Spelling approximate repeated or common motifs using a suffix tree

Marie-France Sagot

Service d'Informatique Scientifique, Institut Pasteur
28, rue du Dr. Roux - Paris
and
Institut Gaspard Monge, Université de Marne la Vallée
2, rue de la Butte Verte - Noisy le Grand

Abstract. We present in this paper two algorithms. The first one extracts repeated motifs from a sequence defined over an alphabet Σ. For instance, Σ may be equal to {A, C, G, T} and the sequence represents an encoding of a DNA macromolecule. The motifs searched correspond to words over the same alphabet which occur a minimum number q of times in the sequence with at most e mismatches each time (q is called the quorum constraint). The second algorithm extracts common motifs from a set of $N \geq 2$ sequences. In this case, the motifs must occur, again with at most e mismatches, in $1 \leq q \leq N$ distinct sequences of the set. In both cases, the words representing the motifs may never be present exactly in the sequences. We therefore speak of the motifs, repeated in a sequence or common to a set of them, as being "external" objects and denote them by the expression "valid models" if they verify the quorum constraint q. The approach we introduce here for finding all valid models corresponding to either repeated or common motifs starts by building a suffix tree of the sequence(s) and then, after some further preprocessing, uses this tree to simply "spell" the models. Assuming an alphabet of fixed size, the total time needed is $O(nN^2\mathcal{V}(e,k))$ using $O(nN^2/w)$ space, where n is the (average) length of the sequence(s), k is the length of the models sought or is the length of the longest possible valid models, w is the size of a word machine and $\mathcal{V}(e,k)$ is the number of words of length k at a Hamming distance at most e from another k-length word. $\mathcal{V}(e,k)$ may be majored by $k^e|\Sigma|^e$. This improves on an algorithm by Waterman [23]. It is also a better time bound than our previous approach [15] for the common motifs problem whenever $N < k|\Sigma|$, and a better space bound when $N/w < k$. It is a better time and space bound in absolute for the repeated motifs problem. The complexities obtained in this second case are $O(n\mathcal{V}(e,k))$ and $O(n)$ respectively. Finally, we suggest how to extend these algorithms to deal with gaps.

1 Introduction

We present in this paper two algorithms. The first one extracts repeated motifs from a sequence, typically of DNA, that is, a sequence defined over Σ = {A, C, G, T}. The motifs searched correspond to words over the same alphabet which occur a minimum number q of times in the sequence with at most e mismatches each time. The second algorithm extracts common motifs from a set of $N \geq 2$ sequences. In this last case, the motifs must occur, again with at most e mismatches, in $1 \leq q \leq N$ distinct sequences of the set.

In both cases, the words representing the motifs may never be present exactly in the sequences. We therefore speak of the motifs, repeated in a sequence or common to a set of them, as being "external" objects and denote them by the term *models*. We also call q the quorum constraint such models have to verify to be considered valid.

Objects such as these were first introduced in the literature by Waterman (under the name of consensus patterns) [7] [21] [22] [23] and later employed by ourselves [15] with the aim of solving the common motifs problem.

The main inconvenient of Waterman's approach is that it obtains the models either by generating all words over Σ^k for some k and then looking for them in the sequences, or by looking only for those models, also of length k, that have a chance of being valid but this requires more space. In the first case, the amount of memory necessary is $O(nN)$ where n is the average length of the sequences, however the time complexity is $O(nNk|\Sigma|^k)$. In the second case, models of length k which have a chance of being valid are those in the e-neighborhood [13] of the words of same length present in the sequences of the set. A model m of length k is said to be in the e-neighborhood of a word u if the (in this case Hamming) distance from m to u is no more than e (i.e. we need at most e substitutions to obtain m from u). The e-neighborhood of a word u of length k contains $\sum_{j=0}^{e} \binom{k}{j} (|\Sigma| - 1)^j \leq k^e|\Sigma|^e$ elements - we denote this number by $\mathcal{V}(e,k)$ ($\mathcal{V}$ for "Vicinity"). The time requirement for the second approach to Waterman's algorithm can be reduced to $O(nNk\mathcal{V}(e,k))$ but this requires now $O(nN|\Sigma|^k)$ space (as given in [23] using a window of length n since they have to remember which models have already been generated). In both cases, the method is therefore limited to small values of k (typically 6). It is also suitable for small alphabets only.

One could improve Waterman's approach by using more efficient techniques of pattern matching with e-mismatches against a text like those by Baeza-Yates [1], Manber [24] [25] or Myers [14] that are based on bit-parallelism. This would reduce the time complexity to $O(nN|\Sigma|^k)$ or $O(nN\mathcal{V}(e,k))$ but one would then still have to deal with a multiplicative factor of $|\Sigma|^k$ in the time or space complexity of the algorithm that would make such approaches prohibitive for big alphabets (if one dealt with proteins for instance instead of DNA sequences) and/or big values of k (as happens with some DNA signals such as the CRP binding site which is believed to be 22 bases long [10]).

