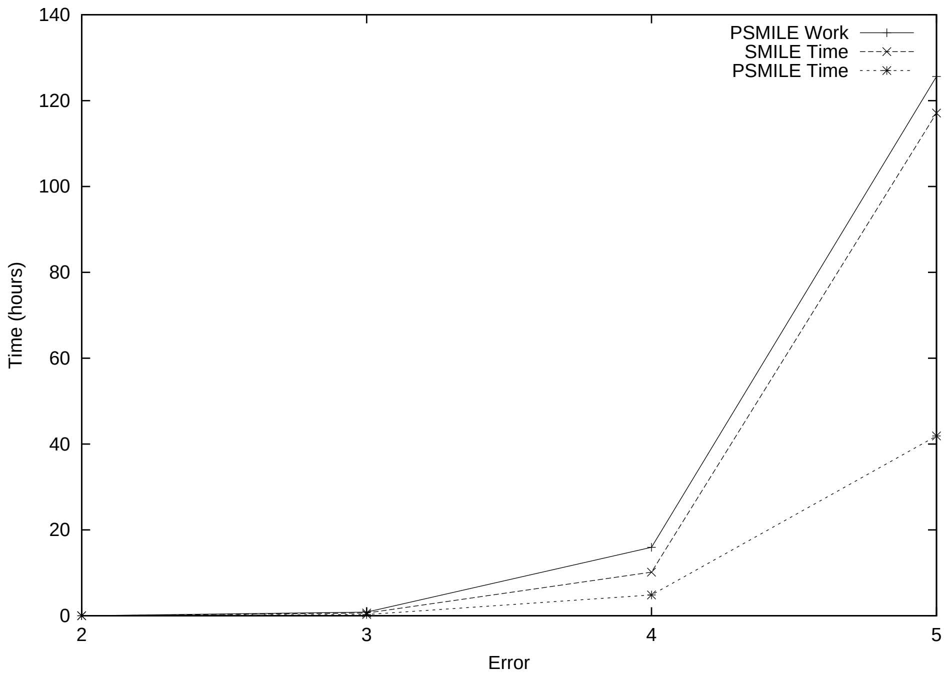

Figure 5.7: PSMILE results with 3 processing units.

available processing units is only achieved when we have $e = 5$.

Error	2	3	4	5
Speed-up	2.0	2.2	2.1	2.8

Currently we are increasing the size of the grid in order to perform more exhaustive tests.

Chapter 6

Conclusions

The crux of this thesis is twofold. First, we proposed a new algorithm and data structure for the extraction of repeated motifs in DNA sequences. Second, we proposed a generic method for the parallelization of algorithms to extract such motifs. Both efforts present several contributions which are detailed in the following.

The first significant contribution is the new algorithm for extracting structured motifs, presented in Section 4.2, which exhibits an exponential time and space gain, in the worst case analysis, relatively to existing algorithms for the extraction of structured motifs, proposed by Marsan and Sagot [MS00] and presented in Section 3.3. This improvement is obtained because the time complexity does not depend on the distances between the boxes of the structured models. The only added cost comes from the computation of box-links, which is negligible in comparison with the time required to perform the extraction of the structured motifs. Moreover, the proposed algorithm only requires the creation of a suffix tree pruned at the depth of the largest box of the structured models, called a factor tree [AS03], saving space in comparison with the algorithms proposed in [MS00] that are based on the full suffix tree. On the other hand, in the average case scenario, the box-link data structure may have a slight increase in memory requirements, but it is straightforward to make a trade-off between computing some box-links and having others stored in memory.

Another significant contribution comes from an implementation of the proposed algorithm that was applied to extract frequent motifs in a few biologically relevant datasets. Moreover, we presented an empirical comparison between our implementation and available implementations of existing algorithms that are capable of detecting multiple motifs composed of any number of boxes. The experimental results show that the new RISO algorithm is much faster than SMILEv1.45 algorithm [MS00], in some cases, more than two orders of magnitude faster.

Finally, concerning contributions of the parallelization work presented in Chapter 5, a new interesting strongly NP-complete problem, the PARTITION UP TO ε problem, was presented. This problem is relevant in the design of efficient parallel searching algorithms where the search space is represented by a tree. For this reason an approximation algorithm for the optimization version of the problem was established. Moreover, by capitalizing on the previous result, a parallel algorithm for the efficient extraction of structured motifs from genomic sequences was proposed. This algorithm is shown to be work efficient, with respect to the sequential algorithm, under easily achievable conditions.

