A parallel algorithm for the extraction of structured motifs

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 work we propose a parallel algorithm for the efficient extraction of binding-site consensus from genomic sequences. This algorithm, based on an existing approach, extracts structured motifs, that consist of an ordered collection of $p \geq 1$ boxes with sizes and spacings between them specified by given parameters. The contents of the boxes, which represent the extracted motifs, are unknown at the start of the process and are found by the algorithm using a suffix tree as the fundamental data structure. By partitioning the structured motif searching space we divide the most demanding part of the algorithm by a number of processors that can be loosely coupled. In this way we obtain, under conditions that are easily met, a speedup that is linear on the number of available processing units. This speedup is verified by both theoretical and experimental analysis, also presented in this paper.

Keywords: bioinformatics, suffix tree, structured motifs, grid computing, parallel algorithm, complexity

1 Introduction

The large-scale genome sequencing era brought important computational challenges for the analysis of nucleotide sequences. Gene prediction is a crucial task in this context and is based in the recognition of coding regions as well as in the prediction of the corresponding promoter. The promoter is an integral part of the gene that mediates and controls the initiation of transcription and encompasses three regions, each one containing several sequences called the promoter binding sites. The first one, the core promoter, is the region that suffices to

[*]Visiting Research Fellow at King's College London, UK.
[†]Partially supported by CNRS-INRIA-INRA-INSERM action BioInformatique and Wellcome Trust Foundation.

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 sequence that can be located farther upstream from the core promoter and its main function is to stimulate transcription.

The DNA sequences involved in promoter function were first identified by comparisons of the nucleotide sequences of a series of different genes isolated from *E. coli.* These comparisons revealed that the region upstream of the transcription initiation site contains sets of sequences that are similar in a variety of genes. Since then, consensus extraction has been addressed in a variety of ways. Recent approaches have confronted the problem of extracting site consensi [2, 15, 17] or considered the fact that binding sites often come together in a well ordered and regularly spaced manner [11, 5, 3, 10, 18].

This paper presents a parallelized algorithm, based on an existing sequential approach [11], to extract binding-site consensus. To reflect the fact that a promoter is fragmented in several binding sites this algorithm introduces the concept of structured motifs. A structured motif is described as an ordered collection of $p \geq 1$ boxes, a maximum allowed error for each box, and an interval of distance for each pair of consecutive boxes. The contents of the boxes, the motifs themselves, are unknown at the start of the algorithm. A suffix tree is used to find such motifs. Optimal work complexity of the parallel version is only achieved if structured motifs extraction is balanced over the available processing units. Hence, the main issue of the parallelization is to define a balanced partition of the structured motifs searching space. This space can be represented by a lexicographic trie, and in some special cases, by the suffix tree of the input sequences.

Finding a balanced partition of a tree, and in particular of a suffix tree, is an issue of the utmost importance. This follows from the fact that many search algorithms are based on suffix trees and a parallel balanced solution can help solving the increasing amount of data incoming from the bioinformatics community. There has already been a considerable effort on the development of algorithms for building suffix trees using distributed approaches [8, 1]. However, the main goal of these algorithms is to reduce the time complexity of the suffix tree construction, and this step is by far the fastest of the sequential algorithm we are going to parallelize. Other approaches have been taken to solve the problem of suffix tree construction for the whole genome [14, 9]. Herein, the purpose is to build suffix trees on disk in order to amortize the construction cost over many thousand searches. This question is addressed with an algorithm to build consecutive partitions of the suffix tree, where the partition size depends on the available memory, and compose them together on disk to proceed with searches. Again, this is not the goal of our parallelization. Since the extraction step of the structured motifs algorithm is the most time consuming, we want the search tree space partitioned among the available processors, to proceed in each one with a separate extraction. Moreover, we want a balanced partition of the search tree space over all the available processors that is not necessarily imposed by memory constraints.

In our approach, we abstract the problem of finding a balanced partition of a tree to a generalization of the 3-PARTITION problem [6], which we call PARTITION UP TO ε problem. We show that this new problem is NP-complete in the strong sense and can not be straightforwardly reduced to 3-PARTITION. This indicates that the PARTITION UP TO ε problem is interesting per se. Furthermore, unlike 3-PARTITION, the PARTITION UP TO ε problem has an optimization version for which we propose an approximation algorithm. This approximation algorithm is applied to obtain a partition of the search tree space among the available processing units and it is the core of the proposed parallelization.

The parallel version of the algorithm is implemented using grid technology. Nevertheless, it is worthwhile to notice that the algorithm is designed in the CREW-PRAM computational model [4] and therefore any realization of this model will support the algorithm. Moreover, our solution does not require the grid nodes to communicate, which boosts the overall performance. We obtain as a simple corollary that our algorithm can be made work-efficient, with respect to the sequential algorithm, under easily achievable conditions.

