Suffix Tree Characterization of Maximal Motifs in Biological Sequences[*]

Maria Federico[◇*] Nadia Pisanti[◇]

[◇] Dipartimento di Informatica, Università di Pisa, Italy.

[*] Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Italy.

Abstract

Finding motifs in biological sequences is one of the most intriguing problems for string algorithms designers due to, on the one hand, the numerous applications of this problem in molecular biology and, on the other hand, the challenging aspects of the computational problem. Indeed, when dealing with biological sequences it is necessary to work with approximations (that is, to identify fragments that are not necessarily identical, but just similar, according to a given similarity notion) and this complicates the problem. Existing algorithms run in time linear with respect to the input size. Nevertheless, the output size can be very large due to the approximation (namely exponential in the approximation degree). This often makes the output unreadable, next to slowing down the inference itself. A high degree of redundancy has been detected in the set of motifs that satisfy traditional requirements, even for exact motifs. Moreover, it has been observed many times that only a subset of these motifs, namely the *maximal* motifs, could be enough to provide the information of all of them. In this paper, we aim at removing such redundancy. We extend some notions of maximality already defined for exact motifs to the case of approximate motifs with Hamming distance, and we give a characterization of maximal motifs on the suffix tree. Given that this data structure is used by a whole class of motif extraction tools, we show how these tools can be modified to include the maximality requirement without changing the asymptotical complexity.

1 Introduction

Finding frequent patterns (motifs) in biological sequences has myriads of applications in molecular biology. Following the hypothesis that sequence similarity is often a necessary condition for function correlations, many versions of the problem of finding motifs as particularly frequent patterns in a biological sequence, or as patterns surprisingly shared by several distinct sequences, have been suggested in the literature for as many different biological applications. Depending on the specific task one wants to perform on a biological sequence, the motifs sought can be more or less long, and more or less frequent. However, in all cases, the motif search is *approximate*, that is, distinct occurrences of the same motif are not necessarily identical, but just *similar*, according to a given similarity notion. From the computational complexity point of view, this makes the task of finding over-represented patterns harder, whatever the type of approximation one uses. A typical way to deal with point mutations is to allow up to a certain given degree of *edit distance* between the motif and its occurrences, or between each pair of occurrences of the motif. When one wants to exploit the fact that certain amino acids have similar chemical properties, the best approximation is instead achieved by using a *degenerate alphabet*, which groups these amino acids into sets identified with labels that form, indeed, a new alphabet. A special case of this is to just add to the alphabet a special symbol, named *don't care* symbol, that matches all letters and means simply that at that position of the motif there can be anything. This can be used in representing binding sites, which are DNA fragments that are very well conserved as they are recognized by

[*]This work is supported by the PRIN Project MAINSTREAM funded by the italian Ministry of Education.

agents (such as transcription factors) that act where these sites are located. While these binding sites are well conserved, they are not contiguous sequences and thus detected as motifs with a few don't care symbols inside. For the very same task, a more frequent approximation method is that of using the *Hamming distance*. This distance is defined between patterns of the same length and it simply consists of the number of differences that occur between these patterns. As with the edit distance, one usually sets a maximum allowed distance and then requires that the motifs differ by at most that number of letters substitutions.

As anticipated, finding approximate motifs is a computationally challenging task because, even though there are methods whose time complexity is linear with respect to the input size, the output itself can be very large. Indeed, its size can be exponential with respect to a parameter that measures the approximation (the maximum distance, the degeneracy degree of the degenerated alphabet, the number of don't care symbols used, etc.). Even when the output is not large, partial results during the motifs inference (for example, short motifs that are candidates to form longer motifs by length extension) might lead to memory saturation or to at least a heavy slow down of the computation. This is a big drawback that, next to making the inference task possibly too slow, often leads to a poor usability of the results as they are too large to be investigated by the naked eye. The difficulty in using the results of some motif finding tools is often due to the fact that there are many motifs that satisfy the requirements, while only some of them are significant or, more general, only some of them contain enough information to actually represent all the others. The reason is a redundancy of the output mainly due to nested motifs: as a toy example, if we require motifs to have length at least 2 and $AAAA$ is a motif, then so are AA and AAA as well as, in the approximation case, all approximations of AA, AAA and $AAAA$.

In this paper, we aim at eliminating most of the redundancy that makes the output of a whole class of methods to find approximate motifs unreadable. To this purpose, we extend some notions of maximality already defined for exact motifs to the case of approximate (according to the Hamming distance) motifs, and we give a characterization of maximal motifs on the suffix tree data structure. Moreover, given that the suffix tree is used by many algorithms and tools that infer motifs, we show how the characterization can be used to adapt those algorithms in order to infer on the fly maximal motifs only, and without additional complexity.

2 Preliminary Definitions

We consider strings that are finite sequences of characters drawn from an alphabet Σ. We denote by $s[i]$ the character at position i in a string s and by $|s|$ the length of s. Consecutive characters of s form a *substring* of s. The substring of s that starts from position i and ends at position j is denoted by $s[i..j]$, where $1 \leq i \leq j \leq |s|$. A *prefix* of the string s is a substring that starts at the first position of s, that is $s[1..i]$. A *suffix* of the string s is a pattern that ends at the last position of s, that is $s[i..|s|]$. Given a string x drawn from the same alphabet as s (or from one of its subsets), we say that $s[i..j]$ *exactly occurs* at position i in s if and only if $x = s[i, i+|x|-1]$. In this case, we also say that $s[i, i+|x|-1]$ is an *occurrence* of x in s. We will focus our attention on strings that represent biological sequences, that is, they use the DNA alphabet $\Sigma = \{A,\ C,\ G,\ T\}$. When searching patterns in biological sequences, it is necessary to possibly identify strings that actually show a (limited) number of differences: the Hamming distance between two strings x and y, denoted as $d_H(x, y)$, is the smallest number of letter substitutions that transform x into y (or vice versa as the distance is symmetric). Given an integer $e \geq 0$, we say that a substring y of a string s is an *e-occurrence* of a string x, if and only if $d_H(x, y) \leq e$. In this case we will also talk about an *approximate occurrence*, or simply an *occurrence*, of x in s. When $e = 0$ an approximate occurrence coincides with the exact occurrence defined above as a special case. The list of all occurences of a pattern x in s is denoted by $L_{(e,x)}$ and is called *position set*. We will also denote the position set of x simply with L_x, omitting e. Given a set $L_{(e,x)}$, we denote by $L_{(e,x)} + k$ the set $\{l + k \mid l \in L_{(e,x)}\}$, where k is a given (possibly negative) integer. We will also make use of the *occurrence set*, denoted by $O_{(e,x)}$, that contains pairs of integers (p_i, d_i), where $p_i \in L_{(e,x)}$ and $d_i = d_H(x, s[p_i..|x|-1])$ for each $1 \leq i \leq |O_{(e,m)}|$. Clearly, $|L_{(e,m)}| = |O_{(e,m)}|$ and $d_i \leq e$ for each

$1 \leq i \leq |O_{(e,m)}|$. Intuitively, a motif is a pattern that occurs at least a certain number of times inside a sequence.

