# An Efficient Algorithm for the Identification of Structured Motifs in DNA Promoter Sequences

Alexandra M. Carvalho*†, Ana T. Freitas†, Arlindo L. Oliveira†
Marie-France Sagot‡

**Abstract**

We propose a new algorithm for identifying cis-regulatory modules in genomic sequences. The proposed algorithm, named RISO, uses a new data structure, called box-link, to store the information about conserved regions that occur in a well-ordered and regularly spaced manner in the dataset sequences. This type of conserved regions, called structured motifs, is extremely relevant in the research of gene regulatory mechanisms since it can effectively represent promoter models. The complexity analysis shows a time and space gain, over the best known exact algorithms, that is exponential in the spacings between binding sites. A full implementation of the algorithm was developed and made available online. Experimental results show that the algorithm is much faster than existing ones, sometimes by more than four orders of magnitude. The application of the method to biological datasets shows its ability to extract relevant consensi.

**Keywords**: Box-link, Factor tree, Structured motif, Promoter, Binding site consensus.
**Availability**: `http://algos.inesc-id.pt/~asmc/software/riso.html`.

## 1 Introduction

An important part of gene regulation is mediated by specific proteins, called *transcription factors* (TFs), which influence the transcription of a particular gene by binding to specific sites on DNA sequences, called *transcription factor 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*. 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. 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 sites of cooperating TFs are usually organized into short sequence units, called *cis-regulatory modules* (CRMs), 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 [15].

*Work partially supported by the FCT grant SFRH/BD/18660/2004 and the FCT Project FEDER POSI/SRI/47778/2002 BioGrid

†A. M. Carvalho, A. T. Freitas and A. L. Oliveira are with IST/INESC-ID, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal, Email: {asmc, atf, aml}@algos.inesc-id.pt

‡M.-F. Sagot is with Inria Rhône-Alpes, Université Claude Bernarde, Lyon I, 43 Bd du 11 Novembre 1918, 69622 Villeurbanne Cedex, France, Email: marie-france.sagot@inria.fr

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 detailed explanation on possible models for prediction and recognition of eukaryotic promoters can be found in the literature [32]. Therein, an eukaryotic promoter is viewed as being composed of three CRMs with different functions, each one having one or more TF 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.

The DNA sites involved in gene regulation 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 TF. The second approach is called *phylogenetic footprinting* [8] and requires careful identification of the appropriate genes to use.

There are two central problems concerning motifs in sequences: localization and extraction [7]. The goal of the *motif localization* problem is to find the positions in a sequence of the occurrences of a given motif [19]. The task addressed in this paper is the *motif extraction* problem and aims to identify *de novo* binding site consensi from a set of non-coding DNA sequences. Given the flexibility of regulatory mechanisms, it is essential to develop computer-assisted motif 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.

Identifying promoter sequences is notoriously difficult for both prokaryotic and eukaryotic organisms. There are two major challenges 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 prediction 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 [3, 30]. In the following we concentrate on briefly surveying methods that extract conserved single or multiple binding sites, possibly located at constrained distances from one another in a set of sequences which potentially contain a cis-regulatory region.

The first methods for detecting promoter regions in DNA sequences [23, 27] looked only for a unique binding site, called *single motif*. In the search for more complex cis-regulatory models methods have appeared that extract DNA sites composed by two binding sites [4, 29]. The first attempts to identify several binding sites, called *multiple motifs*, consisted in crossing compatible single motifs [2, 9, 10], which takes time at least quadratic in the number of single motifs and their occurrences. To address this problem, the lists for single motifs were trimmed by statistical significance before the crossing operation. However, a motif composed of several binding sites may be statistically significant even though none of the sites taken individually are. Indeed, one of the main interests in seeking for multiple motifs lies exactly in this fact.

There are few realistic methods in the literature which attempt to find a modular organization of binding sites for TFs that cooperate in the regulation of genes. Some probabilistic methods were proposed to identify CRMs and their component TFs using only the raw sequence data as input [24, 25]. The main problem of these methods is that in the attempt to reduce false positives they also eliminate true positive motifs. Moreover, an exact algorithm [17] was also proposed to flexibly identify motifs composed of any number of binding sites, possibly distributed over different CRMs. The main drawback of this method is its incapacity to deal with large amounts of genomic data since its complexity grows rapidly both in time and in space. The prime objective of this paper is to address the challenge related to the high computational complexity of this approach.

The main contribution of this work is a new algorithm, named RISO, to identify CRMs from a set of promoter regions of co-regulated genes. The new method achieves an exponential time and space gain, in the worst case analysis, relatively to the best exact algorithm for the problem of multiple motif extraction [17]. Clearly, time and space savings of this magnitude are of the utmost importance when searching through genomic data. In practice, the exponential gain reflects that the extraction remains independent of the distances between the binding sites that build up the multiple motif. This improvement is very important to find eukaryotic TFs since the promoter model can be very complex with consensus sequences observed over very large and variable distances. The most important acceleration element of the proposed algorithm is a new data structure, called *box-link*, which stores the information about how to jump within the DNA sequences from site to site in the multiple motif. The algorithm uses a *factor tree* [1], also known in the literature as *truncated suffix tree* [6], which is a suffix tree [18, 26, 31] built only up to a certain level, leading to significant space savings.

In Section 2 we present related work, describing two algorithms for the extraction of structured motifs [17] and the factor tree data structure [1, 6]. In Section 3 we present the main contribution of this paper, the box-link data structure and the new algorithm to extract structured motifs [5]. A complete complexity analysis of the algorithms for the extraction of structured motifs is presented in Appendix, which can be skipped in a first reading. We demonstrate our results on simulated and real data in Section 4.

## 2 Previous work

We start this section with an overview of the computational model used to identify cis-regulatory regions, the structured motifs, first presented by Marsan and Sagot [17]. We then present two algorithms devised by the same authors [17] for its extraction from a set of DNA sequences. We end this section by presenting the factor tree data structure [1, 6].

### 2.1 Model overview

A *single model*, or simply a *model*, is a non-empty string over the alphabet $\Sigma$={A,C,G,T}. From this point on, we denote the length of a single model $m$ by $k$. A measure of the difference between two sequences of the same length over $\Sigma$ is given by the *Hamming distance*, which is defined as the minimum number of substitutions needed to transform one sequence into another. 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 $e$. A model is said to be a *valid model*, or simply a *motif*, if it has an occurrence in at least $q$ input sequences, where $q$ is called the *quorum*. Motifs are used to describe highly conserved strings in a set of DNA sequences which, in the case of sequences from co-regulated genes, are candidates for binding sites.

**Definition 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 and $d = (d_{min_i}, d_{max_i})_{1 \leq i < p}$ is a $(p-1)$-tuple of pairs, denoting $p-1$ intervals of distance between the $p$ single models.

Each element $m_i$ of a structured model is called a *box* and we denote its length by $k_i$.