Our own algorithm for the common motifs problem [15] generates the models by increasing lengths by simulating the traversal of a lexicographic tree of all possible objects over Σ^+ where at each node x are preserved the occurrences in s of the model m labeling the path from the root to x. The traversal is kept efficient because the tree may be pruned at the branch leading to a model m whenever m does not verify the quorum constraint anymore. Models m' having m as prefix are thus never considered. The counterpart of this approach as against Waterman's is that, since models are built by increasing lengths, a multiplicative factor of k is introduced in the time and space complexities, where k is the length of the models looked for. However, neither complexity depends anymore on $|\Sigma|^k$. The search for all models of length k having occurrences at a Hamming distance at most e in at least q distinct sequences of the set of N takes then $O(nNk|\Sigma|\mathcal{V}(e,k))$ time and $O(nNk)$ space as indicated in the paper. A more space demanding version of the algorithm allows to perform the same search in $O(nN \log k|\Sigma|\mathcal{V}(e,k))$ time but with $O(nN\mathcal{V}(e,k))$ space. Let us observe that, in practice, $(k|\Sigma|)^e$ is much lower than $|\Sigma|^k$ since e/k is approximately 10-15%. Consider for instance the following not untypical values of $k = 16$, $e = 2$ and $|\Sigma| = 4$. We then have $(k|\Sigma|)^e/|\Sigma|^k = 4^{-10}$. In both versions of the algorithm, as the process of model construction is not based on the generation of all those of a given length k, we have the further advantage of not having to fix the value of k beforehand as is the case with Waterman. We may therefore look for valid models of maximum length still verifying the quorum, or for all those between lengths k_1 and k_2 for $1 \leq k_1 \leq k_2 \leq \infty$ (using the first version) while remaining within the same bounds.

When either Waterman's algorithm or our own is applied to the repeated motifs problem, both time and space complexities remain the same except N is now equal to 1.

The new approach we introduce here starts by building a suffix tree of the sequences and then, after some further preprocessing, uses this tree to simply "spell" the valid models. It is therefore this tree that is now traversed to obtain such models.

Assuming an alphabet of fixed size, the tree can be constructed in $O(nN)$ time employing $O(nN)$ space and the preprocessing takes time $O(nN^2/w)$, where w is the size of a word machine, and space $O(nN^2/w)$. The time needed for the model "spelling" operation itself is $O(nN^2\mathcal{V}(e,k))$ with $O(nN)$ additional space required. This is a better time bound for the common motifs problem whenever $N < k|\Sigma|$, and also a better space bound when $N/w < k$. It is a better time and space bound in absolute for the repeated motifs problem since the complexities then become $O(n\mathcal{V}(e,k))$ and $O(n)$ respectively.

Observe that, in this second case, if no errors are allowed, we obtain the same time and space complexities, in $O(n)$, of the best algorithms for identifying repeated motifs [3] [5]. This is not true for the common motifs problem where we have an $O(nN^2)$ time bound whereas Hui obtains an $O(nN)$ bound [9]. His approach should thus be preferred when $e = 0$. Since both algorithms share similar structures, we show that only a minor modification to ours is needed so as to be able to switch to Hui's when e is zero (which seldom happens when one is dealing with biological sequences).

Suffix trees for approximate searches (allowing mismatches and gaps) have been used before, notably by Ukkonen [20] and Cobbs [4]. Although in both cases what is searched is known beforehand making of it a quite different problem (of pattern matching as against pattern extracting), their approaches and ours share some similarity, if only because the same basic data structure (a suffix tree) and the same technique (a form of dynamic programming) are used. However, a much simpler traversal of the tree is required here. This is obviously the case when mismatches only are allowed, but is true also should gaps be permitted. Indeed, we quickly sketch an extension of the algorithm for the repeated motifs problem (the common motifs problem would be handled in a similar way) that deals with gaps by following the same philosophy. The time complexity is then $O(n\mathcal{N}(e,k))$ where $\mathcal{N}(e,k)$ is the number of models m at a (this time) Levenshtein distance at most e from a word of length k. We therefore avoid introducing an additional factor of k in the complexity as would be the case should we adopt Cobbs' method, but may bring in instead a factor in, at most, $2e$ in relation to his approach. Since in general we have $2e < k$, we nevertheless obtain a better bound besides presenting a simpler algorithm.