Definition 1 *Given a sequence s, a* quorum *$q \geq 2$, and $e \geq 0$, a pattern m is a* motif *iff $|L_{(e,m)}| \geq q$.*

If $e = 0$ we speak about *exact motifs*, because no differences between motifs and their occurrences are allowed; otherwise, when $e > 0$, we call them *approximate motifs*. We will also use the general term *pattern* to indicate either a motif or just a word occurring in s, approximate or not.

Example 1 *Consider the string $s = ACCGAGGACG$. If $q = 3$ and $e = 1$ we have that $m = AC$ has four approximate 1-occurrences in s at positions $1, 2, 5, 8$, among which two are exact (because $m = s[1, 2] = s[8, 9]$). Hence, $L_{(1,m)} = \{1, 2, 5, 8\}$ (whereas $L_{(0,m)} = \{1, 8\}$), and $O_{(1,m)} = \{(1, 0), (2, 1), (5, 1), (8, 0)\}$.*

The traditional motif extraction problem gives as input: (i) the string in which one wants to find the repeated motif (or the set of strings in which one wants to find the common motif); (ii) the quorum q; (iii) the (minimal) length ℓ required for the motif; (iv) optionally, an approximation measure (e.g. the Hamming distance), and the value of e for the approximation measure. The requested output is simply the set of all patterns of length (at least) ℓ that have at least q (possibly approximate) occurrences in s, that is, the complete set of motifs. Within this traditional framework, the output can be very noisy as it contains redundant data. For example, if $\ell = 2$ and $AAAA$ is a motif, then so are AA and AAA, as well as, in the 1-approximation case, all strings obtained from AA, AAA, or $AAAA$ by changing any of the letters with any other letter of the alphabet. With real problem's sizes (long motifs in very long input strings), this redundancy can lead to unfeasibility of motif extraction (due to memory problems) or, in the best case, to an unreadable and thus little useful output. In this paper, we suggest a way to overcome this drawback by introducing a notion of maximality for motifs approximated with Hamming distance, thus identifying a subset of interesting representatives, and an efficient way to detect directly only those. To this purpose, we first introduce the notion of length extension of a motif. With *left extension* (resp. *right extension*) of m, we mean a pattern m' obtained by the concatenation of m with characters at its left (resp. right), and so such that m is a substring, and in particular a prefix (resp. suffix), of m'. If there exists a right or left extension m' of a motif m that is also a motif, then we will say that m is *included* in m'.

Definition 2 *Let m and α be patterns of s. The pattern $m' = m\alpha$ (resp. $m' = \alpha m$) is a* mandatory right *(resp.* mandatory left*) extension of m iff all the occurrences of m in s are followed (resp. preceded) by α. In this case, we call $k = |\alpha|$ the* degree of the extension.

It follows that if m' is a mandatory extension of m then $|L_{(e,m)}| = |L_{(e,m')}|$, and in particular:
- if m' is a mandatory right extension of m then $L_{(e,m)} = L_{(e,m')}$;
- if m' is a mandatory left extension of m with degree d then $L_{(e,m)} = L_{(e,m')} + d$.

Notice that Definition 2 above is the same for both exact and approximate motifs. In the latter case, if m' is a mandatory right/left extension of m, then the two motifs m and m' have the same number of occurrences and also the total number of letter mismatches is the same because the right/left extension of m does not introduce further substitutions between the motif and its occurrences. In other words, we have that $\{d_i \mid (p_i, d_i) \in O_{(e,m)}\} = \{d_i \mid (p_i, d_i) \in O_{(e,m')}\}$. It follows that if m is a motif, then also m' is a motif and vice versa. We will call an extension *mandatory* (without specifying whether it is left or right) if it is an extension on possibly both sides. We can observe that for a motif there exists at most one left mandatory and one right mandatory extension with a certain degree d. It is intuitive to observe that if the occurrences of a motif m are not all followed (resp. preceded) by the same character, then there can not be a mandatory right (resp. left) extension of degree 1 of m, and hence neither a mandatory right (resp. left) extension of higher degree can exist.

2.1 Maximal and Supermaximal Motifs

Notions of motif maximality have been defined in [6] for exact motifs. Further notions of maximality have been defined for approximate motifs: the approximation is achieved using a degenerate alphabet in [17], the edit distance in [9] and don't care symbols in [13, 1]. We extend here some notions of maximality, already introduced for the special case of exact motifs in [6], to motifs approximated with Hamming distance. Other (different) maximality notions for this type of approximate motifs exist in the literature but, contrary to our notions, these are not meant for the general case. In [8] the notion is restricted to the case of tandem repeats. The notion of maximality for motifs approximated with Hamming distance given in [9] does not apply to the whole occurrences set of the motifs, but rather to the special case of *repeats*, that are pairs of occurrences. Hence, to the best of our knowledge, our notions are the first of (super)maximality for motifs approximated with Hamming distance. Intuitively, a motif m is *maximal* if it cannot be further on extended without losing occurrences or introducing more mismatches with occurrences in the input sequence.

Definition 3 *A motif m is* right *(resp.* left*)* maximal *iff it has no mandatory right (resp. left) extension of degree 1. A motif m is* maximal *iff it is both right maximal and left maximal.*

Example 2 *Given the sequence $s = ACTGAGGACT$ with $q = 3$, $e = 1$, and minimal length $\ell = 2$. The motif $m_1 = AC$ with occurrence set $O_{(1,m_1)} = \{(1,0),(5,1),(8,0)\}$ is maximal because it has no mandatory right/left extension of degree 1. Indeed, the right extension of m_1 with character T, that is $m_2 = ACT$, does not occur at position 5; its occurrence set is thus $O_{(1,m_2)} = \{(1,0),(8,0)\}$. On the other hand, the right extension of m_1 with character G, that is $m_3 = ACG$, does not lose occurrences but it introduces new mismatches between the motif and its occurrences at positions 1 and 8. Indeed, its occurrence set is $O_{(1,m_3)} = \{(1,1),(5,1),(8,1)\}$.*
The left extensions of m_1 with characters T and G, resp. $m_4 = TAC$ and $m_5 = GAC$, have occurrence sets resp. $O_{(1,m_4)} = \{(7,1)\}$ and $O_{(1,m_5)} = \{(4,1),(7,0)\}$, hence some occurrences are lost with both m_4 and m_5, and also further differences are introduced with m_4.

