A highly scalable algorithm for the extraction of cis-regulatory regions

Alexandra M. Carvalho*
INESC-ID
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
asmc@algos.inesc-id.pt

Ana T. Freitas
IST/INESC-ID
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
atf@inesc-id.pt

Arlindo L. Oliveira
IST/INESC-ID
Rua Alves Redol, 9
1000-029 Lisboa, Portugal
aml@inesc-id.pt

Marie-France Sagot †‡
Inria Rhône-Alpes
Université Claude Bernarde, Lyon I
43 Bd du 11 Novembre 1918
69622 Villeurbanne Cedex, France
Marie-France.Sagot@inria.fr

Abstract

In this paper we propose a new algorithm for identifying cis-regulatory modules in genomic sequences. In particular, the algorithm extracts structured motifs, defined as a collection of highly conserved regions with pre-specified sizes and spacings between them. This type of motifs is extremely relevant in the research of gene regulatory mechanisms since it can effectively represent promoter models. The proposed algorithm 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. The complexity analysis shows a time and space gain over previous algorithms that is exponential on the spacings between binding sites. Experimental results show that the algorithm is much faster than existing ones, sometimes by more than two orders of magnitude. The application of the method to biological datasets shows its ability to extract relevant consensi.

1 Introduction

In this large-scale genome sequencing era, the main bottleneck to the progress in molecular biology is data analysis. The aim of this analysis is the extraction of biological information from genome sequence data. An important task in this context consists in the detection of regulatory signatures in DNA sequences as well as the prediction of the corresponding

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

†Visiting Research Fellow at King's College London, UK.

‡Partially supported by CNRS-INRIA-INRA-INSERM action BioInformatique and Wellcome Trust Foundation.

promoter. An important part of gene regulation is mediated by specific proteins, called the *transcription factors* (TF), which influence the transcription of a particular gene by binding to sequence specific sites on the DNA sequence, called *transcription factor binding sites*. Such binding sites are located in *promoter regions*. In prokaryotic organisms, the binding sites are predominantly in the immediate vicinity of the gene. However, in higher eukaryotes, the binding sites of cooperating TFs are organized into short sequence units called *cis-regulatory modules* (CRM). We refer the reader to [13] for additional details.

The first methods for detecting promoter regions in DNA sequences [7, 11] looked only for a unique binding site – *single motif*. In the search for more complex promoter models methods have appeared that extract promoter regions composed by two binding sites [3, 12]. The first attempts to identify several binding sites – *multiple motif* – consisted on crossing compatible single motifs [2, 5, 4], 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 by 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 directly lies 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 [9, 8]. 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 [6] 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 genomic data since it explodes both in time and in space. The prime objective of this paper is to attack the explosion problem of this approach.

The main contribution of this work is a new algorithm to identify cis-regulatory modules 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 [6]. 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], which is a suffix tree [10] built only up to a certain level, leading to an important space saving.

We demonstrate our results on simulated and real data. In the first case, the goal was to test the ability of the algorithm to deal with large amounts of human simulated data. In the second case, we wanted to verify the accuracy of the algorithm in recovering known single and multiple motifs in yeast, and to apply it to discover novel motifs.

2 Model overview

A *single model*, or simply a *model*, is a non-empty string over the alphabet Σ={A,C,T,G}. From this point on, we denote the length of a single model m by k. A model m is said to have an *e-occurrence* in the input sequences if there is one word u of length k in the input sequences such that the Hamming distance between u and m is less than or equal to e. Recall that the Hamming distance between two sequences of the same length is the minimum number of substitutions needed to transform one sequence into another. 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.

A *structured model* is a pair (m, d) where m is a p-tuple $(m_i)_{1\leq i\leq p}$ of single models and d is a $(p-1)$-tuple of pairs $(d_{\min_i}, d_{\max_i})_{1\leq i<p}$, denoting $p-1$ intervals of distance between the p single models. Each element m_i of the structured model (m, d) is called a *box* and we denote its length by k_i. 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 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 $1 \leq i < p$. 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 TF binding sites.