This paper is organized as follows. We give in section 2 some basic definitions and state the two problems we wish to solve. We then discuss in section 3 the solution to the repeated motifs problem first. We start by recalling the suffix tree data structure and introduce the further preprocessing of it we have to perform before using the tree to obtain the models. We then show how to use the tree to "spell" the models and discuss the time and space complexities obtained. In section 4, we present the modifications we have to do to the previous algorithm to treat the common motifs problem. These concern both the preprocessing of the suffix tree before the spelling operation and the spelling itself. We end by suggesting in section 5 how to extend the first of the two algorithms to be able to deal with gaps as well as mismatches.

2 Basic Definitions and Statement of the Problems

In what follows, we denote by s a (unique) sequence where repeated motifs are searched and by $\{s_i,\ 1 \leq i \leq N$ for some $N \geq 2\}$ a set of sequences from which we want to extract common motifs. In the case of DNA sequences, s and s_i are therefore elements of Σ^+ where $\Sigma = \{$A, C, G, T$\}$. We call u a word in s, or s_i, if the sequence is equal to xuy with $x, y \in \Sigma^*$. The empty word is denoted by λ.

A model m is also an element of Σ^+. It is said to occur (or to be present) in a sequence s if there is at least one word u in s of same length as m and such that $Hamming(m,u) \leq e$ where $Hamming(m,u)$ is the Hamming distance between m and u (it is the minimum number of substitutions needed to transform m into u) and e is a non negative integer.

The problems we wish to solve may then be stated as follows:

The Repeated Motifs Problem. Given a sequence s and two integers $e \geq 0$ and $q \geq 2$, find all models m such that m is present at least q times in s (some of the occurrences of m may overlap);

The Common Motifs Problem. Given a set of N sequences s_i (for $1 \leq i \leq N$) and two integers $e \geq 0$ and $2 \leq q \leq N$, find all models m such that m is present in at least q distinct sequences of the set.

In both cases, models satisfying the above conditions are called *valid*. We are therefore looking for all valid models that correspond either to repeated or common motifs depending on the problem.

In [15], we proposed an algorithm for solving the common motifs problem (actually, a little more than that since we also dealt with gaps). It is easy to modify it so that it can handle the repeated motifs problem as well. However, the approach described there did not try to take advantage of the sequence (or sequences) structure in order to obtain the valid models as was done in [9] but for identically repeated motifs only (no mismatches allowed) or in [11] but for fixed-length motifs that had to appear at least once exactly in the sequence. In the present paper, this underlying structure is exploited to obtain a new model building algorithm dealing with a Hamming distance that has a better complexity in absolute for the first problem stated above, and a better complexity over some range of parameters we explicit later on in the case of the second problem. We recall that the models need never be present exactly in the sequence(s). We start by looking on this new way of solving the problem when repeated motifs are sought.

3 Solving the Repeated Motifs Problem

3.1 Preprocessing

Constructing the Suffix Tree. We do not describe the suffix tree construction, this can be found in either [12], [19] or (for a review of this and other data structures and text algorithms) [6] and [8]. We just recall here some of the basic properties of such structures (these are taken from [12]).

Basic Properties of the Suffix Tree $\mathcal{T}$ of a Sequence s.

1. A branch of $\mathcal{T}$ may represent any nonempty substring of s;
2. Each node of $\mathcal{T}$ that is not a leaf, except for the root, must have at least two offspring branches (compact version of the tree);
3. The strings represented by sibling branches of $\mathcal{T}$ must begin with different symbols of Σ.

Observe that property 2 means that a branch of $\mathcal{T}$ may be labeled by an element of Σ^k for $k \geq 2$ (for space considerations, each branch of $\mathcal{T}$ is in fact labeled by a pair of numbers corresponding to the start and end positions in s of the substring it represents, or its start position and length).

The key feature of a suffix tree is that for any leaf i, the concatenation of the labels of the branches on the path from the root to leaf i spells the suffix of s starting at position i. Reciprocally, the path spelled by every suffix of s leads to a distinct leaf if we assume that the last symbol of s appears nowhere else in s. To achieve this, we just need to concatenate at the end of s a symbol not appearing in Σ.

An example of a suffix tree $\mathcal{T}$ for s = AACCACG is given in Fig. 1. This is adapted from [6].

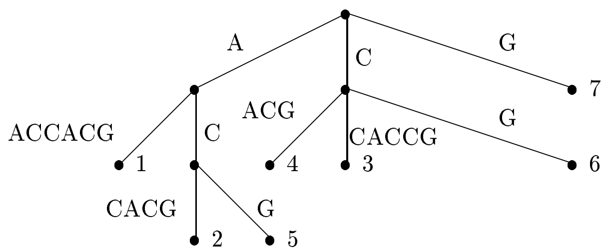

Fig. 1. Compact suffix tree for s = AACCACG

We shall assume we have adopted the McCreight's compact suffix tree construction. However, we need some more information to be added to our tree. This is described next.