**Definition 2** Given a $p$-tuple $(e_i)_{1 \leq i \leq p}$ of allowed substitutions, a structured model $(m, d)$ is said to have an $(e_i)_{1 \leq i \leq p}$*-occurrence* in the input sequences if, for all $1 \leq i \leq p$, there is an $e_i$-occurrence $u_i$ of $m_i$ such that: (i) $u_1, \ldots, u_p$ are in the same input sequence; (ii) the space between the end position of $u_i$ and the start position of $u_{i+1}$ in the sequence is in $[d_{\min_i}, d_{\max_i}]$, for all $1 \leq i < p$.

**Definition 3** A structured model is said to be a *valid structured model*, or a *structured motif*, if it has an $(e_i)_{1 \leq i \leq p}$-occurrence in at least $q$ input sequences.

As expected, structured motifs try to capture highly conserved complex regions in a set of DNA sequences which, in the case of sequences from co-regulated genes, simulate functional combinations of transcription factor binding sites.

## 2.2 Extraction of structured motifs

The two algorithms devised by Marsan and Sagot [17] to extract structured motifs from upstream sequences of co-regulated genes use, as the basic data structure, a suffix tree. A *suffix tree* for a string is a tree where the branches spell all the suffixes of the string. Such a data structure exposes the internal structure of a string and can be used to solve many string problems. For more details see for instance [11]. The construction of a suffix tree in linear time is a problem already addressed by Weiner [31], McCreight [18] and, more recently, by Ukkonen [26]. The suffix tree construction for a set of $N$ input sequences, 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. An usual way to distinguish the input strings of a generalized suffix tree is by storing at each tree node $v$ a Boolean array, called the $Colors_v$ array [13] (usually implemented as a bit vector with dimension $N$). This array indicates the strings in the input set that contain the substring labeling the path from the root to the tree node $v$.

Both algorithms have the same pre-processing phase, which consists in building the generalized suffix tree and adding the $Colors$ array to all its nodes. The only difference resides in the extraction phase. The extraction of structured motifs is based on an algorithm devised by Sagot to extract single motifs [23]. Given a maximum number $e$ of substitutions allowed, it has been shown by Sagot that extracting all $k$-size single motifs can be done by simultaneously and recursively traversing, in a depth-first way, the lexicographic tree $\mathcal{M}$ 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, $\mathcal{M}$ is the tree 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.

Before presenting the extraction phase for both algorithms we need to introduce the following definition.

**Definition 4** A *e-node-occurrence* of a model $m$ is a pair $(v, e_v)$ such that: (i) $v$ is a node in the suffix tree $\mathcal{T}$; (ii) $e_v$ is the Hamming distance between the label of the path from the root to $v$ and $m$; (iii) $e_v \leq e$.

Whenever $e$ is understandable from context we use *node-occurrence* instead of $e$-node-occurrence and we denote it only by $v$. Clearly, when substitutions are allowed ($e > 0$) a model can have more than one node-occurrence in $\mathcal{T}$.

### 2.2.1 Jumping in the suffix tree

We now describe the first published algorithm to extract structured motifs [17]. 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 extraction of structured motifs starts by finding single motifs of length $k$ [23], one at a time. 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 $m_1$ (depicted as thick lines in Figure 1a), a jump is made in 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 1b). Single extractions of all possible second boxes proceed from these nodes (Figure 1c).

The ExtractMotifs algorithm for $p$ boxes is presented in Algorithm 1. It makes use of the $PotentialStarts$ variable, which stores the potential start node-occurrences of the next box being extracted (Figure 1b), and the KeepMotif function, which stores all information concerning valid motifs.

### 2.2.2 Jumping in the suffix tree using suffix-links

The second published algorithm to extract structured motifs [17] capitalizes on the suffix-link data structure of suffix trees. This algorithm achieves an exponential worst case time gain relatively to Algorithm 1 presented in Section 2.2.1.

For clarity of exposition, we consider only structured motifs with two boxes. Moreover, we assume again that each box has the same size $k$, distanced from the next box by some value in the interval $[d_{min}, d_{max}]$, and the same maximum allowed error $e$ per box. The extraction of structured motifs starts by extracting single motifs of length $k$ [23], one at a time. For each node-occurrence $v$ of a first box $m_1$ (Figure 2a), a jump is made in the suffix tree to the descendants of $v$ situated at distances $[k + d_{min}, k + d_{max}]$ from $v$ (Figure 2b). The content of the Boolean $Colors$ array of the nodes 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 2c). Back

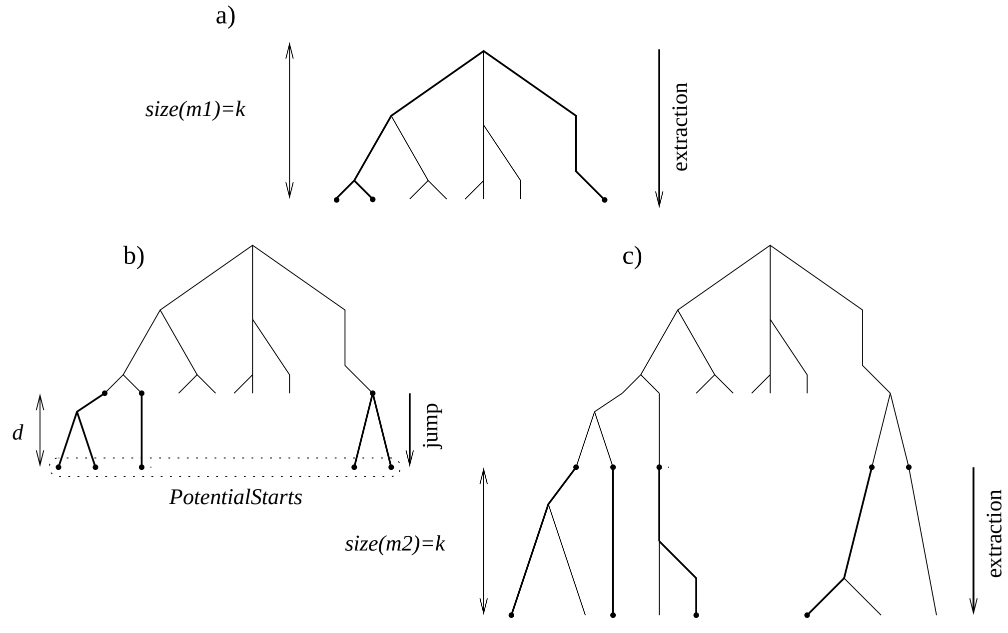


Figure 1: Extracting structured motifs jumping down in the tree.

at level $k$, the grabbed information is used to temporarily update the Boolean $Colors$ of the suffix tree (Figure 2d). The extraction of the second box $m_2$ of a potentially valid structured model then proceeds in the same way (Figure 2e) over the previously updated tree. 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 $m_1$ then follows and the process continues 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 1 presented in Section 2.2.1, it merges at a higher level some branches of lower levels, simplifying the process of the numerous single extractions and leading to an exponential time gain, as we shall see in the Appendix. On the other hand, more space is needed by the second algorithm to restore the tree after each modification it undergoes. The algorithm uses 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 2b), 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 2c), a list $Lptr_{v_k}(i)$ of pointers to the correspondent nodes at $[ik + (i-1)d_{min}, ik + (i-1)d_{max}]$ levels.