In Section 2 we give a brief presentation of suffix trees. In Section 3 we present the sequential algorithm to extract structured motifs. In Section 4 we establish the PARTITION UP TO ε problem and present an algorithm to approximate the optimization version of this problem. In Section 5 we apply this algorithm to the practical case of parallel structured motifs extraction and we present some experimental results in Section 6.

2 Suffix trees

A suffix tree is a data structure built over all the suffixes of a string. Such a data structure exposes the internal structure of a string and is often used to solve many string problems in linear-time. The construction of a suffix tree in linear-time is a problem already addressed by Weiner in 1973 [19], by McCreight in 1976 [12], and more recently by Ukkonen [16] in 1995.

We define a suffix tree for an arbitrary string S of length n over an alphabet Σ as presented in [7]. After that we generalize the suffix tree to handle sets of strings.

Definition 2.1 A *suffix tree* of a n-character string S is a rooted directed tree with exactly n leaves, numbered 1 to n. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf i, the label of the path from the root to the leaf i exactly spells out the suffix of S that starts at position i.

The previous definition of a suffix tree does not guarantee the existence of a suffix tree for any string S. The problem is that if a prefix of a suffix of S matches a suffix of S, the path for the later suffix would not end at a leaf. To avoid this problem we place at the end of S a special symbol that is not in the alphabet. In this paper we use the symbol \$ for the termination character.

The suffix tree construction for a set of strings, called a generalized suffix tree, can be easily achieved by consecutively building the suffix tree for each string of the set. The resulting suffix tree is built in time proportional to the sum of all the string lengths. The leaf number of the single string suffix tree can easily be converted to two numbers, one identifying the string and the other identifying the starting position in that string. For example, the generalized suffix tree for the strings S_1=TACTA and S_2=CACTCA is presented in Figure 1.

3 Structured motifs extraction

The problem of structured motifs extraction [11] addresses the extraction of consensus motifs that appear together in a well-ordered and regularly spaced manner. A structured motif can be described as an ordered collection of $p \geq 1$ boxes, a maximum allowed error for each box, and an interval of distance for each pair of consecutive boxes. A suffix tree is used to find such

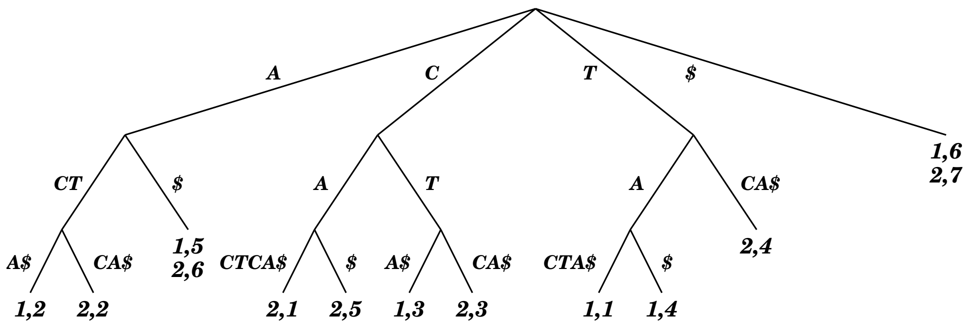

Figure 1: Generalized suffix tree for S_1=TACTA and S_2=CACTCA.

motifs in a set of N input sequences. We need to modify the suffix tree in order to store at each tree node v a Boolean array of size N, denoted by $colors_v$ [13], indicating the sequences in the input set that contains the string labeling the path from the root to the tree node v.

To set up the algorithm to extract structured motifs we have to introduce some notation:

- A *model* is an element in Σ^+. 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[1] between u and m is less than or equal to e.
- A model is said to be a *valid model* if it has an occurrence in at least q input sequences, where q is called the *quorum*.
- A *node-occurrence* of a model m is represented by a pair (v, e_v) where v is a tree node and $e_v \leq e$ is the Hamming distance between the label of the path from the root to v and m.
- A *structured model* is a pair (m, d) where:
  - m is a p-tuple of single models $(m_i)_{1\leq i\leq p}$, denoting the p boxes;
  - d is a $(p-1)$-tuple of triplets $(d_{\min_i}, d_{\max_i}, \delta_i)_{1\leq i\leq p-1}$, denoting the $p-1$ intervals of distance.