3 Factor tree

In this section we restrict our attention to a basic data structure upon which our algorithm is based. A *factor tree* is a data structure used to index strings, proposed by Allali and Sagot [1], which is very similar to suffix trees [10]. A factor tree, also called the *k-factor tree*, indexes the substrings of a string whose length does not exceed k. As for suffix trees, the time complexity for constructing a factor tree is linear in the length of the string. However, compared with a suffix tree, the k-factor tree offers a substantial gain in terms of space complexity for small values of k.

As an example, consider the 5-factor tree for a string s=AGACAGGAGGC\$ presented in Figure 1 (\$ is commonly used as a termination character to guarantee that all substrings of s which match a prefix of a substring of s have a path from the root that ends at a leaf, e.g., the substring C of s matches a prefix of the substring CAGGA). All substrings up to size

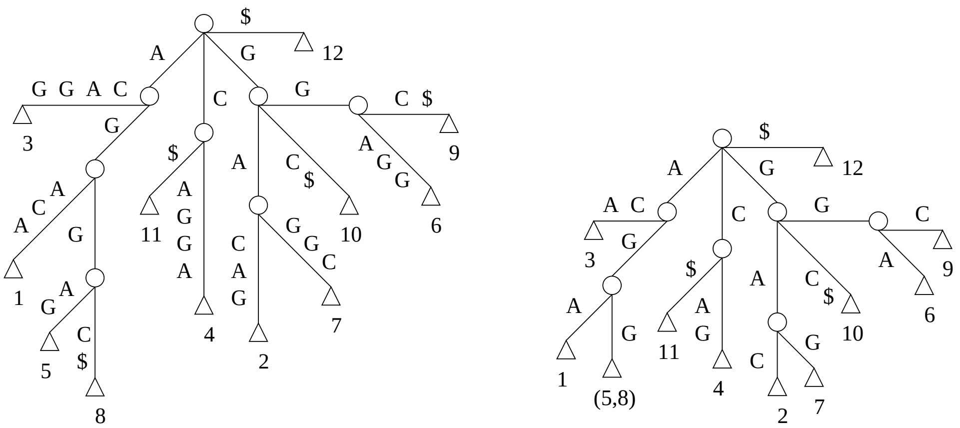

Figure 1: The 5-factor tree (on the left) and the 3-factor tree (on the right) for the string AGACAGGAGGC\$.

5 are depicted in the factor tree (e.g. AGACA, GACAG, etc.) and the corresponding leaf contains the positions where the substring occurs in the input string (the first position in the string is position 1). Note that the 5-deep factor tree does not have any leaf with a collapsed position, since there is no common substring of size 5 in the string s. However, with $k = 3$, the substring AGG occurs twice in s, at positions 5 and 8, and we obtain a 3-factor tree with collapsed positions, as depicted in Figure 1.

The factor tree construction for a set of N strings, called a *generalized factor tree*, can be easily obtained by consecutively building the factor tree for each string of the set. The resulting factor tree is built in time proportional to the sum of all the string lengths. An usual way to distinguish the input strings is by storing at each tree node v a Boolean array, called the $Colors_v$ array [7] (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.

4 Structured motif extraction

In this section we introduce the main contribution of this paper. A new data structure, called *box-link*, is proposed and used in a new algorithm for structured motif extraction. For the sake of exposition, we consider only structured motifs with p boxes of the same size k, same distance d between boxes, and a fixed number of allowed substitutions e for each box. The general case was studied and implemented. Furthermore, all complexity results lift naturally, but the full details are out of the scope of this paper.

4.1 Box-link

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 structured model. Formally, a box-link can be defined as follows. Let L 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. A *box-link of size* i, with $1 \leq i < p$, is a $(i+1)$-tuple in L^{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.

As an example, consider the 3-factor tree for AGACAGGAGGC$ presented in Figure 2 with box-links depicted for 2 boxes distanced by a spacer of 4 nucleotides.