The ExtractMotifs algorithm for $p$ boxes is presented in Algorithm 2 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 2b). The variable $NextEnds$ stores the

**Algorithm 1** ExtractMotifs, structured motif extraction jumping in the suffix tree

ExtractMotifs(model $m$, box $i$)

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

nodes at level $k$ needed to update the suffix tree (Figure 2d). 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. 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 $Colors_z = Colors_{\overline{z}} + Colors_{\widehat{z}}$ ($Colors$ are usually implemented as a bit vector, so this means the bitwise OR operation). 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. The function KeepMotif stores all information concerning valid motifs.

#### 2.2.3 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 proposal of these algorithms [17] presents extensions to handle boxes with variable length $k_i$, variable substitution rate $e_i$ and variable intervals of distance $(d_{min_i}, d_{max_i})$. Another proposed constraint imposes the frequency of one or more nucleotides in a box (or among all boxes) to be below or above a certain threshold.

### 2.3 Factor tree

Factor trees are a data structure to index strings [1, 6] very similar to suffix trees. This data structure, also called *k-factor tree*, indexes the substrings of a string whose length does not exceed $k$, and only those. A factor tree is simply a suffix tree pruned at depth $k$. As an example, consider the 5-factor tree for string $S =$AGACAGGAGGC\$ presented in Figure 3 (left). Note that the 5-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-factor tree with collapsed start positions, as depicted in Figure 3 (right).

As for suffix trees, the time and space complexities for constructing factor trees are linear

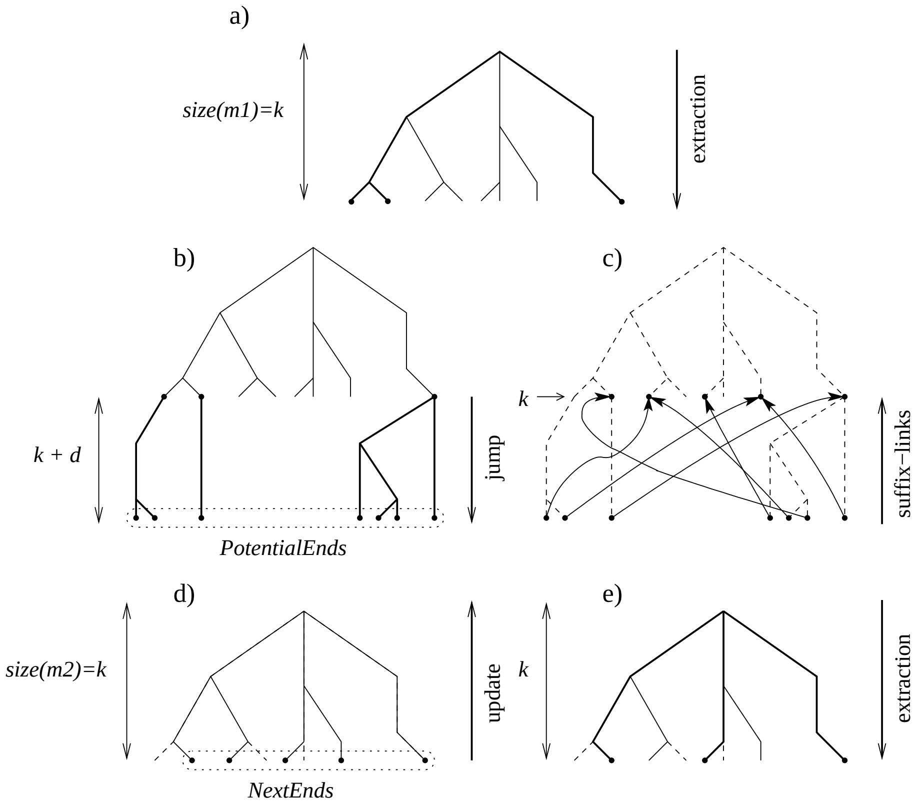


Figure 2: Extracting structured motifs following suffix-links.

in the length of the string [1]. 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. For more details see [1].

To implement the factor tree construction algorithm a new codification is used [1], based on an *Improved Linked List Implementation*, proposed by Kurtz [16] 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 (c.f. positions 5 and 8 in Figure 3). Whenever a new position is added to an already existing leaf we say that the leaf is being updated.

In Section 3.1, we are going to use $list_{leaf}$ to store the leaves of the factor tree as they are inserted or updated. Clearly, the *illi* coding provides this $list_{leaf}$ straightforwardly. As an example, recall Figure 3 (right) 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 $list_{leaf}$ is the length of the input sequence.

**Algorithm 2** 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 $[2k + d_{min}, 2k + d_{max}]$
  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 $[ik + (i-1)d_{min}, ik + (i-1)d_{max}]$
  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) $Colors_z = \overrightarrow{0}$ and 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($\mathcal{T}, L(i)$)

# 3 Extraction of structured motifs using box-links

We now introduce the main contribution of this paper, the box-link data structure. We will first sketch the core ideas of the concept, hoping that it will convey a general idea of its usage. The construction of the box-links from the input sequences is straightforward and we shall examine an algorithm for that purpose quite carefully in Section 3.2. Then, in Section 3.3, we present the algorithm to extract structured motifs using the new data structure. Finally, in Section 3.4 we introduce some extensions to the proposed algorithm.

## 3.1 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. Informally, a box-link is a tuple of tree nodes, corresponding to jumps in 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 in these conditions are illustrated in Figure 4. Note that only a 3-factor tree is needed to solve this problem.

For the sake of simplicity we assume that all $p$ boxes of a structured model are of the

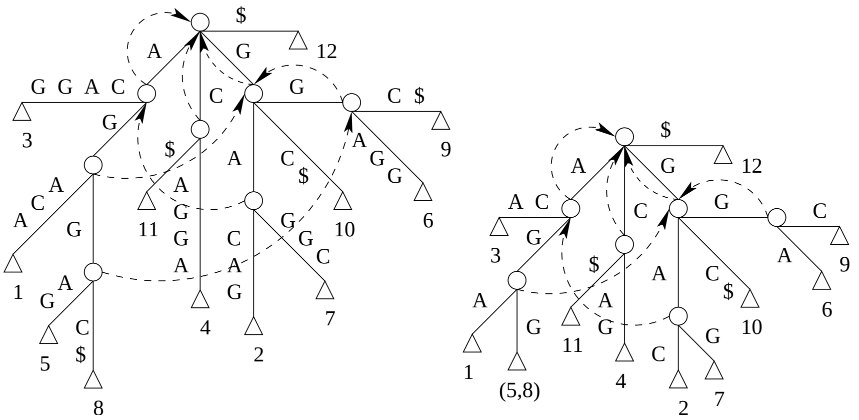

Figure 3: The 5-factor tree (left) and 3-factor tree (right) for string AGACAGGAGGC$.