  The terms $d_{\min_i} \leq d_{\max_i}$ represent a minimum and maximum allowed distance between the parts and δ_i an allowed interval around that distance. When $\delta_i = (d_{\max_i} - d_{\min_i} + 1)/2$, δ_i is omitted.
- A structured model (m, d) is said to be a *valid structured model* if for all $1 \leq i \leq p-1$ and for all occurrences u_i of m_i, there exist occurrences $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_p$ of the single models $m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_p$ such that:
  - $u_1, \ldots, u_p$ belong to the same input sequence;
  - there exists d_i, with $d_{\min_i} + \delta_i \leq d_i \leq d_{\max_i} - \delta_i$, such that the distance between the end position of u_i and the start position of u_{i+1} in the sequence is in $[d_i - \delta_i, d_i + \delta_i]$;
  - d_i is the same for p-tuples of occurrences present in at least q distinct sequences.

[1]The Hamming distance between two sequences is the minimum number of substitutions to transform one sequence into another.

We are now able to describe the sequential algorithm to extract structured motifs [11]. This algorithm is based on a previously published algorithm to extract single motifs [13]. For the sake of exposition, we assume that all boxes of the structured motifs have the same size k and maximum allowed error e, the number of boxes p equals 2 and δ equals $(d_{max}-d_{min}+1)/2$.

The extraction of single motifs is done by a simple depth-first traversal of the suffix tree $\mathcal{T}$ of the input sequences. When errors are allowed this traversal is more complex since 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 $\mathcal{T}$. We denote by $\mathcal{M}$ the lexicographic trie of all these models pruned at the nodes where the quorum is no longer verified. In practice, $\mathcal{M}$ is never built but can be virtually traversed by a more complex traversal over $\mathcal{T}$. Observe that if no errors are allowed and the quorum equals 1 then $\mathcal{M}$ and $\mathcal{T}$ present the same models.

The extraction of structured motifs starts by extracting single valid models of length k. Once a single valid model m_1 is obtained the extraction of all single models m_2 with which m_1 could form a structured model $((m_1, m_2), (d_{\min}, d_{\max}))$ starts. Note than in this case and when the quorum equals 1, $\mathcal{M}$ corresponds to the lexicographic trie of all sequences of size $2k + d_{\min}$ to $2k + d_{\max}$ where the two k-length boxes are at most at distance e from suffixes of the tree $\mathcal{T}$. The pseudo-code of the algorithm can be found in Figure 2.

```
ExtractModels(suffix tree $\mathcal{T}$)
1. find all single valid models $m_1$ on $\mathcal{T}$
2. for each model $m_1$ found
3.     for each node-occurrence $(v, e_v)$ of model $m_1$ on $\mathcal{T}$
4.         put in PotentialStarts the children $w$ of $v$ at levels $k + d_{\min}$ to $k + d_{\max}$
5.     for each node $w$ in PotentialStarts
6.         find all single valid models $m_2$ starting at node $w$ of $\mathcal{T}$
7.         report the structured model $((m_1, m_2), (d_{\min}, d_{\max}))$ as a valid model
```

Figure 2: ExtractModels algorithm extracts structured motifs from $\mathcal{T}$.

It is shown that for two boxes [11], the ExtractModels algorithm takes $O(Nn_{2k+d_{\max}}\nu^2(e,k))$ time, where $n_{2k+d_{\max}}$ is the number of tree nodes at depth $2k+d_{\max}$ and $\nu(e,k)$ is the number of distinct words that are at a Hamming distance at most e from a k-long word. It is easy to see that the following upper bound for $\nu(e,k)$ holds:

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

In general, the ExtractModels algorithm for p boxes requires

$$O(Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)) \tag{1}$$

time.

4 Balanced partition

The problem of determining a balanced partition of a lexicographic trie can be abstracted to the following general problem.

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

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

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

Theorem 4.1 The PARTITION UP TO ε problem is NP-complete in the strong sense.

Proof: Clearly this problem is in NP. Take as witness space the set of all r-partitions made with z gold bars. Consider as the polynomial time verifier the program that checks whether a witness is a partition up to ε. Then, if there is one partition up to ε, with z gold bars, take it as witness. Moreover, if there is no partition up to ε, with z gold bars, then there is no witness for which the polynomial time verifier will answer yes. Finally, to prove the strong NP-completeness we transform the 3-PARTITION problem [6], which is NP-complete in the strong sense, to the PARTITION UP TO ε problem. Consider the set $A = \{a_1, \ldots, a_{3m}\}$ as an arbitrary instance of the 3-PARTITION and make an instance of the PARTITION UP TO ε where $r = m$, $\ell = 3m$, $w_j = a_j$ with $1 \leq i \leq 3m$, $\varepsilon = 0$, $z = 3m$ and $c = 1$. This transformation can clearly be performed in time polynomial in the input length alone. Furthermore, the length and the largest number of the constructed instance remain the same as in the given 3-PARTITION instance. Finally, there is a solution for the 3-PARTITION problem iff there is a solution for the PARTITION UP TO ε problem. We can conclude that we have a pseudo-polynomial transformation and so the PARTITION UP TO ε problem is strongly NP-complete. $\square$