Future work can progress in several directions. From an algorithmic point of view, and keeping in mind further improvements of the technique proposed in Chapter 5, it would be interesting to partition a suffix or factor tree among the available processing units in a way that optimal parallel space complexity is obtained. With such an algorithm we could obtain work-efficiency for the proposed parallelization in an even more general setting. Furthermore, we shall apply the parallelization technique proposed in Chapter 5 to the new algorithm proposed in Section 4.2. This approach will allow the use of even larger datasets which is a major limitation of current solutions. From an implementation point of view, it is worthwhile noticing that the code developed for the prototype implementation of the algorithm presented in Section 4.2 can be further optimized. From a biological point of view, it would be interesting to integrate the RISO algorithm with a database of transcription factors and respective promoter consensi, which is presently being set up, providing in this way semi-automatic methods for processing experimental results. With such a tool users can easily analyze complex interactions between gene networks and proteins. Finally, in a more general point of view, an obviously relevant research problem in this area is the design and development of faster algorithms to extract binding-site consensi from genomic sequences, possibly putting into consideration new biological and computational models for cis-regulatory regions.

Bibliography

[AIL+88] A. Apostolico, C. Iliopoulis, G. Landau, B. Schieber, and U. Vishkin. Parallel construction of a suffix tree with applications. *Algorithmica*, 3:347–365, 1988.

[AS03] J. Allali and M.-F. Sagot. The at most k-deep factor tree. Submitted for publication, 2003.

[BE94] T. L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the 2nd International Conference on Intelligent Systems for Molecular Biology*, pages 28–36, 1994.

[BE95a] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using em. *Machine Learning*, 21(1-2):51–80, 1995.

[BE95b] T. L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *Proceedings of the 3rd International Conference on Intelligent Systems for Molecular Biology*, pages 21–29, 1995.

[BJEG98] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):279–305, 1998.

[BJVU98] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research*, 8(11):1202–1215, 1998.

[CFOS04a] A. M. Carvalho, A. T. Freitas, A. L. Oliveira, and M.-F. Sagot. Efficient extraction of structured motifs using box-links. Submitted for publication, 2004.

[CFOS04b] A. M. Carvalho, A. T. Freitas, A. L. Oliveira, and M.-F. Sagot. A parallel algorithm for the extraction of structured motifs. In *Proceedings of the 19th ACM Symposium on Applied Computing*, pages 147–153, 2004.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press, 1990.

[CS92] L. R. Cardon and G. D. Stormo. Expectation Maximization algorithm for identifying protein-binding sites with variable length from unaligned DNA fragments. *Journal of Molecular Biology*, 223(1):159–170, 1992.

[CS04] M. Crochemore and M.-F. Sagot. *Motifs in sequences: localization and extraction.* Handbook of Computational Chemistry. Marcel Dekker, Inc., 2004. To appear.

[DB97] L. Duret and P. Bucher. Searching for regulatory elements in human noncoding sequences. *Current Opinions in Structural Biology*, 7(3):399–406, 1997.

[EKGP03] E. Eskin, U. Keich, M. S. Gelfand, and P. A. Pevzner. Genome-wide analysis of bacterial promoter regions. In *Proceedings of the 8th Pacific Symposium on Biocomputing*, pages 29–40, 2003.

[EP02] E. Eskin and P. A. Pevzner. Finding composite regulatory patterns in DNA sequences. *Bioinformatics*, 18(1):354–363, 2002.

[FMFM95] Y. M. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit. Identification of common motifs in unaligned DNA sequences: application to Escherichia coli Lrp regulon. *Computer Applications in Biosciences*, 11(4):379–387, 1995.

[GJ79] M. Garey and D. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman, 1979.

[Gus97] D. Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, 1997.

[HAI01] E. Hunt, M. P. Atkinson, and R. W. Irving. A database index to large biological sequences. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 139–148, 2001.

[Har97] R. Hariharan. Optimal parallel suffix tree construction. *Journal of Computer and Systems Sciences*, 55(1):44–69, 1997.

[Hui92] L. C. K. Hui. Color set size problem with applications to string matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Spriger-Verlag, 1992.

[KFQW99] A. Klingenhoff, K. Frech, K. Quandt, and T. Werner. Functional promoter modules can be detected by formal models independent of overall nucleotide sequence similarity. *Bioinformatics*, 15(3):180–186, 1999.

[KOB89] S. Karlin, F. Ost, and B. E. Blaisdell. Patterns in DNA and amino acid sequences and their statistical significance. In M. S. Waterman, editor, *Mathematical Methods for DNA Sequences*, pages 133–158. CRC Press, 1989.

[Kur99] S. Kurtz. Reducing the space requirement of suffix trees. *Software: Practice and Experience*, 29(13):1149–1171, 1999.