Adding Information to the Nodes of the Tree. In the case of the first problem concerning us (finding repeated motifs in a sequence), the further preprocessing of the tree that we need to do is easy to realize.

Indeed, in order to be able to spell the models present at least q times in s, all that remains for us to know is, for each node x of $\mathcal{T}$, how many leaves are contained in the subtree of $\mathcal{T}$ having x as root. Let us denote $Leaves_x$ this number for each node x. This information can be added to the tree by a simple traversal of it.

3.2 Spelling the Models

Let us consider first the case where $e = 0$, that is, no mismatches are allowed. Valid models verify two properties.

1. All their prefixes are also valid;
2. Spelling these models (which is the same as spelling any of their occurrences) leads to a node x in $\mathcal{T}$ for which $Leaves_x$ is at least q.

Once errors are allowed, the first property is still verified but spelling all the occurrences of a valid model may now lead to more than one node of the tree. However, the values of $Leaves_x$ for all such nodes x sums up to at least q also. The second property above is thus replaced by:

2. Spelling all the occurrences of a model m leads to nodes $x_1, \ldots, x_l$ in $\mathcal{T}$ for which $\sum_{j=1}^{l} Leaves(x_j)$ is at least q.

As a matter of fact, it is not occurrences we shall spell, but instead the models that will be read off the tree.

The main difference with our previous approach is therefore that, in the present case, we extract models from the suffix tree of s whereas in [15] we constructed them by a simulated traversal (within bounds) of the lexicographic tree of all possible models, i.e. of all possible elements of Σ^*. This means in particular that occurrences are now grouped into classes and "real" ones, that is occurrences considered as individual words in s, are never directly manipulated. Present case occurrences of a model are thus in fact nodes of the suffix tree (we denote them by the term "node-occurrences") and are extended *in* the tree instead of in the sequence as in [15]. Once the process of model spelling has ended, the start positions of the "real" occurrences of the valid models may be recovered by traversing the subtrees of the nodes reached so far and reading the labels of their leaves. As in [15], the suffix tree need not be entirely traversed, although this time the tree itself must be fully constructed. There are two reasons that may lead to stop the spelling of a model: we have reached the length required for models, the model may not be further extended while remaining valid. For any model, we may also stop the descent down a path in the tree as soon as too many mismatches have been accumulated along it.

The algorithm is a development of the recurrence formula given in the lemma below where x denotes a node of the tree, $father(x)$ its father and err the number of misspellings between the label of the path going from the root to x as against a model m.

Lemma 1. *(x, err) is a node-occurrence of $m' = m\alpha$ with $m \in \Sigma^k$ and $\alpha \in \Sigma$ if, and only if, one of the following two conditions is verified:*

(match) *$(father(x), err)$ is a node-occurrence of m and the label of the branch from $father(x)$ to x is α;*
(subst.) *$(father(x), err-1)$ is a node-occurrence of m and the label of the branch from $father(x)$ to x is $\beta \neq \alpha$.* □

A sketch of the procedure to follow is shown in Fig. 2 for the case where models of a given length k are sought.

In order to do this model spelling operation, we have to make use of the following:

- a set Ext_m of symbols by which a model may be extended at the next step (implemented as a bit-vector);
- a set Occ_m of node-occurrences of a model m. We recall that these correspond in fact to classes of occurrences. Each node-occurrence x is represented by a pair (x, x_{err}) where x_{err} is the number of mismatches between m and the label of the path leading from the root to x in the tree;
- a variable *nbocc* that counts the number of "real" occurrences of the model we are currently trying to extend;

– a function *KeepModel*(m) that either stores all information concerning a valid model of the required length for printing later, or immediately prints this information.

The function *SpellModels* is called with arguments:

$$(0, \lambda, Occ_\lambda = \{(root, 0)\}, Ext_\lambda), \text{ where}$$
$$Ext_\lambda = \begin{cases} \Sigma & \text{if } e > 0 \\ label_b \text{ for branches } b \text{ leaving the root} & \text{otherwise} \end{cases}$$

Where valid models of maximum length are sought, we just need to change lines 1 and 2 into the code given in Fig. 3. Variable k_{max} is initialized to 0 before first entering function *SpellModels*.

For the sake of simplicity, the code shown here assumes we are dealing with an uncompact version of the tree (that is, with a trie). Using a compact version affects the operations done in lines 9, 10, 12, 14, 15 and 17. Indeed, we need in that case to know at any given step whether we are:

– at a node x, or
– inside a branch b between nodes x and x'

and, if we can travel down $\mathcal{T}$ with a symbol α, whether that:

– gets us to a new node x', or
– keeps us inside branch b.