This maximality property may not be enough to significantly bound the number of motifs. This is because if a motif is maximal, then there might exist several extensions of it that are also maximal, but having occurrences that are a subset of those (possibly shifted) at which the motif itself occurs. This can result in a drawback that, in some cases, may also not be necessary if, for example, a specific application does not require this kind of *redundancy* of the output.

Example 3 *For example, in the string $s' = ATCGATATATCGAT$ with quorum $q = 2$ and $e = 0$, we have that AT is maximal and it occurs at positions $1,5,7,9$ and 13. Two of its extensions are also maximal with less occurrences: $ATAT$ that occurs at 5 and 7, and $ATCGAT$ occurring at 1 and 9.*

Due to what we just observed, it can be useful to use an even more strict notion of maximality of a motif in terms of its extension. Moreover, in some applications long patterns with few occurrences can be more interesting than short patterns with a lot of occurrences. For these reasons, we formulate the notion of *supermaximality*[1]: intuitively, a motif m is supermaximal if there exists no right or left extension m' of m that is a motif.

Definition 4 *A motif m is* supermaximal *iff it is not a substring of another motif.*

In other words, a motif is supermaximal if the quorum property does not hold anymore when one tries to extend it in either way.

Example 4 *Consider the string $s'' = A^n = AAA\ldots AAA$ and assume $q = 2$, $e = 0$ and $\ell = 1$. The set of exact maximal motifs has $n-1$ elements, namely all the motifs A^i with $1 \le i \le n-1$*

[1]Since our definition here is the natural extension to the case of approximate motifs of that given in [6], we keep the same name.

and position set $\{1, 2, \ldots, (n-i)+1\}$*. That is, all the motifs of* s'' *are maximal. On the contrary, there is a unique exact supermaximal motif, namely* A^{n-1} *occurring at positions* 1 *and* 2*, because it is the one not included in any other motif, and any right or left extension breaks the quorum constraint.*

As we can see also from the previous example, a supermaximal motif does not include as a substring any other supermaximal motif. In the approximate motifs case, this means that even though we add mismatches, we have to extend as much as possible (provided the quorum constraint is satisfied) a motif to obtain a supermaximal motif. Observe that the supermaximal motifs are a subset of the maximal motifs.

3 Characterization of Motif (Super)maximality on Suffix Tree

In this section we will give a characterization of maximal and supermaximal motifs of a string s on the suffix tree of the string itself. We will do this both for exact and for approximate motifs. For exact motifs, the characterization has already been introduced in [6], but we start Section 3.1 with a brief description of it, as this will introduce some notions and terminology that will help understand the characterizations for approximate motifs as well.

The suffix tree is a tree data structure that indexes a text and whose edges are labelled by nonempty words. In particular, the suffix tree T of a sequence s has the following properties:

- there is a root-leaf path per each suffix of s, and vice versa;
- all internal nodes (except possibly the root) have at least two children;
- the input sequence is padded with a terminal symbol (not present in the sequence itself), ensuring that no suffix is a prefix of another.

As a consequence, each root-node path for a node u corresponds to a substring of the text, to which we will refer to as *the word spelled by* u, or *path-label of* u. In addition, given that for some substrings the path does not end at a node but rather inside an edge, we will also talk about *the word spelled by a path*. The positions of the occurrences of a word spelled by a node u, or by a path ending at u, in the input sequence are represented by the (starting positions of the suffixes that label the) leaves in the subtree rooted at u, whose number is denoted by $Lv[u]$. If u is a leaf, then we set $Lv[u] = 1$. It is possible to annotate all internal nodes of a suffix tree with the value of $Lv[u]$ in linear time, by a simple traversal of the suffix tree (as shown in [16]).

If $|s| = n$, then the space complexity of the suffix tree is in $O(n)$, and so is the time complexity for its construction [22, 12, 18].

There is a variant of the suffix tree data structure, called *generalized suffix tree*, which indexes a *set of* $N > 1$ input sequences. There is a root-leaf path per each suffix of each one of the N sequences. At the end of each input sequence a distinct terminal symbol is appended, allowing one to distinguish root-leaf paths that label suffixes of different sequences. If all the input sequences have size n, then the space complexity of the generalized suffix tree is in $O(nN)$, as well as the time complexity for its construction: again, linear with respect to the input size.

3.1 Exact Motifs

In [6], Gusfield already showed a characterization of maximal and supermaximal repeats on suffix tree. The generalization of Gusfield's results to the case of maximal and supermaximal exact motifs (rather than repeats, that is going from $q = 2$ to any $q \geq 2$) is trivial. It suffices to perform simple subtree size counting, which is actually already suggested in [6], in order to count whether there are enough occurrences, rather than just looking whether a branching exists (which is enough to guarantee 2 occurrences). Nevertheless, both for maximality and supermaximality, we now briefly describe such characterizations (for a detailed explanation see [6]) because this

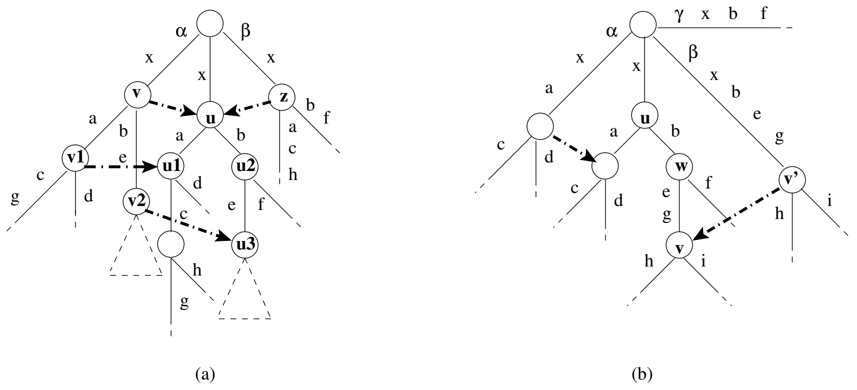

Figure 1: (a) An internal node u of T reached by two suffix links. (b) An internal node u that is not reached by any suffix link.

will introduce some useful terminology that we will need also later for approximate motifs (this is especially the case for supermaximality, where the extension is a bit less trivial). A pattern is a motif if it labels a path on the suffix tree that ends at an internal node v_m such that $Lv[v_m] \geq q$, or inside an edge that ends at an internal node for which this is the case. We recall the following two definitions introduced by Gusfield in [6]. The *left character of a leaf* of a suffix tree T for the sequence s is the character preceding the suffix of s at the position (where the suffix starts) represented by that leaf. A node v is *left diverse* if at least two leaves in the subtree rooted at v have different left characters, so if m labels a left diverse node it means that there are at least two occurrences of m in s preceded by different characters. By definition, a leaf cannot be left diverse.