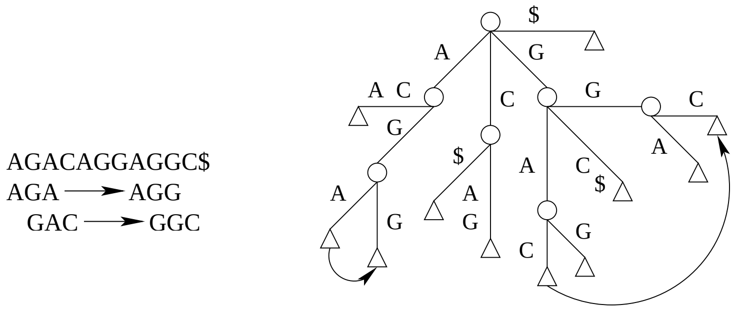

Figure 2: The 3-factor tree for the string AGACAGGAGGC$ with box-link for $p = 2$ and $d = 4$.

When considering a generalized factor tree for a set of N strings, a box-link b has to be endowed with a Boolean array of dimension N, similar to the one associated with factor tree nodes, defined as: $Colors_b[i] = 1$, if b links boxes of the i-th input sequence, and $Colors_b[i] = 0$, otherwise, with $1 \leq i \leq N$.

4.2 Box-links construction

In this section we present an algorithm to build box-links. The algorithm makes use of two variables. First, the variable $list_{leaf}$ has the list of all leaves inserted in the factor tree, which can be easily obtained during the factor tree construction. 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 average length n (the average length of an input sequence). Observe that the substring labeling the path from the root to the j-th leaf of $list_{leaf_i}$ corresponds to the j-th at most k-length substring of the i-th input string. Second, the variable b_j stores the j-size box-links being built. Moreover, we have to set up 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 box-link b and 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. The pseudo-code of the algorithm to build box-links is presented in Algorithm 1.

Algorithm 1 BoxLink(Boxes p, BoxSize k, BoxDistance d, ListLeaf $list_{leaf}$)

1. for i from 1 to N
2.   while size of $list_{leaf_i} \geq pk + (p-1)d$
3. $b_0 = \text{AddBoxLink}(nil, list_{leaf_i}[0], i)$
4.   for j from 1 to $p-1$
5. $b_j = \text{AddBoxLink}(b_{j-1}, list_{leaf_i}[jk + jd], i)$
6.   remove the first leaf of $list_{leaf_i}$

Next, we establish the time and space complexity for the BoxLink algorithm. We denote by n_l the number of nodes at depth l of the generalized suffix tree for the same input sequences as the factor tree where the box-links are being constructed (note that this suffix tree is never built, it only serves the purpose of providing us a value to establish the complexity analysis). Moreover, we define $b_p(k,d) = \min\{n_k^p, n_{pk+(p-1)d}\}$.

Proposition 4.1 Algorithm 1 takes $O(N^2np)$ time and $O(Nb_p(k,d))$ space.

Proof: Step 1 requires $O(N)$ time. Step 2 requires $O(n)$, where n is the average number of leaves in $list_{leaf_i}$. Step 4 requires $O(p)$ time. Step 5 requires $O(N)$ time, which corresponds to the creation or update of $Colors$ array. Hence, Algorithm 1 takes $O(N^2np)$ time. Briefly, the space complexity is given by the number of box-links (which is upper bounded by $b_p(k,d)$) times its size (which is upper bounded by N). □

4.3 Structured motif extraction using box-links

In this section we introduce the algorithm to extract structured motifs from upstream sequences of co-regulated genes. The basic data structure used by the algorithm is a factor tree. The first step is the construction of the generalized k-factor tree $\mathcal{T}$ for the input set of DNA sequences (recall that k is the size of the boxes in the structured models). Next, the tree is modified in order to store at each node the *Colors* array, as explained in Section 3. The pre-processing phase ends with the construction of box-links, which are added to the leaves of the factor tree. At this point we are in condition to start the extraction phase, but before presenting the pseudo-code of the algorithm we need to introduce the following concept. A *e-node-occurrence* of a model m is a pair (v, e_v) such that: (i) v is a node in the factor 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. Clearly, when substitutions are allowed ($e > 0$) a model can have more that one node-occurrence in $\mathcal{T}$.