Note that the previous result about strong NP-completeness guarantees that PARTITION UP TO ε problem cannot be solved by a pseudo-polynomial time algorithm, unless P=NP. Next we propose an approximation algorithm to solve this problem. The main goal is to find an optimal balanced r-partition of the gold bars up to some positive weight ε. Since there is no hope to find an efficient optimal solution we consider a trade-off between optimal solution presentation and optimal number of cuts. In our algorithm the balanced partition is defined as a family of intervals $(I_i)_{1\leq i\leq r}$. This feature has the main advantage that it is straightforward to check whether a gold bar belongs to a certain person. On the other hand, our solution cuts successively all gold bars, which is not necessarily optimal. The algorithm to determine the ith partition set I_i is presented in Figure 3.

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

Figure 3: SimpleCut algorithm approximates the PARTITION UP TO ε problem.

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

$$z = \ell \times c^t \tag{2}$$

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

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

Next we establish the time complexity and a ratio bound result [4] for the algorithm at hand, but first notice that the value t can be computed as

$$t = \lceil \tfrac{\log(\max w_j) + \log(\frac{1}{\varepsilon})}{\log(c)} \rceil. \tag{3}$$

Proposition 4.2 The SimpleCut algorithm requires $O(\ell)$ time.

Proof: In the first step of the SimpleCut algorithm the value t can be computed as given by (3) in constant time. In the second and third steps we determine ℓ intervals, but notice that the upper limit of one interval is the lower limit of the next, so in the end we only have to compute ℓ values, each one taking $O(1)$ time. Hence, the time complexity of these steps are $O(\ell)$. In the fourth step the ℓ summations are done in $O(\ell)$ time. In the seventh step we only have to compute the interval I'_i for the correspondent partition set i, in constant time. In the last step we have to determine in which interval V_j the variable x belongs, which takes $O(\log \ell)$ time. All the other operations of this step are $O(1)$. Hence, we can conclude that the overall time complexity of the algorithm SimpleCut is $O(\ell + \log(\ell)) = O(\ell)$. $\square$

Proposition 4.3 The SimpleCut algorithm has a ratio bound $\rho(\ell, r, (w_j)_{1 \leq j \leq \ell}, c, \varepsilon) = O(\frac{\max w_j}{\varepsilon})$.

Proof: By (2) and (3) the SimpleCut algorithm returns a total number of gold bars given by

$$z = O(\ell \times c^t) = O(\ell \times c^{\frac{\log(\max w_j) + \log(\frac{1}{\varepsilon})}{\log(c)}}) = O(\ell \times c^{\frac{\log(\frac{max w_j}{\varepsilon})}{\log(c)}}) = O(\ell \times 2^{\log(\frac{\max w_j}{\varepsilon})}) = O(\frac{\ell \max w_j}{\varepsilon}).$$

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

5 Parallel structured motifs extraction

We apply the previous algorithm to establish a balanced partition of the lexicographic trie $\mathcal{M}$ of models. As mentioned in Section 3, the trie $\mathcal{M}$ is never built, but is virtually used to define a partition for the extraction of structured motifs from the suffix tree $\mathcal{T}$ of the input sequences.

In Section 5.1 we describe and illustrate how to use the SimpleCut algorithm to establish a balanced partition of the trie $\mathcal{M}$. In Section 5.2 we present the parallel algorithm that partitions the extraction of structured motifs over $\mathcal{T}$ capitalizing on a balanced partition over $\mathcal{M}$.

5.1 Tree partition

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

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

Before adapting the SimpleCut algorithm to set up a partition of the lexicographic trie $\mathcal{M}$ note that the depth of $\mathcal{M}$ is finite and upper bounded by the sum of the length of all boxes. Hence, the depth $t+1$ to which we have to descend to define the partition might be larger than the depth of $\mathcal{M}$. Therefore, we have to ensure that $t+1$ does not exceed the depth of $\mathcal{M}$. In order to fulfill this condition we replace the first step by the one presented in Figure 4.

```
SimpleCut(partition set $i$, alphabet size $\ell$, grid nodes $r$, weights $(w_j)_{1\le j\le \ell}$, alphabet size $c$, work overload $\varepsilon$)
    1. let $t = \min(depth(\mathcal{M}) - 1, t')$ where $t'$ is the smallest integer such that $\frac{max\ w_j}{c^{t'}} \le \varepsilon$
```

Figure 4: First step of the SimpleCut algorithm applied to tree partition.

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

In the first step of the algorithm we obtain $t = 1$. The value of $t+1$ give us the depth where the tree partition is going to be made. In Figure 5 we can see a tree with nodes at depth $t+1$ as targets to determine a balanced distribution among the r grid nodes.