Note that a motif is right maximal if it labels a path on T that ends at an internal node, and it is left maximal if such a node is left diverse. Summing up, an exact motif is maximal if and only if it labels an internal node v of T such that $Lv[v] \geq q$ and v is left diverse, because in this case right and left mandatory extensions of degree 1 cannot exist for m.

Gusfield also provides a characterization on suffix tree of supermaximal repeats. In particular he proves that supermaximal repeats label internal nodes v of T such that all the children of v are leaves and each has a distinct left character. In such a case, indeed, none of the extensions of a repeat has at least two occurrences. This idea can be generalized to the case of supermaximal exact motifs in a straightforward manner. For this purpose, we introduce the notion of *right* and *left q-limited node*.

Definition 5 *Let T be the suffix tree for the sequence s. A node v of T that spells a pattern m is* right q-limited *(resp.* left q-limited*) iff m has no right (resp. left) extension of degree 1 that occurs at least q times in s. We say that a node is* q-limited *if it is right q-limited and left q-limited.*

Note that *not* being right q-limited (or left q-limited) is a property that propagates upward: if an internal node v is not right (resp. left) q-limited, then neither is any of its ancestors in the tree. If $Lv[v] < q$, then v is q-limited because the word spelled by v does not occur at least q times in s, and none of its extensions can occur at least q times in s.

A right extension of degree 1 of m can label a child of v or a path ending inside an edge with a child of v as destination. It follows that v is right q-limited if and only if its children are all nodes v' such that $Lv[v'] < q$.

The characterization on the suffix tree of the left q-limited property is a bit less immediate and it involves the so-called suffix link [6], which is a pointer that connects a node v spelling ax (with $a \in \Sigma$ and $x \in \Sigma^*$) to the node u that spells x. In other words, this pointer provides a link from node v to node u such that v spells a left extension of degree 1 of the path-label of u. Note that the left extensions of x do not always label nodes, as they can also label paths ending inside edges. Node u is reached by as many suffix links as the number of left extensions of degree 1 of x that label a node (these are at most $|\Sigma|$), as is shown in Figure 1(a). If there exists a left extension that labels a path ending inside an edge leading to node v', then there exists a suffix link from v' to a descendant of u labelled by a pattern whose left extension of degree 1 is the path-label of v'. An example of this can be seen in Figure 1(b), where there is a suffix link from v' to v, which is a descendant of u. It follows that if all the left extensions of degree 1 of the pattern spelled by a node label paths ending inside edges, then this node is not reached by any suffix link, as is the case for u in Figure 1(b). In particular, we can observe that an internal node u, labelled by x, is reached by a suffix link only if at least two of its children are such that both their path-labels are preceded by the same character α at some (possibly all) of their occurrence positions in the input sequence. In this case, indeed, there exists left extension αx that labels an internal node from which a suffix link directed to u originates. Moreover, note that there exists only one suffix link directed to the leaf node representing the suffix at position i in the input sequence s; such a link originates from the leaf representing the suffix at position $i-1$ in s.

Due to what we just showed about suffix links and to the fact that not being left q-limited is a property that propagates upward in the tree, we observe that a node v is left q-limited only if $Lv[u] < q$ holds for each node u from which a suffix link to v originates, and its children are left q-limited nodes.

Theorem 1 *Given a quorum q, a sequence s and its suffix tree T, the pattern m labelling the path to a node v of T is a supermaximal exact motif iff $Lv[v] \geq q$ and v is q-limited.*

Proof If v is q-limited and $Lv[v] \geq q$, then clearly m satisfies the quorum and it has no right and left extensions of degree 1 that occur at least q times in s, and hence it is supermaximal. Conversely, if m is a supermaximal exact motif then it has at least q occurrences and thus it must be that $Lv[v] \geq q$; moreover, m is not included in any other motif, that is all its right and left extensions of degree 1 have less than q occurrences, which is equal to saying that v is q-limited. □

It follows that an exact motif is supermaximal if and only if it labels a path on T ending at a q-limited node.

3.2 Approximate Motifs

We now extend Gusfield's characterization of (super)maximality on suffix tree to the case of motifs approximated with Hamming distance, according to the definition of (super)maximality that we have given in Section 2.1. As we have noted in Section 3.1, an exact motif labels a single path on suffix tree. This is not true for approximate motifs. Let $x_1, \ldots, x_h$ be distinct occurrences of an approximate motif m in a sequence s, that is x_i is a substring of s such that $d_H(m, x_i) \leq e$ and $x_1 \neq \cdots \neq x_h$. Each occurrence x_i labels a distinct path from the root that ends at a node or inside an edge: these paths are called *occurrence-paths* of m.

3.2.1 Maximal Approximate Motifs

The characterization on suffix tree provided in Section 3.1 for maximal exact motifs can be simply extended to approximate motifs.

Theorem 2 *Let T be the suffix tree for the sequence s. An approximate motif m is right maximal iff:*

1. *at least one occurrence-path of m labels a node, or*
2. *all the occurrence-paths of m end inside edges and at least two of them have different characters at depth $|m| + 1$.*

Proof In both cases, there exist at least two occurrences of m that are followed by different characters in s. In particular, the first case is the same as that showed for maximal exact motifs with the additional possibility that the occurrence path ends at a leaf (in which case the motif is clearly right maximal because one of its occurrences is a suffix of s and then it cannot be extended to the right). In the second case, let x and y be the labels of two occurrence-paths of m ending inside edges and being followed by different characters, α and β, at string-depth $|m|+1$. The substrings x and y of s, which are distinct occurrences of m, are then resp. followed by α and β, and therefore there exists no mandatory right extension of degree 1 for m, which is thus right maximal. If neither of the two cases holds, then all occurrence-paths of m end inside edges and have the same character at string-depth $|m|+1$. In this situation, all its occurrences are followed by the same character, and hence m has a mandatory right extension of degree 1 and it is not right maximal. □

Reminding that if a node of T is not left diverse then all the leaves in its subtree have the same left character and that, by definition, leaves are not left diverse, we give a characterization also for the left maximality of approximate motifs.