This means that additional information has to be kept relative to each node-occurrence, and, consequently, extending such an occurrence implies more work. This however increases the algorithm's time and space complexity by a constant factor only.

3.3 Complexity

Assuming an alphabet of fixed size, a compact suffix tree $\mathcal{T}$ may be constructed in $O(n)$ time where n is the length of sequence s and occupies $O(n)$ space [12] [19].

Adding information to the nodes of the tree as described in section 3.1 takes time $O(n)$ and requires $O(1)$ space per node of $\mathcal{T}$.

Concerning the spelling operation, we have that:

Lemma 2. *Spelling all valid models for the Repeated Motifs Problem given $\mathcal{T}$ requires $O(n\mathcal{V}(e,k))$ time where k is either the length of the models sought or is a maximum length.*

Proof. Let us consider valid models of length k are searched for. Spelling them requires descending down the tree of at most k levels (we may sometimes not reach that level if a given model stops verifying the quorum constraint q at an earlier stage). At level k, there are $p \leq n$ nodes (since there are exactly

$SpellModels(l, m, Occ_m, Ext_m)$

1. if $(l = k)$
2. $KeepModel(m)$
3. else if $(l < k)$
4.   for each symbol α in Ext_m
5. $nbocc = 0$
6. $Ext_{m\alpha} = \emptyset$
7. $Occ_{m\alpha} = \emptyset$
8.   for each pair (x, x_{err}) in Occ_m
9.   if there is a branch b leaving node x with a label starting with α
10.   add to $Occ_{m\alpha}$ the pair (x', x_{err}) where x' is the node reached by following branch b from x
11. $nbocc = nbocc + Leaves_{x'}$
12. $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x' & \text{if } x_{err} = e \\ \Sigma & \text{otherwise} \end{cases}$
13.   if $x_{err} < e$
14.   for each branch b leaving x except the one labeled by α if it exists
15.   add to $Occ_{m\alpha}$ the pair $(x', x_{err} + 1)$ where x' is the node reached by following branch b from x
16. $nbocc = nbocc + Leaves_{x'}$
17. $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x' & \text{if } x_{err} = e - 1 \\ \Sigma & \text{otherwise} \end{cases}$
18.   if $nbocc \geq q$
19. $SpellModels(l + 1, m\alpha, Occ_{m\alpha}, Ext_{m\alpha})$

Fig. 2. Sketch of the procedure for spelling models corresponding to repeated motifs

1. if $(l \geq k_{max})$
2.   if $(l > k_{max})$
3.   throw away all preceding kept models
4. $k_{max} = l$
5. $KeepModel(m)$

Fig. 3. Modification to apply to the code of Fig. 2 in order to generate valid models of maximum length - the lines given here replace lines 1 and 2

n leaves in $\mathcal{T}$ and the number of leaves is greater than the number of nodes at any particular level k above that of the lowest leaf). From these p nodes, there are p paths up to the root of $\mathcal{T}$ (note that in our algorithm, we in fact go *down* the paths, not up) and $\mathcal{V}(e,k)$ ways of misspelling their labels, that is spelling the labels with at most e mismatches. This corresponds also to an upper bound to the total number of visits that may have to be done to the branches of $\mathcal{T}$ (or, equivalently, its nodes) in order to obtain the requested models. Since each visit to a branch costs us constant time (basically, we need to increment a counter, add a node-occurrence to one set and a list of symbols to another set - a constant time operation if the list is implemented as a boolean array), then the total number of operations needed to spell all the valid models given $\mathcal{T}$ is bounded over by $p\mathcal{V}(e,k)$, that is, by $O(n\mathcal{V}(e,k))$. □

Lemma 3. *Spelling all valid models for the Repeated Motifs Problem given $\mathcal{T}$ requires $O(n)$ space.*

Proof. The space required is that of the tree plus that of the auxiliary structures Occ_m and Ext_m. We need to keep such structures for the model m currently being treated and for all its prefixes (we are traversing the tree recursively). However, the sets Occ_m are all disjoint between them so that $\sum_{m' \ prefix \ of \ m} Occ_{m'} \leq n$. Since $\sum_{m' \ prefix \ of \ m} Ext_{m'} \leq k|\Sigma|/w$, the total space complexity is $O(n + n + k|\Sigma|/w) = O(n)$ if we assume a fixed length alphabet. □

Note that if $e = 0$, then $\mathcal{V}(e,k) = 1$. Let us point out also that $\mathcal{V}(e,k)$ is an upper bound for the number of models that corresponds to the maximum size of the output and is seldom observed.