[KYD96] C. Kirchhamer, C. Yuh, and E. Davidson. Modular cis-regulatory organization of developmentally expressed genes: two genes transcribed territorially in the sea urchin embryo, and additional examples. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 93, pages 9322–9328, 1996.

[McC76] E. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[MS00] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7(3-4):345–362, 2000.

[PTVF93] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C: the art of scientific computing.* Cambridge University Press, 1993.

[PVMZ04] A. Policriti, N. Vitacolonna, M. Morgante, and A. Zuccolo. Structured motifs search. In *Proceedings of the 8th Annual International Conference on Computational Molecular Biology*, pages 133–139, 2004.

[RDR$^+$02] S. Robin, J.-J. Daudin, H. Richard, M.-F. Sagot, and S. Schbath. Occurrence probability of structured motifs in random sequences. *Journal of Computational Biology*, 9:761–773, 2002.

[Sag98] M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In C. Lucchessi and A. Moura, editors, *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics*, volume 1380 of *Lecture Notes in Computer Science*, pages 111–127. Spriger-Verlag, 1998.

[SS02] K.-B. Schürmann and J. Stoye. Suffix tree construction for large strings. Technical Report CS-01-02, Fachbereich Informatik, Universität Rostock, 2002.

[Tom99] M. Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, pages 262–271, 1999.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[vHACV98] J. van Helden, B. André, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281(5):827–842, 1998.

[vHRCV00] J. van Helden, A. F. Rios, and J. Collado-Vides. Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Research*, 28(8):1808–1818, 2000.

[VMLS00] A. Vanet, L. Marsan, A. Labigne, and M.-F. Sagot. Inferring regulatory elements from a whole genome. An analysis of Helicobacter pylori sigma(80) family of promoter signals. *Journal of Molecular Biology*, 297(2):335–353, 2000.

[VMS99] A. Vanet, L. Marsant, and M.-F. Sagot. Promoter sequences and algorithmical methods for identifying them. *Research in Microbiology*, 150(9-10):779–799, 1999.

[Wei73] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[Wer99] T. Werner. Models for prediction and recognition of eukaryotic promoters. *Mammilian Genome*, 10(2):168–175, 1999.

Index

3-PARTITION problem, 50, 51

at most k-deep factor tree, 33

bacteria, 5

balanced partition, 49, 50, 52, 53, 55

base, 5

binding site, 6, 7

box, 8, 9, 17

box-link, 2, 33, 35, 36

BoxLink algorithm, 39

cis-regulatory region, 6, 8

coding sequence, 5

Colors array, 13, 15, 16, 19, 24, 37, 42, 45

consensus, 6, 7

core promoter, 6

CREW-PRAM computational model, 50

decision version, 51

distal promoter, 6

DNA, 5

e-occurrence, 15

enhancer, 6

eukaryotes, 5

exon, 5

ExtractMotifs algorithm, 19, 27, 42

factor tree, 33

functional sequence, 7

gene, 5

generalized suffix tree, 12

genome, 5

Globus, 63

grid, 50, 63, 65

grid node, 50, 55, 56, 59, 60

Hamming distance, 15

illi coding, 35

Improved Linked List Implementation, 35

intergenic region, 5

intron, 5, 6

L(i) array, 25, 27

lexicographic trie, 49, 55

Lptr(i) array, 25, 27

MEME, 8, 45

model, 15

motif, 6, 15

motif extraction, 7

motif localization, 7

multiple motif, 8

mutation, 8

node-occurrence, 15

non-coding sequence, 5–7
non-functional sequence, 7
nucleotide, 5, 6

occurrence, 15
optimization version, 51
orthologous genes, 6

PARTITION UP TO ε problem, 50, 51
PExtractMotifs algorithm, 59
phylogenetic footprinting, 6
prokaryotes, 5
promoter, 6–8
promoter region, 6–9
protein, 5
proximal promoter, 6
PSMILE, 60–63
PSMILE algorithm, 60, 61

quorum, 15

ratio bound, 54
RISO, 46, 47
RNA, 5

SimpleCut algorithm, 52, 56
single motif, 8, 9
SMILEv1.45, 46, 47, 62, 63, 68
SMILEv2, 46, 47
SpellMotifs algorithm, 17
structured model, 17
structured motif, 8, 19
suffix tree, 11, 15, 33
suffix trie, 15
suffix-link, 14, 23–26, 30, 37, 42, 43

transcription, 5
transcription factor, 6
transcription factor binding site, 6
translation, 5

valid model, 15
valid structured model, 19

work, 60
work-efficient, 61