Theorem 3 *Let T be the suffix tree for the sequence s. An approximate motif m is left maximal iff:*

1. *at least one occurrence-path of m labels a left diverse node or ends inside an edge ending at a left diverse node, or*
2. *there are at least two distinct occurrence-paths of m ending (inside edges ending) at nodes that are not left diverse, and such that the leaves reached following these paths do not have the same left character.*

Proof In the first case, from the definition of left diverse node follows that there exist at least two occurrences of m in s that are preceded by different characters, and hence m is left maximal. In the second case, among the two distinct occurrence-paths, let one end at a node x (or resp. within an edge ending at node x), and the other one at a node y (resp. within an edge ending at y). By hypothesis, neither x nor y is left diverse. Note that only one of the two paths can end at a leaf, because there cannot be two leaves at the same depth in the tree. If the subset of the occurrence positions of m in s represented by the leaf x, or by the leaves in the subtree of x, are preceded by a character that is distinct from the character that precedes either the occurrence position represented by y, if it is a leaf, or the subset of occurrence positions in the subtree of y, then there exist at least two distinct occurrences of m that are preceded by different characters in s, and hence m is left maximal. Conversely, if neither of the two cases holds, then all the occurrence-paths label nodes that are not left diverse, or end inside edges that do not have a left diverse destination node, and all the leaves reached following these paths have the same left character. If this is the case, then all the occurrences of m are preceded by the same character, and hence m is not left maximal. □

3.2.2 Supermaximal Approximate Motifs

In this subsection we show the characterization on suffix tree of supermaximal approximate motifs. In the case of exact motifs we used the notion of right and left q-limited node to verify supermaximality of a motif. When we consider approximate motifs, this notion does not suffice because there could be multiple occurrence-paths for the same motif. Indeed, even if all the distinct occurrences of a motif m maintain less than q positions when extended, it can still be the case that the sum of the number of positions at which the extensions occur is greater than q.

Theorem 4 *An approximate motif m is supermaximal iff, for every $\alpha \in \Sigma$, $m' = m\alpha$ (resp. $m' = \alpha m$) has occurrence-paths ending at nodes, or inside edges with destination nodes, $u_1{}', \ldots, u_h{}'$ such that $\sum_{i=1}^{h} Lv[u_i{}'] < q$.*

Proof The proof is straightforward. Indeed, if all the extensions of degree 1 of m have occurrences labelling nodes, or edges ending at nodes, $u_1{}', \ldots, u_h{}'$ such that $\sum_{i=1}^{h} Lv[u_i{}'] < q$, then none of them is a motif (because they do not satisfy the quorum constraint) and thus m is supermaximal.

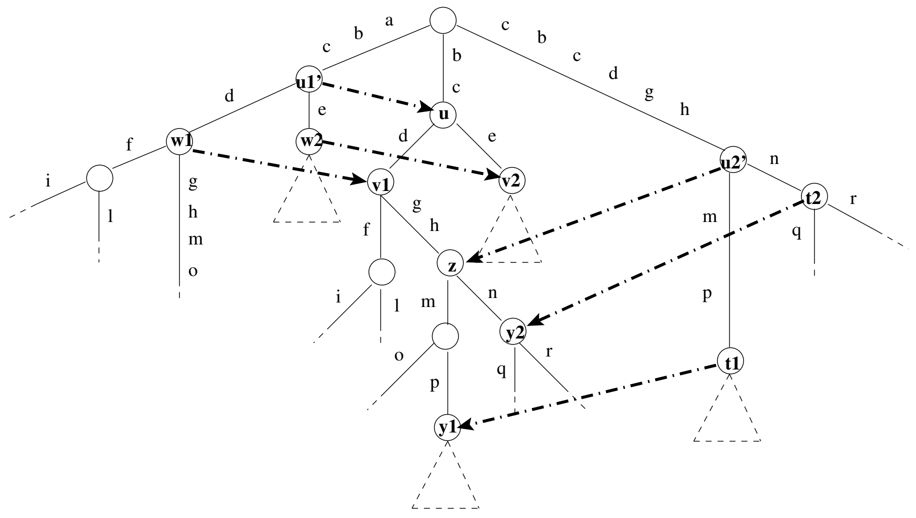

Figure 2: Suffix links and left extensions of path-labels

On the contrary, if there exists an extension of m such that $\sum_{i=1}^{h} Lv[u_i{}'] >= q$, then this is a motif and thus m is not supermaximal. $\square$

The suffix tree is very suitable to count occurrences that are preserved when extending a motif m to the right or to the left, and hence to understand whether m has right and left extensions of degree 1 that are motifs. In detail, let x be an occurrence of m such that $d_H(m,x) = d$, and let $m' = m\alpha$ with $\alpha \in \Sigma$ be a right extension of m.

If x labels a path ending inside an edge (u,v), or at an internal node u where v is a child of u, then $m' = m\alpha$ has an occurrence-path ending inside (u,v) or at v only if the character at string-depth $|m|+1$ along (u,v) in T is α, or if it is $\beta \neq \alpha$ and $d < e$. Whereas, if x labels a path ending at a leaf node, then the right extension of m with α causes in every case the loss of the occurrence spelled by that leaf. Concerning left extensions, we have to follow suffix links directed to u and to its descendants. Actually, not all these suffix links are interesting, but only those whose source node is not a child of a node from which a suffix link directed to u, or to a descendant closest to u, originates. As an example, in Figure 2 the interesting suffix links for node u are those from $u1'$ to u and from $u2'$ to z, which is a descendant of u. Denoting by $sln[u]$ the set of nodes from which interesting suffix links originate, $m' = \alpha m$ has an occurrence-path ending at string-depth $|m|+1$ along the edge ending at some node $v \in sln[u]$, only if the label of v starts with α, or if it starts with $\beta \neq \alpha$ and $d < e$. Using these observations for all distinct occurrences x of an approximate motif m, we know how to count the number of occurrences of a motif while extending it to the right and to the left and, in particular, to check whether it keeps on satisfying the quorum constraint.

Finally, even though the notion of q-limited node does not suffice for the case of approximate motifs, as we explained at the beginning of this subsection, it is nevertheless important to observe that:

Observation 1 *If there exists an occurrence-path of an approximate motif m ending at an internal node that is not q-limited, or inside an edge with a destination node that is not q-limited (that is, an occurrence of m is followed or preceded by the same character in at least q positions in s), then there must exist a right or left extension of m that occurs more than q times in s, and thus m is not supermaximal.*

4 Inferring Maximal and Supermaximal Motifs

4.1 Motif Extraction with Suffix Tree

The first (exact) algorithm for the extraction of motifs with mismatches was introduced in [16]. Motifs are considered in lexicographical order starting from the empty word, and they are extended on the right as long as the quorum is satisfied until either a valid motif of maximal length is found (if the required length is reached), or the quorum is no longer satisfied. At each moment, all paths spelling approximate occurrences of the current motif are taken into account. The number of the motif's occurrences is then computed as the sum of $Lv[v]$ for all nodes or destination nodes v of edges at which such occurrence-paths end. The algorithm exploits the property that these occurrence-paths on the suffix tree are such that those of the motif $m\alpha$ (where $m \in \Sigma^*$ and $\alpha \in \Sigma$) are found along the occurrence-paths of m and checking whether there is a character α following or whether a new mismatch can be introduced.