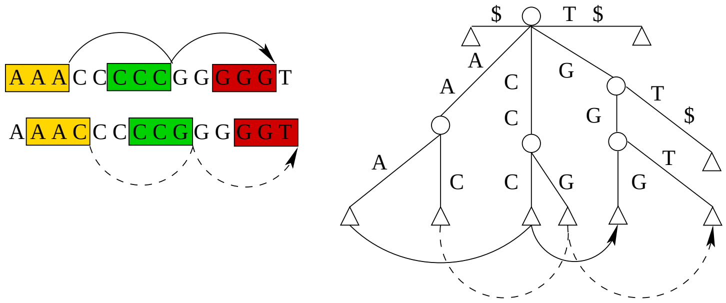


Figure 4: A general idea of box-links.

same size $k$ with distances between them ranging over the interval $[d_{min}, d_{max}]$. A box-link is defined as follows.

**Definition 5** Let $L_k$ be the set of leaves at depth $k$ of a $k$-factor tree $\mathcal{T}$ for a string $S$ and $L_k^i$ denote all possible $i$-tuples over $L_k$. A *box-link of size* $i$, with $1 \leq i < p$, is a $(i+1)$-tuple in $L_k^{i+1}$ such that there is a substring $S'$ of $S$ where: (i) the length of $S'$ is $ik + (i-1)d$; (ii) the $k$-length substring of $S'$ ending at position $jk + (j-1)d$, with $1 \leq j \leq i$, is the path in $\mathcal{T}$ spelled from the root to the $j$-th leaf of the box-link tuple.

The key idea 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 5a), 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 5b). Clearly, there may exist several leaves $l_2$ fulfilling the previous condition. Figure 5 depicts the differences between Algorithm 2 presented in Section 2.2.2, to extract structured motifs using suffix-links (Figure 5b), and the one using box-links (Figure 5c). 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 a box-link $b$ has to be endowed with a Boolean array defined as:

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

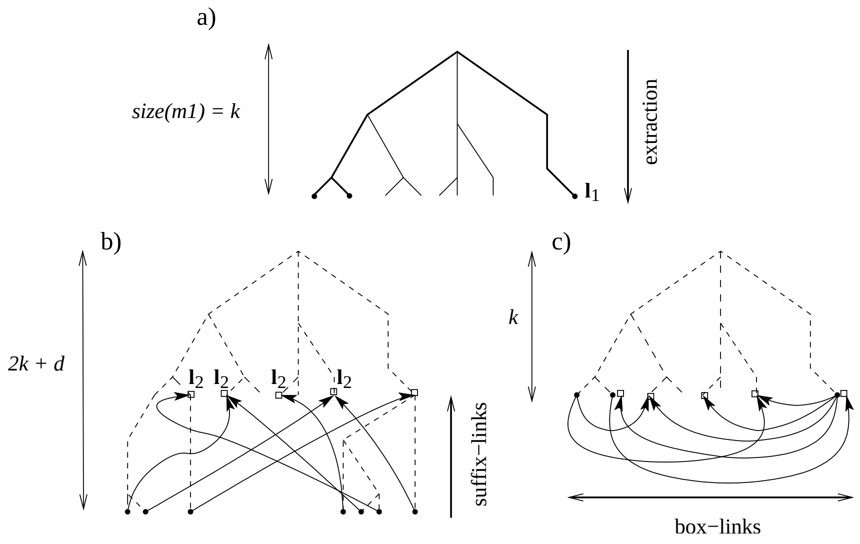


Figure 5: Extracting structured motifs following b) suffix-links and c) box-links.

### 3.2 Box-link construction

The algorithm that builds box-links takes into account that some box-links may collapse from some box on, placing them in a same equivalence class, and leading to a time complexity that is negligible when compared to the time spent extracting structured motifs.

Consider once again the case where we have all $p$ boxes of the same size $k$ with distances between them over the interval $[d_{min}, d_{max}]$. It should be clear (see Figure 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,

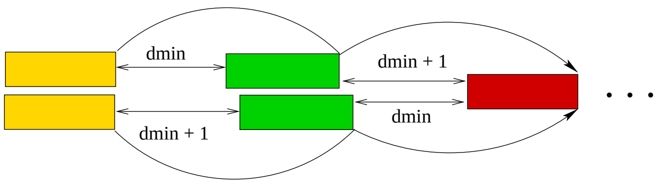


Figure 6: Merge of equivalent box-links from the third box on.

we consider that both box-links collapse in the same box-link from the third until the last box. The algorithm we propose takes this into account by placing box-links in equivalence classes.

In order to define the method used to construct box-links, presented in Algorithm 3, 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 construction of the factor tree, as explained in Section 2.3. In fact, for the sake of exposition, $list_{leaf}$ can be viewed as a family of variables $(list_{leaf_i})_{1 \leq i \leq N}$ (one for each input sequence), where each $list_{leaf_i}$ has the same length as the $i$-th 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 $[b_j]_g$ represents the equivalence class of $j$-size box-links that have the sum of all $j$ distances between boxes equal to $g$. Additionaly, we need 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 a box-link does not exist yet between box-link $b$ and node $v$ (merging in this way equivalent box-links). Either way, creating or not a new box-link, the AddBoxLink function sets the Boolean $Colors$ array entry $i$ to 1.

**Algorithm 3** 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. $[b_0]_0 =$ 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. $[b_j]_{g+h} =$ AddBoxLink($[b_{j-1}]_g, list_{leaf_i}[jk+g+h], i$)
9. remove the first leaf of $list_{leaf_i}$

Algorithm 3 requires some explanation. The first step iterates over all input sequences, while the second step iterates 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 viewed as a 0-size box-link, that is, a 1-tuple which contains only the starting leaf. The fourth step iterates 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 $j$-size box-links, with $1 \leq j < p-1$. The fifth step iterates over all possible source positions for the next link to build, and for each such source step 6 runs over the possible target positions. The instruction in line 7 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.

### 3.3 Jumping in the factor tree using box-links

Assume 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 [17]. 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. The variables and functions used by this algorithm are the same as for Algorithm 2 (Section 2.2.2).

The correctness of Algorithm 4 follows straightforwardly from the correctness of Algorithm 2, presented in the original paper by Marsan and Sagot [17], since box-links mimic the behavior of jumping down the tree and following up the suffix-links.

**Algorithm 4** ExtractMotifs, structured motif extraction using box-links

```
ExtractMotifs(model m, box i)
for each node-occurrence v of m
  for each leaf z such that there is a box-link b_<v,z> from v to z
    put z in L(i)
    if (first time z is reached) Colors_z = 0⃗ and put z in NextEnds
    Colors_z = Colors_z + Colors_{b_<v,z>}
UpdateTree(T, NextEnds)
for each motif m_i obtained by traversing T from the root
  if (i < p) ExtractMotifs(m = m_1 ... m_i, i + 1)
  else KeepMotif(m = <(m_1, ..., m_p), (d_min, d_max)>)
RestoreTree(T, L(i))
```

### 3.4 Extending the algorithm

All extensions presented in Section 2.2.3, 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 when handling boxes of variable length, which we now discuss.