We now give a summary of the extraction process for $p = 2$. The pseudo-code for the extraction is presented in Algorithm 2. The algorithm recursive process is initialized with ExtractMotifs($m = \varepsilon, i = 1$), where ε represents the empty model. The algorithm starts by extracting single motifs m_1 of length k, one at a time. The extraction of single motifs is done by a simple depth-first traversal of the factor tree $\mathcal{T}$ (step (7)) [7]. Since $i = 1 < p = 2$ a recursive call is made with ExtractMotifs($m = m_1, 2$) (step (8)). For each node-occurrence v of a first box m_1 (step (1)), box-links are followed to reach nodes z (step (2)) and the content of the Boolean array *Colors* stored in these box-links is used to temporarily and partially modify the *Colors* of the target nodes z (steps (3) to (6)). The extraction of the second box m_2 then proceeds in the same way, but only over this modified part of the tree (step (7)). Once the extraction of all valid motifs $\langle(m_1, m_2), d\rangle$ has ended (step (9)), the factor tree is restored to its previous state (step (10)). The construction of another single motif m_1 follows (step (7)), and the whole process unwinds in a recursive way until all structured motifs are extracted.

Algorithm 2 ExtractMotifs(Motif m, Box i)

---

1. for each node-occurrence v of m
2.   for each leaf z such that there is a box-link $b_{\langle v,z\rangle}$ from v to z
3.   put z in $L(i)$
4.   if (first time z is reached) set $Colors_z$ to $\overrightarrow{0}$ and put z in $NextEnds$
5. $Colors_z = Colors_z + Colors_{b_{\langle v,z\rangle}}$
6. UpdateTree($\mathcal{T}$,$NextEnds$)
7. for each motif m_i obtained by a depth-first traversal of $\mathcal{T}$
8.   if ($i < p$) ExtractMotifs($m = m_1 \ldots m_i$,$i + 1$)
9.   else KeepMotif($m = \langle(m_1, \ldots, m_p)$,$d\rangle$)
10. RestoreTree($\mathcal{T}$,$L(i)$)

A proper appreciation of the algorithm needs some set up that follows. The 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 $NextEnds$. Moreover, $L(i)$ is an array that stores the state of the nodes at level k for the $(i-1)$-th box of a structured model. The 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. Finally, KeepMotif stores all information concerning valid motifs.

Next, we establish the complexity of Algorithm 2. The term $\nu(e,k)$ denotes the number of distinct words that are at a Hamming distance at most e from a k-long word.

Proposition 4.2 Algorithm 2 takes $O(Nb_p(k,d)\nu^p(e,k))$ time and $O(Nb_p(k,d) + Npn_k)$ space.

Proof: We can parcel out the complexity into: (i) the number of operations needed to build all p parts of structured motifs; (ii) the number of operations needed to update $\mathcal{T}$; (iii) the number of operations needed to restore $\mathcal{T}$. To compute (i) we have to calculate the cost of all visits to nodes between the root and level k (the deeper level ever reached). Notice that when spelling all parts of a motif we are working with nodes between the root and level k only, and because factor trees are compact, being at least binary, there are at most $2n_k$ such nodes. Hence, the number of visits to nodes between the root and level k is upper bounded by twice the number of visits to nodes at level k. Moreover, when no substitutions are allowed, there are at most $b_p(k,d)$ ways of spelling all structured motifs. However, when up to e substitutions are allowed, a node at level k may be visited $O(\nu^p(e,k))$ times more. Hence, (i) takes $O(Nb_p(k,d)\nu^p(e,k))$, where N accounts for the access to the $Colors$ array. To compute (ii) we need to count the number of operations necessary to modify the first k levels of $\mathcal{T}$ which is upper bounded by $O(Nb_p(k,d)\nu^{p-1}(e,k))$. This corresponds to all visits made to nodes z coming from $b_{\langle v,z\rangle}$ 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. Finally, since (iii) is also upper bounded by the time to update $\mathcal{T}$, we conclude that Algorithm 2 takes $O(Nb_p(k,d)\nu^p(e,k))$ time. Briefly, the space complexity is given by the space required by box-links and the space required by the factor tree and the $L(i)$ arrays $(1 \leq i < p)$. $\square$