Assuming that the required length of the motif is ℓ, and that at most e mismatches are allowed, the algorithm has worst-case time complexity in $O(t_\ell \nu(e,\ell))$, where t_ℓ is the number of tree nodes at depth ℓ, and $\nu(e,\ell)$ is the number of words of length ℓ that differ in at most e letters from a word m of length ℓ. This value does not depend on m, and it holds that $\nu(e,\ell) \leq \ell^e|\Sigma|^e$. This upper bound is in practice not at all tight. Nevertheless, no better bound can be given and therefore the time complexity is linear in the input size, but possibly exponential in the number e of mismatches. Finally, the space complexity is in $O(t_\ell)$.

In the next two subsections, we will show how to modify the algorithm of [16] in order to output only maximal or supermaximal motifs.

4.2 Checking Maximality

In this section we provide a brief description of the operations needed to extend the algorithm above in order to extract only maximal motifs, and we show that the additional complexity needed to satisfy the motif maximality requirement is constant. In the following, an occurrence-path is represented by a triple (x, x_{err}, pos), where x is the node (or the destination node of the edge) at which such a path ends, x_{err} is the number of mismatches between the pattern m and the path-label, and pos is equal to -1 if the path ends at a node, while it is equal or greater than 0 if the path ends inside an edge; in the latter case it indicates the depth of the path along that edge.

Let us again consider maximal exact motif extraction as a starting point. In [6], Gusfield presents a linear time algorithm to find left diverse nodes of a suffix tree T. A bottom-up traversal of T is performed, and, for each node v, the algorithm stores either that v is left diverse, or the common left character of every leaf in the subtree rooted at v. Hence, assuming that we have a suffix tree whose nodes are annotated with this information, the additional cost to select only maximal motifs among all exact motifs is constant. Indeed, for every motif found by a motif discovery algorithm, it is enough to verify whether it labels a left diverse node of T. Only if this is the case, the maximal exact motif must be output.

If we search for maximal approximate motifs, the extraction algorithm can simply be extended in order to test the conditions of Theorems 2 and 3. To this aim, it suffices to maintain the following information:

- two booleans *maxDx* and *maxSx* initialized to *false*, that indicate whether a motif is right and left maximal, resp.;
- two variables *charDx* and *charSx*, that store the right and left character, resp., of the motif occurrences.

For every occurrence-path (x, x_{err}, pos) of a pattern $m' = m\alpha$ identified on the suffix tree T by the extraction algorithm, it holds that

- If $pos = -1$ (that is the path ends at node x), then $maxDx = true$ (because condition 1 of Theorem 2 is verified). If x is left diverse, then $maxSx = true$ (because condition 1 of

Theorem 3 is verified). In this case m' is maximal. Otherwise, if x is not left diverse, then it is necessary to compare the left character c_s, which is associated to x, with the variable $charSx$ (if this is not the first examined occurrence-path, and otherwise $charSx$ simply takes the value c_s). If the two characters are different, then $maxSx = true$ (because condition 2 of Theorem 3 is verified).

- If $pos \geq 0$ (that is the path ends inside an edge), then the character c_d at depth $pos + 1$ along the edge is compared with the variable $charDx$ (if this is not the first examined occurrence-path, and otherwise $charDx$ simply takes the value c_d). If the two characters are different, then $maxDx = true$ (because condition 2 of Theorem 2 is verified). At this point, the left maximality is tested as explained in the previous case.

These operations do not necessarily have to be performed for all occurrence-paths of m', but only if $maxDx$ or $maxSx$ is *false*. When $maxSx$ and $maxDx$ become both $true$, there is no need to check anything, because m' is surely maximal. When all the occurrence-paths of m' are found, if the number of occurrences is equal to or greater than the quorum and $maxDx$ and $maxSx$ are both $true$, then m' is maximal. Therefore the motif must be output only if this further condition is verified. It follows that, in all cases, the condition to check consists of two character comparisons, and then the additional cost to extract only maximal approximate motifs is constant.

4.3 Checking Supermaximality

In this subsection we provide a description of the operations needed to extend existing motif discovery algorithms in order to extract only supermaximal motifs, and we show that the additional complexity cost due to the motif supermaximality check is negligible.

Once again, let us first consider the extraction of supermaximal exact motifs. All nodes v of T can be annotated with the right and left q-limited property by a simple bottom-up traversal of T. If $Lv[v] < q$, then v is q-limited. Otherwise, in order to process a node v, the algorithm examines its children: if a child of v is not q-limited, then v is not q-limited. If all children of v are right q-limited, then we must check $Lv[v']$ for every child v' of v. If $Lv[v'] < q$ for every v', then v is right q-limited. If all children of v are left q-limited, then we must check the suffix links directed to v. If no suffix link exists, or if $Lv[v''] < q$ holds for all nodes v'' from which a suffix link to v originates, then v is left q-limited. Assuming that we have a suffix tree whose nodes are annotated with this information, like for maximal motifs, the additional cost to extract only supermaximal exact motifs is constant. Indeed, for every motif found by any motif discovery algorithm, it suffices to check whether it labels a q-limited node of T.

The extension of existing motif discovery algorithms in order to extract only supermaximal approximate motifs is a little more complex. It requires that for every node v of T, in addition to right and left q-limited information, we also have the set $sln[v]$ of nodes from which suffix links directed to v or to its descendants originate. This set can be represented by an array of $|\Sigma|$ positions. At position i of $sln[v]$ there is the node from which a suffix link to v (or to one of its descendant) originates, and whose label starts with the ith character in Σ. The set $sln[v]$ for all nodes of T can be computed with a bottom-up traversal of T. Let us assume to have at the start, for every node v of T, the array $sln[v]$ storing only nodes from which suffix links directed to v originate (if any): this array can be built within the linear time complexity of suffix tree construction and it gives the link between the label of v and its left extensions of degree 1 that end at nodes of T. In order to obtain also a link with the extensions that end inside edges (if any), it is enough to iterate on the $|\Sigma|$ positions of $sln[v]$, and, for each empty position i, check whether a node is stored at the same position in the array of a child of v (note that at most one such child can exist). If this array exists, then the node found is stored also at position i of $sln[v]$. Therefore, being $|\Sigma|$ the size of $sln[v]$ and being the children of v at most $|\Sigma|$, then in the worst case the time to update the array for every node of T is proportional to $|\Sigma|^2$. Assuming that we have a suffix tree whose nodes are annotated with this information, we now show how the existing extraction algorithms can be extended to output only supermaximal approximate motifs. First of all, in order to check the supermaximality of a motif, it is necessary to maintain the following information:

- a variable *numOccEst* that indicates the number of occurrences of any right and left extension of degree 1 of a pattern m, resulting from the extension of the occurrences of m that have a number of differences with m strictly less than threshold e;
- a variable *numOccEstDx* that indicates the number of occurrences of any right extension of degree 1 of m;
- a variable *numOccEstSx* that indicates the number of occurrences of any left extension of degree 1 of m;
- a set *OccEstSet* that stores the occurrence-paths of m with a label that is at Hamming distance equal to e from m;
- a boolean *notSupermax* (initialized to *false*) that is *true* if the motif found is not supermaximal.