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, in addition to 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 3, but now accounting for variations on the size of the boxes. Moreover, few changes are required to Algorithm 4. The main change 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 [17].

# 4 Experimental results

This section presents results obtained by RISO[1], the C implementation of the new algorithm proposed in this paper, as well as benchmark comparisons with SMILE[2], the C implementation of the algorithm presented in [17].

Two implementations of RISO are available. One conforms to the description of this paper, building box-links in the pre-processing phase: RISO-Static. Another one builds box-links during the extraction phase: RISO-Dynamic. The former has the claimed exponential time gain, relatively to SMILE, with some space cost for storing box-links. The latter has the advantage that it only requires the space of the generalized factor tree furnished with *Colors* arrays, with an extra time cost for computing box-links during the extraction phase.

In both implementations of RISO an optimization was made by pruning from the factor tree all nodes that cannot be part of a structured motif. Such nodes are the ones that have path-labels $S_i[n-j..n]$, where $pk+(p-1)d_{\min} < j \leq 0$, for all $1 \leq i \leq N$ (remember that $n$ is the average size of an input string $S_i$). In practice, this optimization leads to important time savings when there is a large number of boxes and when dealing with boxes separated by large distances.

The results presented were obtained using an Intel Pentium IV at 2.4GHz with 1GB of RAM.

## 4.1 Measuring statistical significance

Once all structured motifs have been extracted, they are classified according to their statistical significance in order to assess their biological relevance. We used for such evaluation the same procedure as in Marsan and Sagot [17], a data shuffling approach [14]. In order to obtain the statistical significance of the models found, a $\chi^2$ 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 [20]. To derive the values in the contingency table for the null hypothesis several random shufflings are performed preserving the $\ell$-mer frequency distribution of the input sequences (an $\ell$-mer is a substring of length $\ell$). Both the number of shufflings and $\ell$ are values given by the user (in general, 100 shufflings conserving di/tri-nucleotides are considered). This process estimates the probability of getting the models observed under the null hypothesis.

## 4.2 Human simulated data

To confirm the claimed exponential gain of RISO over SMILE we performed several experiments over a dataset of 1000 sequences of size 1000 obtained using the Regulatory Sequence Analysis (RSA) tools[3]. These DNA sequences were generated by a Markov chain with order 5, calibrated on intergenic oligonucleotide frequencies for the human DNA, with no planted motifs. We used this data to experimentally verify the exponential gain of RISO over SMILE, which reflects that the extraction remains independent of the distances between the binding

[1]Online version and sources of RISO implementation are available at `http://algos.inesc-id.pt/~asmc/software/riso.html`.

[2]Online version and sources of SMILE implementation are available at `http://www-igm.univ-mlv.fr/~marsan/smile_english.html`.

[3]Available at `http://rsat.scmbb.ulb.ac.be/rsat/`.

sites that build up the multiple motif. Figure 7 contains a summary of the experiments as well as the results obtained in this context.

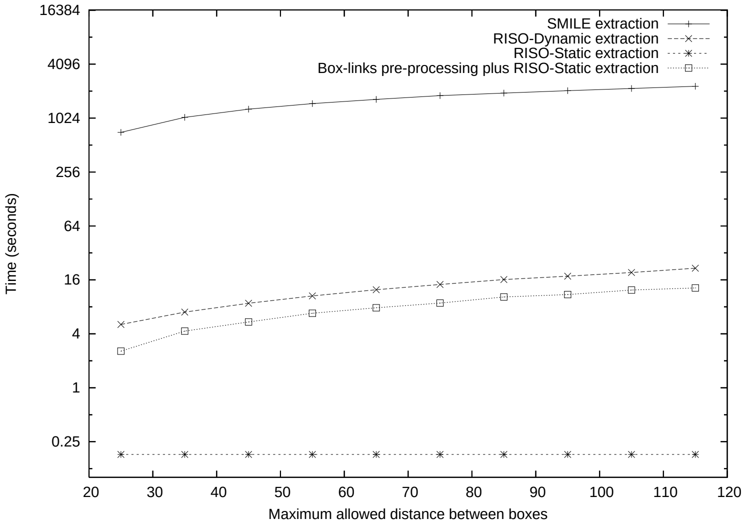


Figure 7: Ten experiments were performed by running SMILE, RISO-Dynamic and RISO-Static over the human simulated dataset. We looked for a dyad consisting of two monad patterns both of size 3 ($k_1 = k_2 = 3$), the first one allowing 1 error and the second one allowing no errors ($e_1 = 1$ and $e_2 = 0$). We wanted these monads to occur at a minimum distance 15 ($d_{min} = 15$) and a maximum distance 25, 35, 45, 55, 65, 75, 85, 95, 105 and 115 ($d_{max} = 25, 35, 45, 55, 65, 75, 85, 95, 105, 115$), for the 10 different experiments, respectively. We required a minimum quorum of 100% ($q = 1000$ in 1000 sequences). Extraction times of RISO and SMILE for this set up are presented in a logaritmic scale.

The extraction time of SMILE and RISO, as well as the gain of RISO over SMILE, for the 10 experiments presented in Figure 7, are presented in Table 1. The first column of RISO-Static concerns the pre-processing time spent to build the box-links, whereas the second column concerns the extraction time of RISO-Static. As expected, fixing $d_{\min} = 15$ and varying $d_{\max}$ from 25 to 115 in 10 different experiments the extraction time of RISO-Static remained 0.18 seconds in all experiments whereas the extraction time of SMILE varied from 709.62 seconds to 2322.91 seconds, as shown in Table 1. The last experiment reflects that RISO performed 12905 times better than SMILE, and this speedup would increase without bounds as $d_{\max}$ grows. In terms of space, SMILE required 20MB whereas RISO-Static required 6MB (with just 1MB to store box-links in all 10 experiments). RISO-Dynamic performed about 140 times better than SMILE in the 10 experiments.

## 4.3 Real data

RISO can be applied to real data as a method to give evidence that some cis-regulatory region occurs in a set of DNA sequences. Such procedure assumes some knowledge of the data, which is reflected in an effective choice of the parameters. When no previous knowledge of the data is available, the only way to proceed is via trial and error over the values of the parameters.

We applied SMILE and RISO to several biological datasets where there are known structured regulatory motifs. In the following we present three of these experiments.