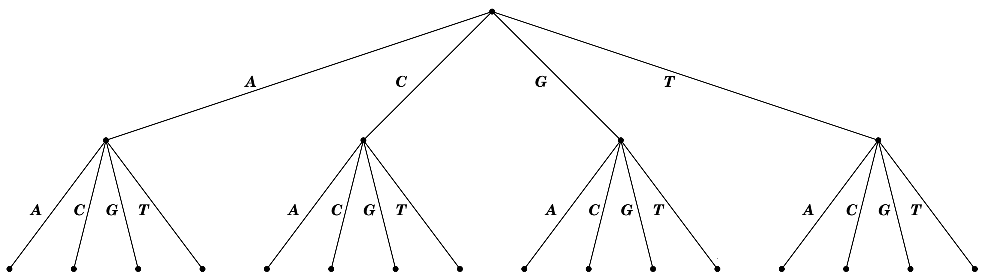

Figure 5: Tree cut at depth $t+1$=2.

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

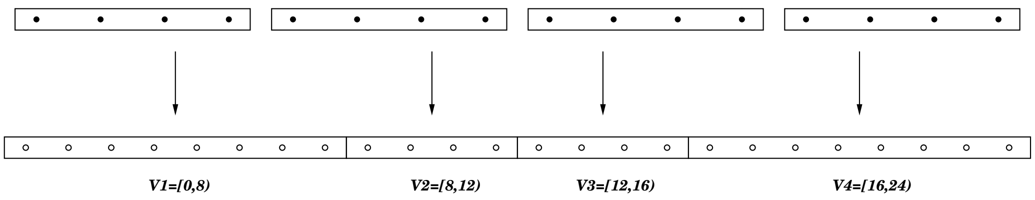

Figure 6: Intervals V_j in the virtual tree space.

In the next four steps, from 3 to 6, we compute the auxiliary set I'_i that corresponds to the partition set on the virtual tree node space for the ith grid node. Observe that to determine the width of each interval I' in this virtual space we compute $\delta = (6 \times 4^1)/5 = 4$ and to determine the number of grid nodes that are going to be overloaded with at most $\varepsilon = 1$ weight we compute $\gamma = (6 \times 4^1) \mod 5 = 4$. Since $\gamma = 4 > 0$ there is an imbalance among the grid nodes, and in this particular case, there are $\gamma = 4$ grid nodes with at most an overload $\varepsilon = 1$ over the remaining grid node. In Figure 7 we see that each grid node has the following interval $I'_1 = [0, 5)$, $I'_2 = [5, 10)$, $I'_3 = [10, 15)$, $I'_4 = [15, 20)$ and $I'_5 = [20, 24)$ of virtual nodes.

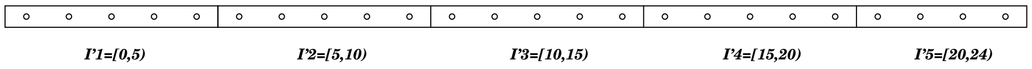

Figure 7: Intervals I'_i in the virtual tree space.

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

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

The intervals I_i computed directly from the function f defined above and the intervals I'_i are represented in Figure 8.

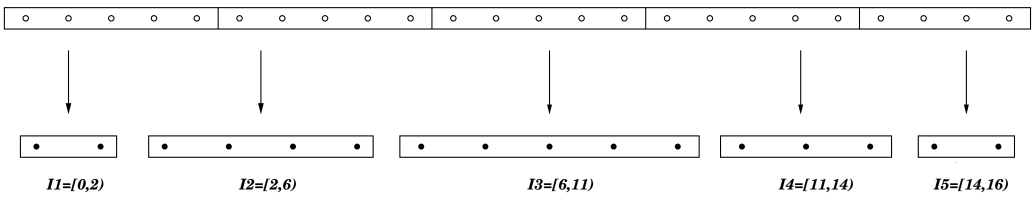

Figure 8: Intervals I_i in the tree node space.

From interval I_i it is straightforward to determine the tree partition assigned to the ith grid node. For instance, grid node number 3 is going to extract models with prefixes CG, CT, GA, GC and GG. The tree that is going to be considered by each grid node is represented in Figure 9.

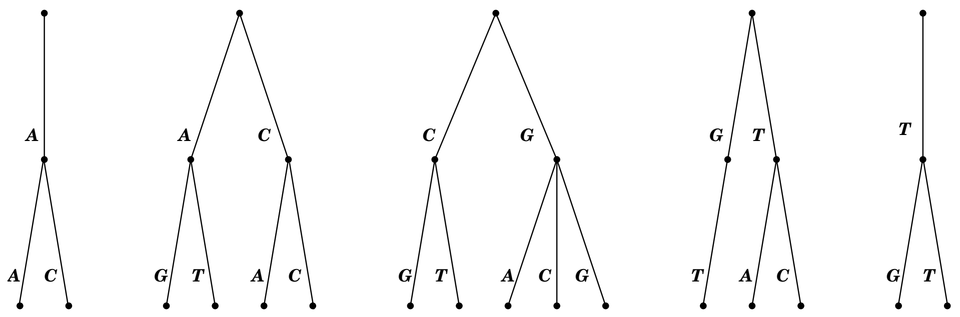

Figure 9: Tree partition up to ε.