For every occurrence-path (x, x_{err}, pos) of a pattern $m' = m\alpha$ found by the motif extraction algorithm during the right extension of a motif m with a character α, in addition to the other operations performed by the motif extraction process, we must do the following:

- if $x_{err} < e$, then *numOccEst* is updated: $numOccEst := numOccEst + Lv[u]$;
- if $x_{err} = e$, then the occurrence-path (x, x_{err}, pos) of m' is added to the set *OccEstSet*.

These operations need not necessarily be performed for every occurrence-path, but only until one of the following two conditions is verified:

- an occurrence-path of m' ending at a node being not right or left q-limited or inside an edge with destination node of this type is found (if any), because in this case m' is not supermaximal (see Observation 1); or
- $numOccEst \geq q$, because if *numOccEst*, which is the number of occurrences of m' that are at Hamming distance strictly less than e from m', exceeds the quorum q, then all right and left extensions of m preserve such occurrences and so they are motifs as well (and hence m' is again not supermaximal).

These checks introduce only a constant additional time cost to the extraction algorithms. After finding all the occurrence-paths of m, if none of the described cases is verified and m' is a motif, then it is necessary to check if m' is supermaximal, that is whether or not all its right and left extensions have less than q occurrences. This is done by counting how many occurrences of m' are preserved by its right and left extensions with any character in Σ, as showed in Section 3.2.2. If there exists a right or left extension of degree 1 of m that occurs more than q times, then m is not supermaximal. This can be verified by checking how many occurrences at Hamming distance equal to e from m' are preserved by the extensions of m'. To do this, it is necessary to examine all occurrence-paths in *OccEstSet*, for every character α in Σ, and to count the number of occurrences preserved by the right and left entensions of m' with α, until an extension of m' that occurs at least q times is found. If such an extension exists, then m is not supermaximal.

In detail, this counting can be performed in the following way. For every character α in Σ:

1. *numOccEstDx* and *numOccEstSx* are both initialized to *numOccEst*.
2. For every occurrence-path (x, x_{err}, pos) of m' in *OccEstSet* (until $notSupermax = true$):

   (a) • if $pos \geq 0$, then x is the destination node of the edge inside which the path ends. If the character at deph $pos + 1$ along that edge is α, then $numOccEstDx := numOccEstDx + Lv[x]$;

- if $pos = -1$, then x is the node at which the path ends. Let $u_1', \ldots, u_t'$, with $t \leq |\Sigma|$, be the children of x. If there exists an edge (x, u_i') with a label starting with α, then $numOccEstDx := numOccEstDx + Lv[u_i']$.
  Thanks to the properties of suffix trees, performing this check only requires a comparison between two characters. Indeed, from every node of a suffix tree, at most $|\Sigma|$ lexicographically sorted edges originate. Let α be the kth character in Σ (assuming it lexicographically sorted too) and let u_j' be the child of x such that the first character of the label of edge (x, u_j') is equal to the ith character in Σ, with $i < k$, that is nearest to α. We have that it is not necessary to compare α with the first character of every edge (x, u_i'), but it suffices to compare it only with the first character of the label of the edge (x, u_{j+1}').

  If $numOccEstDx \geq q$, then $notSupermax = true$, because m' is not right supermaximal. Otherwise:

  (b) Let $u_1', \ldots, u_t'$, with $t \leq |\Sigma|$, be nodes from which suffix links directed to x, or to its descendant, originate, that is the elements of the array $SuffixLinkNodes_x$. If there exists a node u_i' with a label starting with α, then $numOccEstSx := numOccEstSx + Lv[u_i']$. Let k be the position of α in Σ; the node u_i', if it exists, is the one stored at position k in $SuffixLinkNodes_x$, therefore the check has a constant cost.

  If $numOccEstSx \geq q$, then $notSupermax = true$, because m' is not left supermaximal. Otherwise, the next occurrence-path in $OccEstSet$ must be examined and step 2 must be iterated.

3. If all occurrence-paths in $OccEstSet$ are examined and $numOccEstDx$ and $numOccEstSx$ are both less than q (so $notSupermax$ is still *false*), then repeat steps 1-2-3 to examine the extensions of m' with the next character in Σ.

The operations needed to count the occurrences preserved by every right and left extension of an approximate motif m with a character α have a cost proportional to the number of occurrence-paths of m whose label is at Hamming distance e from m (examining a single occurrence-path in $OccEstSet$ has a constant cost because it simply consists of a comparison between two characters). If ℓ is the motif length, these occurrence-paths are at most $p = \binom{\ell}{e} \cdot (|\Sigma| - 1)^e$ and hence the additional cost to verify if a motif is supermaximal is proportional to $O(p|\Sigma|)$.

Summing up, the additional cost to output (super)maximal motifs only is constant in all cases except for the one just described (supermaximality for approximate motifs), where, however, this cost is in practice negligible. Moreover, notice that if a motif is supermaximal, then the extraction algorithm can avoid to further extend it because none of its right extensions can be a motif. Thus, the overhead introduced by the supermaximality check is balanced by a reduction of the number of intermediate length motifs that have to be extended during the extraction process.

4.4 Algorithms for Motif Discovery with Suffix Tree

The algorithm in [16], that we have described in Section 4.1, is actually the ancestor of a whole class of algorithms that use the suffix tree for the extraction of motifs from a sequence or a set of sequences ([3, 2, 4, 15, 10, 11, 21, 20, 14]).