Table 1: Gain of RISO over SMILE for the 10 experiments presented in Figure 7.

| Allowed distance between boxes | Extraction time (seconds) SMILE | RISO-Dyn | RISO-Sta | | Gain SMILE/RISO |
|---|---|---|---|---|---|
| 15..25 | 709.62 | 5.08 | 2.38 | 0.18 | 3942 |
| 15..35 | 1044.00 | 7.00 | 4.11 | 0.18 | 5800 |
| 15..45 | 1289.53 | 8.78 | 5.25 | 0.18 | 7164 |
| 15..55 | 1488.02 | 10.59 | 6.60 | 0.18 | 8267 |
| 15..65 | 1653.87 | 12.42 | 7.64 | 0.18 | 9188 |
| 15..75 | 1823.30 | 14.22 | 8.63 | 0.18 | 10129 |
| 15..85 | 1946.17 | 16.12 | 9.85 | 0.18 | 10812 |
| 15..95 | 2070.23 | 17.59 | 10.79 | 0.18 | 11501 |
| 15..105 | 2193.62 | 19.33 | 12.14 | 0.18 | 12187 |
| 15..115 | 2322.91 | 21.62 | 12.81 | 0.18 | 12905 |

### 4.3.1 The Galp4 promoter in S. cerevisiae

The first biological dataset is formed from a collection of 68 genes of *S. cerevisiae* that are known to be regulated by zinc cluster factors [29]. The upstream sequences were retrieved from positions $-1$ to $-1000$ relative to the ORF start positions. We wanted to detect a dyad signal corresponding to the nucleotides that enter into direct contact with Gal4p [28,29]. The known motif for Gal4p is the long motif CGGrnnnrcynyncnCCG [28] highly degenerated in the 11 middle nucleotides. Therefore, we looked for dyad signals considering the degenerated part as a distance between two distinct monads, as previously done in [29]. For the dyad analysis of the Zn cluster data, we tested two models. A first model with two boxes with size 3 ($k_1 = k_2 = 3$) and distance 11 ($d_{\min_1} = d_{\max_1} = 11$), allowing 1 error in both boxes ($e_1 = e_2 = 1$), and a second one with two boxes with size 4 ($k_1 = k_2 = 4$) and distance 9 ($d_{\min_1} = d_{\max_1} = 9$), allowing 2 errors in both boxes ($e_1 = e_2 = 2$). For the first model the algorithm detected, with a quorum of 100% (68 in 68 sequences), the known motif CGGn$_{11}$CCG. This consensus was classified as the 6th most statistical significant among 4084 extracted motifs. For the second model the consensus CGGAn$_9$TCCG was detected, with a quorum of 100% (68 in 68 sequences), and with highest statistical significance within 65536 extracted motifs. In fact, these dyads are related to each other, and can be assembled to a common pattern CGGan$_9$tCCG. In terms of performance, RISO-Static performed 833 times better than SMILE in the first experiment, and it performed 402 times better in the second one. We stress that the pre-processing time spent to build box-links by RISO-Static was 0.02 and 0.31 seconds in the first and second experiment, respectively. The results are sumarized in Table 2.

Table 2: Dyad analysis of the Zn cluster data.

| Extraction Time (seconds) SMILE | Riso-Dyn | Riso-Sta | Extracted motif |
|---|---|---|---|
| 33.31 | 0.16 | 0.04 | CGGn$_{11}$CCG |
| 1201.50 | 3.27 | 2.99 | CGGAn$_9$TCCG |

### 4.3.2 Transcription activation of CRP in E. coli

The second dataset is composed of 72 sequences from *E. coli* that are known to be regulated by the CRP (Cyclic-AMP Receptor Protein) protein. The CRP complex binds as a dimer and activates transcription by contacting RNA polymerase directly. Although there is great diversity in the way that CRP-dependent promoters are organized, the most commonly found arrangement is for transcription initiation to be dependent on a single CRP dimer, centered between base pairs $-41$ and $-42$ upstream from the transcription start site, with the downstream subunit of the CRP dimer overlapping the core promoter $-35$ region. According to the literature [22], the consensus for the activating regions of the CRP protein is given by the palindromic sequence TGTGA$n_6$TCACA. The core promoter for RNA polymerase contact is given by the consensus sequence TTGACA$n_{16..18}$TATAAT [21]. Following biologists instructions we looked for structured motifs composed of three and four boxes. In particular, we looked for motifs composed of four boxes and searched for evidence of non-overlapping boxes in the $-35$ region of the core promoter. However, we only found the most common triad described in the literature. In Table 3 we present the tests for the following models. First, a model to look for the CRP site only, with two boxes of size 5 ($k_1 = k_2 = 5$), each one allowing 1 error ($e_1 = e_2 = 1$), separated by 5 to 7 nucleotides ($d_{\min_1} = 5, d_{\max_1} = 7$). The consensus TGTGA$n_{5..7}$TCACA was extracted, requiring a quorum of 63% (46 in 72 sequences), from within 117 extracted models. Among the four models classified with highest statistical significance we found all extracted models with first box TGTGA. The first in the rank was our consensus for the CRP site, TGTGA$n_{5..7}$TCACA. Second, we looked for the whole complex with CRP and core promoter sites, searching for a model with three boxes of size 5 ($k_1 = k_2 = k_3 = 5$), with the first box separated by 5 to 7 and the second box separated by 15 to 23 ($d_{\min_1} = 5, d_{\max_1} = 7$ and $d_{\min_2} = 15, d_{\max_2} = 23$) nucleotides. We allowed 1 error for the first and second box and 2 errors for the third box ($e_1 = e_2 = 1$ and $e_3 = 2$). By requiring a quorum of 56% (41 in 72 sequences), the consensus TGTGA$n_{5..7}$TCACA$n_{15..23}$TATAA was found from within 6603 extracted motifs. The statistical significance of this motif was in the best one-third of all extracted motifs, being the best scored motif TGTGA$n_{5..7}$TTCAC$n_{15..23}$TACAT. RISO was 4.2 times faster than SMILE in the first experiment and it was 1.6 times faster than SMILE in the second one. We stress that the consensus TGTGA$n_{5..7}$TCACA$n_{15..23}$TATAA was found, whereas TGTGA$n_{5..7}$TCACA$n_{15..23}$ATAAT was not. This is a clear indication that the third box of the structured model is more degenerated, confirming an assumption that the biologists gave us in first hand. In fact, we only found TGTGA$n_{5..7}$TCACA$n_{15..23}$TATAAT allowing 3 substitutions in the third box and requiring a smaller quorum, which does not give us more useful information than the one we already have with the experiment described in Table 3.

Table 3: Analysis of the *E. coli* data.

| Extraction Time (seconds) SMILE | RISO-Dyn | Extracted motif |
|---|---|---|
| 20.89 | 5.03 | TGTGA$n_{5..7}$TCACA |
| 53.32 | 32.67 | TGTGA$n_{5..7}$TCACA$n_{15..23}$TATAA |

### 4.3.3 The RNA polymerase $\sigma^{70}$ factor in the whole genome of B. subtilis

In the third dataset we used non-coding sequences located between two divergent genes, that is, between genes transcribed in divergent directions (one on each strand), and extracted from the whole genome of *B. subtilis*. This set contains 1062 sequences for a total of 196736 nucleotides and was initially used to test the SMILE algorithm [17]. We wanted to detect a dyad signal corresponding to the sites recognized by the RNA polymerase $\sigma^{70}$ factor (TTGACA$n_{16..18}$TATAAT) [12,17]. The results are summarized in Table 4. We tested a model with two boxes of size 6 ($k_1 = k_2 = 6$), allowing 1 mismatch in both boxes ($e_1 = e_2 = 1$), separated by 16 to 18 nucleotides ($d_{\min_1} = 16$ and $d_{\max_1} = 18$). Requiring a quorum of 12% (128 in 1062 sequences), the consensus TTGACA$n_{16..18}$TATAAT was found with the highest statistical significance from within 6419 extracted motifs. This structured motif appeared in 135 of the 1062 sequences. In terms of performance, RISO-Static was 7.7 times faster than SMILE, and we stress that the time spent to build box-links was only 1.86 seconds.