This algorithm exhibits an exponential time and space gain relatively to the previous approaches to extract structured motifs presented in [6], where the best algorithm takes $O(Nb_p(k,d)\nu^p(e,k) + Nn_{pk+(p-1)d}\nu^{p-1}(e,k))$ time[1] and $O(Nn_{pk+(p-1)d} + Npn_k)$ space. The difference between the time complexity expressions occurs in the second term which is eliminated by Algorithm 2. Observe that in the worst case scenario, the 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 $|\Sigma|^{(p-1)d}$ (we denote by $|\Sigma|$ the cardinality of Σ, so for the DNA alphabet case $|\Sigma| = 4$). 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 (recall that an exponential factor on $(p-1)d$ disappears in the worst case analysis). A similar reasoning applies for the exponential gain in space complexity.

[1]The time complexity presented herein differs from the one presented in [6]. Indeed, the authors of [6] acknowledged a mistake in the time complexity and agree with the expression we present here.

5 Experimental results

This section presents results attained by RISO, the C implementation of the new algorithm proposed in this paper, as well as benchmark comparisons with SMILE, the C implementation of the algorithm presented in [6] made available by the authors. The results presented were obtained using a Intel Pentium IV at 2.4GHz with 1GB of RAM.

The structured motifs extracted by RISO were classified according to their statistical significance in order to give them some biological relevance [6]. The Z-score for a motif was considered, indicating how far and in what direction the number of occurrences of the motif deviates from its distribution mean. This score is especially useful when comparing the relative occurrences of motifs from distributions with different means and different standard deviations. This difference clearly occurs for highly conserved regions on a set of DNA sequences against ordinary uniformly distributed sequences (random background).

5.1 Human simulated data

As a basic test of RISO, we used a set of DNA sequences generated by a Markov chain with order 5, calibrated on intergenic oligonucleotide frequencies for the human DNA. These sequences were obtained taking advantage of the Regulatory Sequence Analysis (RSA) tools (`http://rsat.scmbb.ulb.ac.be/rsat/`). In this context, two datasets were prepared, one with 1000 sequences of size 1000, and another with 2000 sequences of size 2000. Four sets of experiments were performed by running RISO and SMILE over each dataset, requiring a quorum of 20% ($q = 200/1000$) and 40% ($q = 800/2000$) for each data set, respectively. Each set of experiments consisted on three searches for structured motifs of two boxes with sizes 3, 5, and 7, respectively, in all cases distanced by 15 nucleotides. Allowed substitutions varied in the experiments: in the first search $e_1 = 0$ and $e_2 = 0$; in the second search $e_1 = 1$ and $e_2 = 1$; in the third search $e_1 = 2$ and $e_2 = 1$. In Figure 3 we depict the results obtained in these experiments. We emphasize that time results encompass the construction of the factor

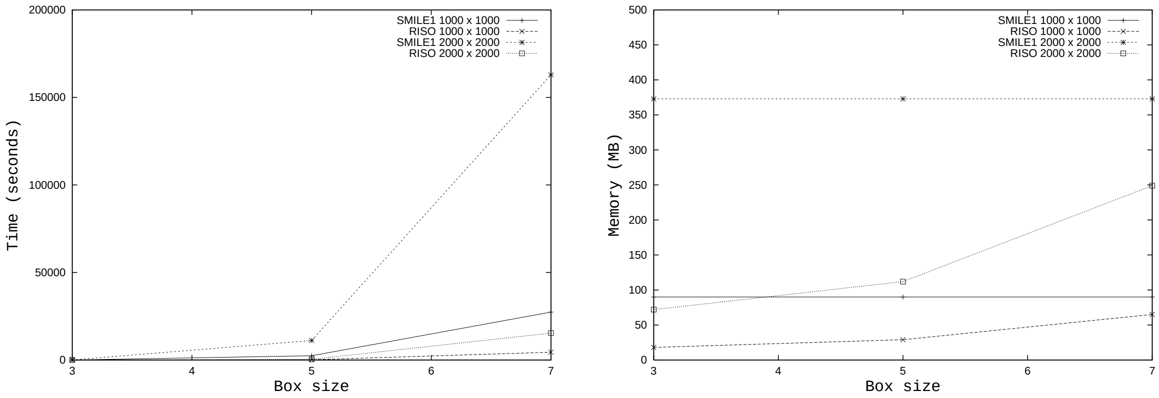

Figure 3: Time (on the left) and space (on the right) comparison of RISO and SMILE.