4 Solving the Common Motifs Problem

4.1 Generalized Suffix Trees

Trees for representing all the suffixes of a set of sequences $\{s_i,\ 1 \leq i \leq N$ for some $N \geq 2\}$ are called generalized suffix trees and are constructed in a way very similar to the construction of the suffix tree for a single sequence [2] [9]. We denote these generalized trees by $\mathcal{GT}$. They share all the properties of a suffix tree given in section 3.1 with, in property 1, sequence s substituted by sequences $s_1, \ldots, s_N$.

In particular, a generalized suffix tree $\mathcal{GT}$ verifies the fact that every suffix of every sequence s_i in the set leads to a distinct leaf. When $p \geq 2$ sequences have a same suffix, the generalized tree has therefore p leaves corresponding to this suffix, each associated with a different sequence. To achieve this property during construction, we just need to concatenate to each sequence s_i of the set a symbol that is not in Σ and is specific to that sequence.

4.2 Adding Information to the Nodes of the Tree

If the construction of a $\mathcal{GT}$ for a set $\{s_i\}$ of sequences is similar to that of a suffix tree $\mathcal{T}$ for a single sequence s, it is not enough anymore to know the values of $Leaves_x$ for each node x in $\mathcal{GT}$ in order to be able to solve the Common Motifs Problem.

We could then modify our preprocessing of the tree so that we calculate, for each node x, no longer the number of leaves in the subtree of $\mathcal{GT}$ having x as root, but the number of different sequences those leaves refer to. Computing this number is what is called the "Color Size Problem" by Hui [9]. The color set size of a node x is precisely the number of different leaf colors in the subtree rooted at x, where a leaf is assigned color i if it represents a suffix of s_i. Let us call this new number CSS_x as in [9].

Knowing CSS_x for all nodes x is however all that is required only in the case where $e = 0$ (in this case, a model has only one node-occurrence). When $e > 0$, we also have to be able to tell which colors are common to 2 or more nodes of the tree.

In order to do that, we need to associate to each node x in the $\mathcal{GT}$ of a set $\{s_i\}$ an array, denoted $Colors_x$, of dimension N that is defined by:

$$Colors_x[i] = \begin{cases} 1 \text{ if at least one leaf in the subtree} \\ \quad \text{rooted at } x \text{ represents a suffix of } s_i \\ 0 \text{ otherwise} \end{cases} \qquad (1 \leq i \leq N) \ .$$

$Colors_x$ may be implemented as a bit vector, or as N/w bit-vectors if $N > w$ where w is the size of a word machine.

The array $Colors_x$ for all x may be obtained by a simple traversal of the tree with each visit to a node taking $O(N/w)$ time (for adding N/w bit-vectors). The additional space required is $O(N/w)$ per node. We shall also use the information provided by CSS_x which Hui showed can be obtained in $O(nN)$ time and uses $O(1)$ space per node. Considering CSS_x is not strictly necessary but may be useful in practice as is suggested when we analyze the complexity of this algorithm later on.

4.3 Spelling the Models

For ease of presentation, we assume here once more that we are looking for all valid models of a fixed length k, and that we are working with an uncompact version of the $\mathcal{GT}$. A sketch of the algorithm for solving the Common Motifs Problem is given in Fig. 4. We use the same auxiliary structures Occ_m and Ext_m as in the previous algorithm, to which we add the following:

- a variable CSS_x as defined in the previous section;
- a boolean array $Colors_x$ (possibly N/w arrays if $N > w$) as defined in the previous section;
- a variable *minseq* that indicates the minimum of CSS_x for all node-occurrences x of the extended model;

- a variable $maxseq$ that indicates the sum of CSS_x for all node-occurrences x of the extended model;
- a boolean array $Color_m$ (possibly N/w arrays if $N > w$) defined by:

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

Observe that, in all cases, we have:

$$minseq \leq \text{(number of distinct sequences the model is present in)} \leq maxseq.$$

4.4 Complexity

What produces an increase in the complexity of the algorithm of Fig. 4 in relation to that of Fig. 2 concerns simply the data structure $Colors_x$: the time needed to create and manipulate it, and the space required to store it.

The space requirement of $Colors_x$ is $O(N/w)$ per node if it is implemented as a bit-vector having same size w as a word machine. The total space requirement of the algorithm is therefore now bounded over by $O(nN^2/w)$. This is smaller than our previously obtained bound [15] of $O(nNk)$ when $N/w < k$.

Creating $Colors_x$ for every node x of the tree takes time $O(nN^2/w)$, however manipulating it, in particular performing the operation indicated in line 31, requires $O(N)$ time per model. Since there can be $O(nN\mathcal{V}(e,k))$ valid models in the worst case, the algorithm's time complexity becomes $O(nN^2\mathcal{V}(e,k))$. This is a better bound than the one given in [15] for $N < k|\Sigma|$.