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

5.2 Parallel extraction

Herein we describe how a partition of the lexicographic trie $\mathcal{M}$ of models, obtained as described in Section 5.1, can be used to define a partition for the extraction of structured motifs over the suffix tree $\mathcal{T}$ of the input sequences.

The structured motifs that are going to be extracted in each grid node are dictated by the correspondent partition set of the trie $\mathcal{M}$. The extraction itself is going to be made over the full suffix tree $\mathcal{T}$. It is important to notice that the full suffix tree $\mathcal{T}$ is going to be built in all grid nodes. This follows from the fact that the algorithm to extract single motifs [13] might need to traverse all the tree $\mathcal{T}$ to check whether a model is valid. A better solution would be to build only the part of $\mathcal{T}$ necessary to check the validity of the models assigned to each grid node. However, in this first approach to parallelize the structured motifs extraction, we do not address the optimal space parallel complexity problem. In what concerns time, the complexity of the extraction is by far larger than the complexity of building $\mathcal{T}$. Therefore, we do not waste too much time by forcing each grid node to compute the full suffix tree.

The modified algorithm to extract structured models of the suffix tree $\mathcal{T}$ in the ith grid node is presented in Figure 10.

Note that in both the first step and the sixth step of the PExtractModels algorithm it is necessary to check whether a model belongs to I_i. If the partition set I_i is not an interval it might be quite hard to check the conditions on those steps. This is why we decided that the SimpleCut algorithm should return intervals as partition sets. Once again we stress that this algorithm was drawn having in mind a tradeoff between finding optimal number of cuts and finding optimal representations of the partition sets.

We are now able to present the parallel algorithm that runs in each grid node i to extract the structured motifs. The pseudo-code of the algorithm is presented in Figure 11.

Before showing that PSmile is work-efficient we have to make some modifications to the algorithm. Observe that in the worst case scenario all tree leaves extract structured motifs.

```
PExtractModels(suffix tree $\mathcal{T}$, partition set $I_i$ of $\mathcal{M}$)
  1. find all single valid models $m_1 \in I_i$ on $\mathcal{T}$
  2. for each model $m_1$ found
  3.         for each node-occurrence $(v, e_v)$ of model $m_1$ on $\mathcal{T}$
  4.                 put in PotentialStarts the children $w$ of $v$ at levels $k + d_{\min}$ to $k + d_{\max}$
  5.         for each node $w$ in PotentialStarts
  6.                 find all single valid models $m_2$ with $m_1 m_2 \in I_i$ starting at node $w$ of $\mathcal{T}$
  7.                 report the structured model $((m_1, m_2), (d_{\min}, d_{\max}))$ as a valid model
```

Figure 10: ExtractModels algorithm extracts structured motifs in I_i from $\mathcal{T}$.

```
PSmile(grid node $i$, work overload $\varepsilon$)
  1. compute weights $(w_j)_{1 \le j \le |\Sigma|}$;
  2. build suffix tree $\mathcal{T}$;
  3. create colors on $\mathcal{T}$;
  4. let $I_i$= SimpleCut$(i, |\Sigma|, r, (w_j)_{1 \le j \le |\Sigma|}, |\Sigma|, \varepsilon)$;
  5. call PExtractModels($\mathcal{T}$, $I_i$);
```

Figure 11: PSmile algorithm extracts structured motifs.

For this reason, the estimation of the frequencies and the computation of the weights done at the first step of PSmile may lead to a partition that produces worse time complexity results than a uniform partition. Hence, in order to achieve work-efficiency, we replace the first step of PSmile by the step presented in Figure 12. Moreover, we assume that the alphabet Σ is

```
PSmile(grid node $i$, work overload $\varepsilon$)
  1. let $w_j = 1$ for $1 \le j \le |\Sigma|$;
```

Figure 12: First step of the uniform partition version of PSmile.

fixed, a common assumption when measuring time complexity of algorithms involving suffix trees. Then, by Proposition 4.2, the time complexity of the SimpleCut algorithm can be expressed as $O(1)$. Furthermore, since both Σ and w_j for $1 \le j \le |\Sigma|$ are constants, w is also constant and we obtain the following result.