tree with box-links (for the RISO case), which took, in the computationally most demanding experiment, less than 5 seconds. In terms of time RISO was always much faster than SMILE (with both RISO curves lying below SMILE curves). RISO showed to be scalable with respect to dataset size increase (RISO performed 3.5 times slower when dataset size increased 4 times, while SMILE performed 6 times slower with the same dataset size). In terms of space RISO also showed a better performance. Moreover, SMILE was not able to cope with a dataset of 4000 sequences of size 4000 in a 1GB memory machine, while RISO was able to perform extractions over this dataset.

5.2 Cis-regulatory regions in yeast

In real data we evaluated the performance of the algorithm to recover known motifs in yeast, as well as to discover new ones. As test sets we used a collection of 68 genes that are known to be regulated by zinc cluster factors. The upstream sequences were retrieved from positions -1 to -1000 relative to the ORF (open-reading frame) start positions. To set up our data we took advantage of the TRANSFAC database (`http://www.gene-regulation.com/`). We made several extractions not allowing substitutions ($e = 0$) and requiring a quorum of 10% ($q = 7/68$) in the collected data. Our method was able to detect 6 CRMs with very high significance out of 9 that we looked for. The results are summarized in Table 1. The first column of the

Table 1: Regulons of Zn cluster proteins.

TF name	Known motif	Predicted motifs	Z-score	Ranking
GAL4	CGGn_{11}CCG	CGGn_{11}CCG (GAL4)	**7.05**	1st
HAP1	CGGnnntanCGG	GGGn_3AGC	3.05	–
		CGGn_6CGG (HAP1)	2.08	–
LEU3	RCCggnnccGGY	GCCn_6GGT (LEU3)	**4.82**	6th
LYS	wwwTCCrnyGGAwww	TTCn_4GGA	3.05	–
PDR	tytCCGYGGary	TCCGCG	3.58	–
PPR1	wyCGGnnwwykCCGaw	CGGn_6CCG (PPR1)	**5.86**	1st
PUT3	yCGGnangcgnannnCCGa	CCGn_{11}GCC	3.05	–
UGA3	aaarccgcsggcggsawt	AGCCGCC	**7.59**	–
		GGCGGCTAA	**27.33**	2nd
UME6	tagccgccga	TAGCCGCC	**12.52**	11th
		GCCGCCGA	**12.52**	12th

table lists the searched TFs, while the known consensus for each TF is presented in the next column, using the standard IUPAC code. For known consensi, lowercase letters represent uncertainty with respect to the corresponding nucleotide, whereas capital letters denote a higher certainty. The third column presents the predicted motifs, where we emphasize the TFs GAL4, LEU3 and PPR1 because they were perfect matches (with respect to the known motifs) with very high significance. The Z-score is presented in the fourth column and its position in the ordered list of the scored motifs is presented in the last column. When no ranking is provided it means that the predicted motif was not in the fifteen best ranked. We stress that all the extractions performed above took less than one minute with RISO, and in the best case RISO was 375 times faster than SMILE.

6 Conclusion

We presented a new algorithm and data structure for the extraction of structured motifs in DNA sequences. The new algorithm exhibits an exponential time and space gain, in the worst case analysis, relatively to existing algorithms for extracting structured motifs [6]. 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 (called a factor tree [1]), saving much space in comparison with the algorithms proposed in [6] that are based on the full suffix tree. Experimental results show that the new algorithm is much faster than the SMILE algorithm [6], in some cases, more than two orders of magnitude faster. The application of RISO to biological datasets shows the ability of the method to extract relevant consensi.

Future work can progress in several directions. First, we are refining RISO in order to have a trade-off between computing some box-links and having others stored in memory, reducing even more the space used by the algorithm. Second, it would 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. Finally, we are 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 and M.-F. Sagot. The at most k-deep factor tree. Submitted for publication, 2003.

[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] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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