Table 4: Analysis of the *B. subtilis* data.

| Extraction Time (seconds) | | | Extracted |
|---|---|---|---|
| SMILE | RISO-Dyn | RISO-Sta | motif |
| 349.44 | 47.42 | 45.33 | TTGACA$n_{16..18}$TATAAT |

# 5 Conclusion

We presented a new algorithm and data structure for the extraction of structured motifs in DNA sequences. A complete complexity analysis of the new algorithm was presented showing an exponential time and space gain, in the worst case, relatively to the best known exact algorithms for extracting structured motifs [17]. The only added cost comes from the computation of box-links but this time 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 level of the largest box of the structured motif, saving much space in comparison with the algorithms proposed in [17] that are based on the full suffix tree.

An online version, together with the sources, of the new algorithm was made available at `http://algos.inesc-id.pt/~asmc/software/riso.html`. Experimental results show that RISO is much faster than SMILE [17], in some cases, more than four orders of magnitude faster. The application of RISO to biological datasets shows the ability of the method to extract relevant consensi. To conclude, the exponential time and space gain of the algorithm reflects the possibility of parsing and extracting motifs from full genomic sequences.

Future work can progress in several directions. The algorithm can be easily extended to allow motif hits located in both strands without changing the asymptotic complexity. This extension only requires parsing two times each string, one for each strand, and making a minor change in the factor tree coding. It would also be valuable to combine our approach with probabilistic ones, possibly by modeling each motif within a structured motif using the standard *position specific scoring matrix* (PSSM) representation. We are also exploring the use of our algorithm as part of a framework to unveil the complex gene regulatory network underlying the yeast response to the 2,4-D herbicide and to a new antimalarial/antitumor drug artesunate.

## References


[1] J. Allali. *Comparaison de structures secondaires d'ARN*. PhD thesis, University of Marne-la-Vallée, 2004.

[2] T. L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with MEME. In *Proc. ISMB'95*, pages 21–29, 1995.

[3] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *J. Comp. Bio.*, 5(2):279–305, 1998.

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

[5] A. M. Carvalho, A. T. Freitas, A. L. Oliveira, and M.-F. Sagot. A highly scalable algorithm for the extraction of cis-regulatory regions. In Yi-Ping Phoebe Chen and Limsoon Wong, editors, *Proc. APBC'05*, volume 1 of *ABCB*, pages 273–282. Imperial College Press, 2005.

[6] J. Chae Na, A. Apostolico, C. S. Iliopoulos, and K. Park. Truncated suffix trees and their application to data compression. *Theor. Comput. Sci.*, 304(1-3):87–101, 2003.

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

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

[9] E. Eskin, U. Keich, M. S. Gelfand, and P. A. Pevzner. Genome-wide analysis of bacterial promoter regions. In *Proc. PSB'03*, pages 29–40, 2003.

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

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

[12] J. D. Helmann. Compilation and analysis of Bacillus subtilis $\alpha$-dependent promoter sequences: evidence for extended contact between RNA polymerase and upstream promoter DNA. *Nuc. Ac. Res.*, 23(13):2351–2360, 1995.

[13] L. C. K. Hui. Color set size problem with applications to string matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proc. CPM'92*, volume 644 of *LNCS*, pages 230–243. Springer-Verlag, 1992.

[14] 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.

[15] 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 *Proc. Natl Acad Sci USA*, volume 93, pages 9322–9328, 1996.

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

[17] 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. *J. Comp. Bio.*, 7(3-4):345–362, 2000.

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

[19] A. Policriti, N. Vitacolonna, M. Morgante, and A. Zuccolo. Structured motifs search. In *Proc. RECOMB'04*, pages 133–139, 2004.

[20] 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.

[21] M. T. Record, W. S. Reznikoff, M. L. Craig, K. L. McQuade, and P.J. Schlax. *Escherichia coli RNA polymerase sigma70 promoters, and the kinetics of the steps of transcription initiation*, volume 1. ASM Press, 1996.

[22] V. A. Rhodius, D. M. West, C. L. Webster, S. J. Busby, and N. J. Savery. Transcription activation at class II CRP-dependent promoters: the role of different activating regions. *Nuc. Ac. Res.*, 25(2):326–332, 1997.

[23] M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In C. Lucchessi and A. Moura, editors, *Proc. Latin'98*, volume 1380 of *LNCS*, pages 111–127. Springer-Verlag, 1998.

[24] E. Segal, Y. Barash, I. Simon, N. Friedman, and D. Koller. A discriminative model for identifying spatial cis-regulatory modules. In *Proc. RECOMB'04*, pages 141–149, 2004.

[25] R. Sharan, I. Ovcharenko, A. Ben-Hur, and R. M. Karp. Creme: a framework for identifying cis-regulatory modules in human-mouse conserved segments. *Bioinformatics*, 19(Suppl 1):i283–i291, 2003.

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

[27] 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. *J. Mol. Bio*, 281(5):827–842, 1998.

[28] J. van Helden, A. F. Rios, and J. Collado-Vides. Comparative amino acid sequence analysis of the C6 zinc cluster family of transcriptional regulators. *Nuc. Ac. Res.*, 24(23):4599–4607, 1996.

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

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

[31] P. Weiner. Linear pattern matching algorithms. In *Proc. SWAT'73*, pages 1–11, 1973.

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

# A Complexity analysis

## A.1 Constant distance between boxes

Herein we establish the time and space complexity for Algorithm 1, Algorithm 2 and Algorithm 4. Even though such results for Algorithm 2[4] have already been shown by Marsan and Sagot [17], we present a detailed proof for the time complexity since it gives additional insight on other results that will be presented later. Moreover, since the complexity is non trivial to compute we start by considering $d_{min} = d_{max} = d$.

We denote by $n_l$ the number of nodes at depth $l$ of the suffix tree and by $\nu(e,k)$ the number of distinct words that are at a Hamming distance at most $e$ from a $k$-long word. The following upper bound for $\nu(e,k)$ holds: $\nu(e,k) = \sum_{i=0}^{e} \left( \begin{array}{c} k \\ i \end{array} \right) (|\Sigma|-1)^i \leq k^e|\Sigma|^e$. We recall that $N$ is the number of input sequences and $n$ is the average size of an input sequence.

**Proposition 1** Algorithm 1 takes $O(Nn_{pk+(p-1)d}\nu^p(e,k))$ time and $O(N^2n)$ space.

**Proof:** The proof of time and space complexities can be found in [17]. $\square$

**Proposition 2** Algorithm 2 takes $O(Ns_p(k,d)\nu^p(e,k) + Nn_{pk+(p-1)d}\nu^{p-1}(e,k))$ time, where $s_p(k,d) = \min\{n_k^p, n_{pk+(p-1)d}\}$, and $O(N^2n + Npn_k)$ space.