The tests of lines 26 and 29 should improve the algorithm's behaviour on average. Observe that the test of line 29 has more chance of being true at the beginning of the algorithm (where models match almost everything) while that of line 26 has a better chance of being verified the longer the model is (because the number of its occurrences will then be quite close to q).

As mentioned in the introduction, when $e = 0$ we do not obtain Hui's better bound of $O(nN)$. In this case though, we only need to remove lines 7, 16, 24, 31 and 32 to fall back to the algorithm Hui introduced in [9]. Observe this also means getting rid of the $Colors$ structure that is no longer necessary. It is easy to modify the algorithm of Fig. 4 so that the instructions contained in the lines just indicated are performed only if $e > 0$.

5 Sketch of Extension Dealing with Gaps

We sketch in this section how to extend the algorithms so as to be able to treat gaps as well as mismatches. This is done only for the repeated motifs problem. The common motifs problem would be dealt with in a quite similar manner. The algorithm is presented without further ado in Fig. s 5 and 6. Node-occurrences must be maintained in Occ_m for m a model in the order in which they would be encountered if the tree were traversed in a depth-first manner. This preorder follows naturally from the way nodes are processed at each step of the algorithm.

```
SpellModels(l, m, Occ_m, Ext_m)

if (l = k)
    KeepModel(m)
else if (l < k)
    for each symbol α in Ext_m
        maxseq = 0
        minseq = ∞
        Colors_mα is initialized with no colors
        Ext_mα = ∅
        Occ_mα = ∅
        for each pair (x, x_err) in Occ_m
            if there is a branch b leaving node x with a label starting with α
                add to Occ_mα the pair (x', x_err) where x' is the node reached by
                following branch b from x
                maxseq = maxseq + CSS_x'
                if CSS_x' < minseq
                minseq = CSS_x'
                add colors in Colors_x to Colors_mα
                Ext_mα = { Ext_mα ∪ label_b' for b' leaving x'   if x_err = e
                         { Σ                                      otherwise
            if x_err < e
                for each branch b leaving x except the one labeled α if it exists
                    add to Occ_mα the pair (x', x_err + 1) where x' is the node reached
                    by following branch b from x
                    maxseq = maxseq + CSS_x'
                    if CSS_x' < minseq
                        minseq = CSS_x'
                        add colors in Colors_x to Colors_mα
                        Ext_mα = { Ext_mα ∪ label_b' for b' leaving x'   if x_err = e - 1
                                 { Σ                                      otherwise
        if maxseq < q
            return (no hope)
        else
            if minseq ≥ q
                SpellModels(l + 1, mα, Occ_mα, Ext_mα)
            else if the number of bits at 1 in Colors_mα is no less than q
                SpellModels(l + 1, mα, Occ_mα, Ext_mα)
```

Fig. 4. Sketch of the procedure for spelling models corresponding to common motifs

We do not prove it here, but the only thing that changes in the complexity of the algorithm is that $\mathcal{V}(e,k)$ is replaced by $\mathcal{N}(e,k)$ where $\mathcal{N}(e,k)$ is the number of models m at a (this time) Levenshtein distance at most e from a word of length k. This comes from the fact that, since each node of the tree is considered at most once as a node-occurrence, Occ_m remains bounded over by $O(n)$. There may simply now be more models.

One can think of the operation performed by the procedure $Treat$ of Fig. 6 as adding the last row of a dynamic programming matrix of model m against the suffix tree of s as in [20] or [4]. As mentioned in the introduction however, the current algorithm has a different way of accounting for the "real" occurrences of a model than the one by Cobbs [4]. Indeed, in his case, for each position i in s, only one occurrence is kept that ends at i. Doing that however implies verifying certain things and this may cost him as much as k additional operations per occurrence. In our case, although we may keep up to $2e$ node-occurrences per valid ending position (the total number of nodes remaining less than Cn for a small constant C), we still get a better time bound since in general $2e < k$. The algorithm is also simpler. Furthermore, it may be interesting in some cases (e.g. when searching for tandem repeats [16]) to know both the start and end positions of an occurrence.

```
SpellModels(l, m, Occ_m, Ext_m)

if (l = k)
    KeepModel(m)
else if (l < k)
    for each symbol α in Ext_m
        nbocc = 0
        Ext_mα = ∅
        Occ_mα = ∅
        for each pair (x, x_err) in Occ_m
            remove (x, x_err) from Occ_m
            Treat(Occ_m, Occ_mα, Ext_mα, Leaves, x, x_err, α, nbocc, 0)
        if nbocc ≥ q
            SpellModels(l + 1, mα, Occ_mα, Ext_mα)
```

Fig. 5. Sketch of the procedure for spelling models corresponding to repeated motifs when gaps as well as mismatches are allowed (internal procedure *Treat* is given in Fig. 6)