Proposition 5.1 The parallel algorithm PSmile is work-efficient with respect to the sequential version when $r = O(\nu^{\frac{p}{2}}(e, k))$ and $\frac{\varepsilon}{w} \le \frac{1}{r}$.

Proof: We start by computing the time complexity of PSmile for each grid node. The first step is $O(1)$ since $w_j = 1$, for all j, and the alphabet is fixed. The second step requires $O(Nn)$ time, where n is the average length of the input sequences, using any linear time suffix tree construction algorithm [19, 12, 16]. The time complexity of the third step is $O(Nn_{pk+(p-1)d_{\max}})$, as shown in the original paper [13]. The fourth step requires $O(1)$ time. Finally, to determine the time complexity of the fifth step, notice that in the worst case scenario $|\Sigma|^{pk}$ is the total number of models. Moreover, the partition of the models assigns to any grid node

$$\begin{aligned} & O(\tfrac{|\Sigma|^{pk}}{r} + |\Sigma|^{pk}\tfrac{\varepsilon}{w}) \\ = \; & O(2 \times \tfrac{|\Sigma|^{pk}}{r}) \qquad \text{(by hypothesis } \tfrac{\varepsilon}{w} \leq \tfrac{1}{r}) \\ = \; & O(\tfrac{|\Sigma|^{pk}}{r}) \end{aligned}$$

models. Furthermore, in the worst case where all possible models are valid, the time complexity associated with the extraction of structured motifs is directly proportional to the number of models extracted, and so by (1) presented in Section 3, each model takes

$$O(\tfrac{Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{|\Sigma|^{pk}})$$

and each grid node takes

$$\begin{aligned} & O(\tfrac{Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{|\Sigma|^{pk}}) \times O(\tfrac{|\Sigma|^{pk}}{r}) \\ = \; & O(\tfrac{Nn_{pk+(p-1)d_{\max}}\nu^p(e,k)}{r}) \\ = \; & O(Nn_{pk+(p-1)d_{\max}}\nu^{\frac{p}{2}}(e,k)) \qquad \text{(by hypothesis } r = O(\nu^{\frac{p}{2}}(e,k))) \end{aligned}$$

time to extract structured motifs. The complexity of the last step upper bounds the complexity of the rest of the algorithm. Thus, a grid node requires $O(Nn_{pk+(p-1)d_{\max}}\nu^{\frac{p}{2}}(e,k))$ time.

We conclude, by comparing with (1), that the overall work complexity is

$$O(Nn_{pk+(p-1)d_{\max}}\nu^{\frac{p}{2}}(e,k)) \times r = O(Nn_{pk+(p-1)d_{\max}}\nu^{p}(e,k))$$

and so PSmile achieves work-efficiency when $r = O(\nu^{\frac{p}{2}}(e,k))$ and $\frac{\varepsilon}{w} \leq \frac{1}{r}$. □

6 Experimental results

The aim of the present section is not to illustrate the extraction of structured motifs [11] but to describe the set up of the parallelized algorithm as well as a comparison between time results of the sequential and the parallel version of the algorithm, Smile and PSmile, respectively.

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

As test sets we used two groups of sequences. The first group contains only one test set with gene sequences of *S. cerevisiae.* This set is composed by 23 gene sequences, for a total of 23,000 nucleotides, encoding proteins that are up-regulated in yeast cells exposed to the herbicide 2,4-D, as assessed by quantitative proteomic analysis. Since this dataset belongs to an eukaryote the definition of the promoter or regulatory site models was more complex. In this work two different models, with two and three boxes, were tested with the objective of recognizing specific promoters, relying on combinations of individual elements. The second group is composed of three sets of non-coding sequences located between two divergent genes and extracted from the whole genomes of *B. subtilis*, *H. pylori* and *E. coli.* The first of these sets contains 1,062 sequences for a total of 196,736 nucleotides. The second set contains 1,1148 sequences and 226,928 nucleotides. The last set contains 308 sequences and 52,100 nucleotides. These three datasets were originally used as test cases for extracting promoter consensi and to test the sequential algorithm [11].

The Smile and PSmile algorithms were tested with two and three boxes, with the first and second group of sequences, respectively, and the results obtained are presented in Table 1. We can conclude by the results that the speed up is almost linear for both tests.

	2 boxes		3 boxes	
	models	time (sec)	models	time (sec)
grid node 1	2	155.83	9987	444.50
grid node 2	1	168.11	6178	385.28
grid node 3	2	245.35	3108	473.70
grid node 4	16	262.51	15884	581.64
total	21	831.80	35157	1885.18
parallel time	262.51		581.64	
sequential time	757.97		1790.70	
speed up	2.9		3.1	

Table 1: Extraction of structured motifs by PSmile and Smile algorithms.