So far, we have focussed on the problem of motif discovery from a single sequence of input. A variant of the problem is that of having N input sequences, and one has to find motifs that are common to (at least a certain percentage of) the input sequences. That is, motifs *common* to a set of sequences, rather than *repeated* within a unique sequence, and the notion of quorum is different. This version of the problem is very relevant in many biological applications, including that of finding transcription factors binding sites. The algorithm for the extraction of motifs common to a set of N sequences is very similar to the algorithm described above. First of all, a generalized suffix tree for the N input sequences $s_1, s_2, \ldots, s_N$ is built. Secondly, now a motif is a pattern that e-occurs in at least q sequences of the input set. Hence, in order to check the new notion of quorum, the information we need to store for each node of the generalized suffix tree is the number of distinct input sequences in which the path-label of the node occurs. That is, we need to store the number of different sequences to which the leaves in the subtrees rooted at these nodes refer. Counting this number is called the "Color Size Problem" by Hui in [7]. The size of the set of colours of a node x is the number of different *colours* associated to the leaves in the subtree rooted at x, where the colour i is associated to a leaf if it is labelled by a suffix of the input sequence s_i. In [7], this number is called CSS_x. However, knowing CSS_x for all nodes only suffices if differences between motif and its occurrences are not admitted. Otherwise, for approximate motifs it is necessary to be able to say how many colours are common to two or more nodes in the tree. To do this, to every node x we associate an array of size N, which is denoted by $Colours_x$ and defined as:

$$Colours_x[i] = \begin{cases} 1 & \text{if at least a leaf in the subtree rooted at } x \text{ is labelled by a suffix of } s_i \\ 0 & \text{otherwise} \end{cases}$$

The array $Colours_x$ may be implemented by a bit vector and can be obtained by a simple traversal of the tree. Finally, a pattern m is output as a motif if and only if in the union of the bit vectors of the nodes (or the destination nodes of the edges) at which occurrence-paths of m end, there are at least q bits equal to 1.

The algorithm that finds common motifs has worst-case time complexity in $O(t_\ell N^2 \nu(e,\ell))$ and space complexity in $O(t_\ell N)$. With respect to the algorithm for repeated motif discovery, there is a complexity increase due to the time and space needed to create and manage the data structure $Colours_x$, that has size N for each node of the tree.

The above algorithm was extended in [10, 11] to the case of *structured motifs*, which are motifs composed of two or more parts lying at a certain given distance. The resulting tool, named SMILE, was applied to promoter signal detection in [20, 21]. Moreover, using a data structure that is an enriched version of the suffix tree, basically the same framework has been used in [2, 4, 3]. Finally, the tool presented in [14], which resulted in [5] to have very good performances, uses an algorithm that is basically an heuristic version of [16], and uses the suffix tree as well.

Our results can be used by all the algorithms and tools mentioned in this section to directly output only (super)maximal motifs, with the obvious improvement in readability and significance.

We expect that the approach we suggest could sensibly reduce the number of output motifs without decreasing their significance and with a constant or negligible extra time complexity. However, we realize that in a worst-case scenario the improvement (that is, the number of discarded motifs) could be none: one can always design a string in which all motifs are maximal. Nevertheless, our experience with the analysis of biological sequences has taught us that for such applications, the redundancy in the output is very well present. Indeed, biological sequences contain many more repetitions than randomly generated sequences, which, in their turn, averagely would be far from containing only maximal motifs.

5 Conclusions

In order to remove the redundancy in the output of existing algorithms for finding motifs, we have extended notions of (super)maximality already defined for exact motifs to the case of approximate (according to the Hamming distance) motifs. For all of them, we have given a characterization on the suffix tree data structure. This has allowed us to show how to adapt a whole class of algorithms based on suffix tree, and for which available tools exist, to infer (super)maximal motifs only. We have proved that the additional computational cost due to the on the fly check of (super)maximality requirements is only constant in all cases except that of supermaximality for approximate motifs, in which case it is however negligible. Therefore, our results suggest a way to improve motif extraction tools providing outputs that are more readable and more usable by biologists, without any additional complexity in the extraction phase.

Acknowledgment

We thank Esko Ukkonen for sending us a copy of [19].

References

[1] A. Apostolico and L. Parida. Incremental paradigms of motif discovery. *Journal of Computational Biology*, 11:15–25, 2004.

[2] A.M. Carvalho, A.T. Freitas, A.L. Oliveira, and M.-F. Sagot. Efficient extraction of structured motifs using box-links. In *Proceedings of the 11th Symposium on String Processing and Information REtrieval (SPIRE)*, volume 3246, pages 267–268, 2004.

[3] 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 *Proceedings of the 3rd Asia-Pacific Bioinformatics Conference (APBC)*, pages 273–282. Imperial College Press, 2005.

[4] A.M. Carvalho, A.T. Freitas, A.L. Oliveira, and M.-F. Sagot. An efficient algorithm for the identification of structured motifs in dna promoter sequences. *IEEE/ACM Transaction on Computational Biology and Bioinformatics*, 3(2):126–140, 2006.

[5] M. Tompa et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–144, 2005.

[6] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

[7] L.C.K. Hui. Color set size problem with application to string matching. In *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 230–243, 1992.

[8] R.M. Kolpakov and G. Kucherov. Finding approximate repetitions under hamming distance. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, pages 170–181, 2001.

[9] S. Kurtz, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich. Computation and visualization of degenerate repeats in complete genomes. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 228–238, 2000.

[10] L. Marsan and M.-F. Sagot. Extracting structured motifs using a suffix tree. Algorithms and application to promoter consensus identification. In *Annual Conference on REsearch in COmputational Molecular Biology (RECOMB)*, 2000.

[11] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory consensus identification. *Journal of Computational Biology*, 7:345–360, 2001.

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

[13] L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao. Pattern Discovery on Character Sets and Real-valued Data: Linear Bound on Irredundant Motifs and Efficient Polynomial Time Algorithm. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2000.

[14] G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole. Weeder web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Research*, 32(Web-Server-Issue):199–203, 2004.

[15] N. Pisanti, A.M. Carvalho, L. Marsan, and M.-F. Sagot. Risotto: Fast extraction of motifs with mismatches. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 757–768, 2006.

[16] M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics (LATIN)*, pages 374–390, 1998.

[17] H. Soldano, A. Viari, and M. Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognition Letters*, 16:243–246, 1995.

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

[19] E. Ukkonen. Structural analysis of gapped motifs of a string. In *Proceedings of the Mathematical Foundations of Computer Science (MFCS)*, pages 681–690, 2007.

[20] A. Vanet, L. Marsan, A. Labigne, and M.-F. Sagot. Inferring regulatory elements from a whole genome. an application to the analysis of the genome of helicobacter pylori sigma 80 family of promoter signals. *Journal of Molecular Biology*, 297:335–353, 2000.

[21] A. Vanet, L. Marsan, and M.-F. Sagot. Promoter sequences and algorithmical methods for identifying them. *Research in Microbiology*, 150(1):1–21, 1999.

[22] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory (former name of Foundations of Computer Science (FOCS))*, pages 1–11, 1973.