**Proof:** We can parcel out the complexity of Algorithm 2 into three parts: (i) the total number of operations needed to build the $p$ parts of all structured motifs; (ii) the total number of operations needed to update $\mathcal{T}$, for all parts of a structured motif; (iii) the total number of operations needed to restore $\mathcal{T}$.

In order to compute (i) we have to calculate the cost of all visits we make 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 make to nodes between the root and level $k$ is upper bounded by twice the total number of visits we make 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 $\sum_{i=1}^{p} \min\left\{n_k^i, n_{ik+(i-1)d}\right\} \leq \min\left\{\sum_{i=1}^{p} n_k^i, \sum_{i=1}^{p} n_{ik+(i-1)d}\right\} = O(\min\left\{2n_k^p, 2n_{pk+(p-1)d}\right\}) = O(s_p(k,d))$. However, when up to $e$ substitutions are allowed, a node at level $k$ may be visited

[4] The time complexity result is different from what is presented by Marsan and Sagot [17] due to an acknowledged imprecision in the later.

$\sum_{i=1}^{p} \nu^i(e,k) = O(\nu^p(e,k))$ times more. Hence, the total number of visits to nodes at level $k$ is $O(s_p(k,d)\nu^p(e,k))$.

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 (ii) we need to count the total number of operations necessary to modify the first $k$ levels of the suffix tree which is upper bounded, up to a constant factor, by $Nn_{pk+(p-1)d}\nu^{p-1}(e,k)$. This corresponds to all visits made 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 (iii) is also upper bounded by the time to update $\mathcal{T}$, we conclude that Algorithm 2 takes $O(Ns_p(k,d)\nu^p(e,k)+Nn_{pk+(p-1)d}\nu^{p-1}(e,k))$ time.

The proof of space complexity can be found in [17]. □

This algorithm exhibits an exponential worst case time gain relatively to Algorithm 1. 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** Algorithm 4 takes $O(Ns_p(k,d)\nu^p(e,k) + Nb_p(k,d)\nu^{p-1}(e,k))$ time, where $b_p(k,d) = \min\{n_k^p, n_{pk+(p-1)d}\}$ is an upper bound for the total number of $(p-1)$-size box-links.

**Proof:** Given the similarity with the ExtractMotifs algorithm presented in Section 2.2.2, to compute the complexity of Algorithm 4 we just have to compute the total number of operations needed to update $\mathcal{T}$ (all other parcels are similar to Algorithm 2).

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 $Nb_p(k,d)\nu^{p-1}(e,k)$ up to a constant factor. This corresponds to all visits made to nodes $z$ coming from box-links $b_{\langle v,z\rangle}$ 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 takes $O(Ns_p(k,d)\nu^p(e,k) + Nb_p(k,d)\nu^{p-1}(e,k))$ time. □

This algorithm exhibits an exponential time gain, in the worst case, relatively to the previous algorithms to extract structured motifs presented in Section 2.2. 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 the extraction time of the motifs remains independent of the distances between them.

It is important to notice that the time spent building box-links is negligible in comparison with the time spent for the extraction, as we shall see in Proposition 8.

**Proposition 4** Algorithm 4 takes $O(Nb_p(k,d) + Npn_k)$ space.

**Proof:** To compute the space complexity of Algorithm 4 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 2.2. To compare space complexity results of Algorithm 4 with Algorithm 1 and Algorithm 2, we have to rewrite the space complexity of the later two considering that they used a $(pk + (p-1)d)$-factor tree. In this case, the space complexity of Algorithm 1 becomes $O(Nn_{pk+(p-1)d})$ whereas the space complexity of Algorithm 2 becomes $O(Nn_{pk+(p-1)d} + Npn_k)$. Observe that in the worst case scenario, the factor trees are complete and we have $b_p(k,d) + pn_k = \min\{|\Sigma|^{pk}, |\Sigma|^{pk+(p-1)d}\} + p|\Sigma|^k = |\Sigma|^{pk} + p|\Sigma|^k$ (which is $O(|\Sigma|^{pk})$ when $|\Sigma| > 1) < |\Sigma|^{pk+(p-1)d} = n_{pk+(p-1)d}$ which reflects an exponential gain of the order $|\Sigma|^{(p-1)d}$.

## A.2 Variable distance between boxes

Herein we establish the time and space complexity for Algorithm 1, Algorithm 2, Algorithm 3 and Algorithm 4, considering the general case where $d_{min} \leq d_{max}$.

To set up the time complexity for Algorithm 1 we consider that $PotentialStarts$ does not have repetitions. 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 every time they are added to $PotentialStarts$. Also, we define $\Delta = d_{min} - d_{max} + 1$.

**Proposition 5** Algorithm 1 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 1 we have to consider the cost of all visits we make to nodes between the root and level $pk + (p-1)d_{max}$. However, contrary to what happened in Proposition 1 (see [17]), this cost is not upper bounded by the cost of all visits we make 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 make 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.

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 make 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 make to the descendant is $\Delta$. Hence, the number of visits we make 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 make 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 make 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 $O(((p-1)(\Delta-1)+1)n_{pk+(p-1)d_{max}}) = O(p\Delta n_{pk+(p-1)d_{max}})$.

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 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 1 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. □

The space complexity of Algorithm 1 is the same as for constant distances between boxes, presented in Proposition 1.

To establish the time complexity for Algorithm 2 we assume that the follow up of suffix-links to level $k$ is never done twice for the same node, for each motif being searched (see Figure 2c to recall the follow up operation), which is equivalent to saying that $PotentialEnds$ has no repetitions.

**Proposition 6** Algorithm 2 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 5, when we have variable distances between consecutive boxes, an extra term appears in this analysis. In fact, the term concerning the update of the suffix tree, presented in Proposition 2, is divided in two terms, 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 2, we start by analyzing the case where no substitutions are allowed. To compute the first term 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 make 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 make to the descendant is $\Delta$. Hence, the number of visits we make 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 term 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 terms concerning the construction of the $p$ parts of the structured models and the restoration of the suffix tree remain as in Proposition 2. To sum up, reasoning as in Proposition 2, when errors are allowed, we have that Algorithm 2 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. □

The space complexity of Algorithm 2 is the same as for constant distances between boxes, presented in Proposition 2.

**Proposition 7** Algorithm 3 takes $O(N^2 n\Delta^2 p^2)$ time.

**Proof:** The time complexity of Algorithm 3 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 create and/or update the array *Colors* for the box-link being added. We conclude that Algorithm 3 takes $O(N^2 n\Delta^2 p^2)$ time. $\square$

**Proposition 8** Algorithm 3 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 3 requires $O(Nb_p(k, d_{max}))$ space. $\square$

Consider now the time and space complexity for Algorithm 4, for the general case where $d_{min} \leq d_{max}$. In this case, both complexities 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 still hold, since in the worst case $b_p(k, d_{max}) = n_k^p = |\Sigma|^{pk}$.