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

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

and computed the quotient

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

between the parallel work and the sequential time results. The results are summarized in Figure 13. We may conclude that the linear coefficient for work-efficiency is between 1 and 1.6.

Other tests were performed to study the exponencial behaviour of the sequential algorithm. We present a result, in Figure 14, where an incremental increase on the factor e of box errors produces an exponencial time result. Note that in this case, a speed up in the order of the available processing units is only achieved when we have $e = 5$.

Error	2	3	4	5
speed up	2.0	2.2	2.1	2.8

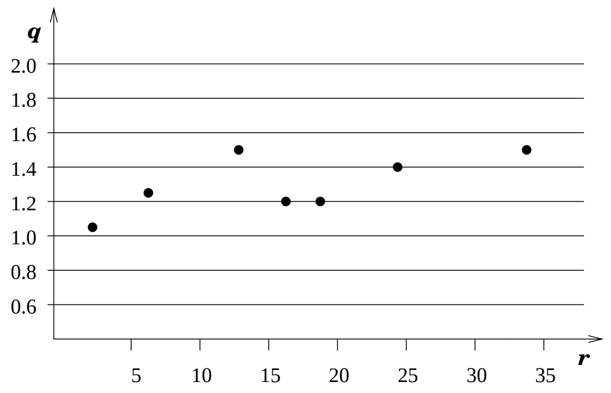

Figure 13: Estimation of the linear coefficient for work-efficiency.

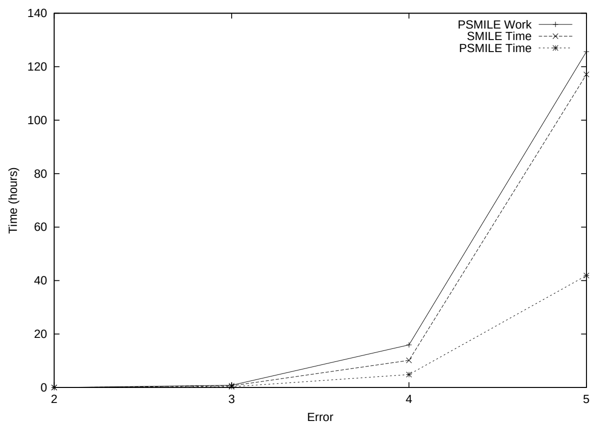

Figure 14: PSmile results with 3 processing units.

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

7 Conclusions

The contributions of this work are threefold. First, we presented a new interesting strongly NP-complete problem, the PARTITION UP TO ε problem. This problem is relevant in the design of efficient parallel searching algorithms where the search space is represented by a tree. For this reason we established an approximation algorithm for the optimization version of the problem. Second, by capitalizing on the previous result, we proposed a parallel algorithm for the efficient extraction of binding-site consensus from genomic sequences. This algorithm is shown to be work efficient, with respect to the sequential algorithm, under easily achievable conditions. Finally, we applied the previous established algorithm for the recognition of specific promoters of the *S. cerevisiae.* The results obtained were cross checked in the laboratory and were assertive for finding the specific promoters.

Future work can progress in several directions. From an algorithmic point of view, and in particular in what concerns parallel algorithms, it would be interesting to partition a suffix tree among the available processing units in a way that optimal space complexity is obtained. With such an algorithm we could obtain work efficiency for the proposed parallelization in an even more general setting. Another obvious relevant problem in this area is to design and develop faster algorithms to extract structured motifs. From a biological point of view, it would be interesting to set up a database of transcription factors and the respective promoters consensus motifs for several organisms. This database would allow users to analyze complex interactions between gene networks and proteins, using semi-automatic methods for processing experimental results.

References

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

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

[3] L. Cardon and G. Stormo. Expectation Maximization algorithm for identifying protein-binding sites with variable length from unaligned dna fragments. *Journal of Molecular Biology*, 223:139–170, 1992.

[4] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms.* MIT Press, 1990.

[5] Y. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit. Identification of common motifs in unaligned dna sequences: application to *escherichia coli lpr* regulon. *Computer Applications in Biosciences*, 11:379–387, 1995.

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

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

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

[9] E. Hunt, M. Atkinson, and R. Irving. A database index to large biological sequences. In *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pages 139–148, 2001.

[10] A. Klingenhoff, K. Frech, K. Quandt, and T. Werner. Functional promoter model can be detected by formal models independent of overall nucleotide sequences similarity. *Bioinformatics*, 1(15):180–186, 1999.

[11] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7:345–360, 2000.

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

[13] M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Latin'98*, volume 1380 of *Lecture Notes in Computer Science*, pages 111–127. Spriger-Verlag, 1998.

[14] K.-B. Schürmann and J. Stoye. Suffix tree construction for large strings. Technical report, Freie Universität Berlin, 2002.

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

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

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

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

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