Treat$(Occ_m, Occ_{m\alpha}, Ext_{m\alpha}, Leaves, x, x_{err}, \alpha, nbocc, level)$

1. if $(level = 0)$ /* deletion */
2. if $x_{err} < e$
3. add to $Occ_{m\alpha}$ the pair $(x,\ x_{err} + 1)$
4. $nbocc = nbocc + Leaves_x$
5. $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x & \text{if } (x_{err}+1) = e \\ \Sigma & \text{otherwise} \end{cases}$
6. for each x' obtained by following, in lexicographic order, a branch (labeled β) from x
7. if $(x',\ x'_{err})$ is the next pair in Occ_m
8. remove $(x',\ x'_{err})$ from Occ_m
9. let $min_{err} = \min \begin{cases} x_{err} & \text{if } \beta = \alpha \text{ /* match */} \\ x_{err}+1 & \text{if } \beta \neq \alpha \text{ /* substitution */} \\ x'_{err}+1 & \text{/* deletion */} \\ x_{err}+1 & \text{/* insertion */} \end{cases}$
10. if $min_{err} \leq e$
11. add to $Occ_{m\alpha}$ the pair $(x',\ min_{err})$
12. $nbocc = nbocc + Leaves_{x'}$
13. $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x' & \text{if } min_{err} = e \\ \Sigma & \text{otherwise} \end{cases}$
14. *Treat*$(Occ_m, Occ_{m\alpha}, Ext_{m\alpha}, Leaves, x', min_{err}, \alpha, nbocc, level + 1)$
15. else
16. remove from Occ_m the sons of x' and all the sons thereof recursively
17. else
18. let $min_{err} = \min \begin{cases} x_{err} & \text{if } \beta = \alpha \text{ /* match */} \\ x_{err}+1 & \text{if } \beta \neq \alpha \text{ /* substitution */} \\ x_{err}+1 & \text{/* insertion */} \end{cases}$
19. if $min_{err} \leq e$
20. add to $Occ_{m\alpha}$ the pair $(x',\ min_{err})$
21. $nbocc = nbocc + Leaves_{x'}$
22. $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x' & \text{if } min_{err} = e \\ \Sigma & \text{otherwise} \end{cases}$

Fig. 6. Procedure *Treat*, used by the algorithm for spelling models corresponding to repeated motifs, when gaps as well as mismatches are allowed

6 Future Work

It is not too difficult to see that the new approach to approximate motif extraction presented in this paper may be extended to deal with the special kinds of alphabets required for protein sequence analysis [18] and with combinatorial alphabets such as introduced in [17]. This will be explored and formalized in a future paper.

The present algorithm should also help deal with models that, although different, have sets of "real" occurrences (or, equivalently, of node-occurrences) that are identical or included in the set of another model. This is a problem often encountered and it may be appropriate to get rid of included sets. It is not completely trivial how to do it in an efficient way.

Acknowledgments

The author would like to thank Maxime Crochemore for his constant encouragements and to all the last eight months visitors to the Institut Gaspard Monge, in particular Véronique Bruyère, Clelia de Felice, Gene Myers, Antonio Restivo, Paul Schupp and Thomas Wilke, for having helped create an even more stimulating environment in which to work. Thanks is also due to Dan Gusfield for being such a good advocate of the merits of suffix trees. Finally, I would like to thank all the referees for their comments and suggestions, in particular the anonymous referee who very kindly sent to me his/her implementation in perl of the first algorithm presented in this paper.

References

1. R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35:74–82, 1992.
2. P. Bieganski, J. Riedl, J. V. Carlis, and E.M. Retzel. Generalized suffix trees for biological sequence data: applications and implementations. In *Proc. of the 27th Hawai Int. Conf. on Systems Sci.*, pages 35–44. IEEE Computer Society Press, 1994.
3. B. Clift, D. Haussler, R. McConnell, T. D. Schneider, and G. D. Stormo. Sequence landscapes. *Nucleic Acids Res.*, 14:141–158, 1986.
4. A.L. Cobbs. Fast identification of approximately matching substrings. In Z. Galil and E. Ukkonen, editors, *Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, pages 41–54. Springer Verlag, 1995.
5. M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf. Proc. Letters*, 12:244–250, 1981.
6. M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.
7. D. J. Galas, M. Eggert, and M. S. Waterman. Rigorous pattern-recognition methods for DNA sequences. Analysis of promoter sequences from E*scherichia coli. J. Mol. Biol.*, 186:117–128, 1985.
8. D. Gusfield. *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology.* Cambridge University Press, 1997.

9. L. C. K. Hui. Color set size problem with applications to string matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer-Verlag, 1992.
10. C. E. Lawrence and A. A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: struct., funct., and genetics*, 7:41–51, 1990.
11. C. Lefevre and J.-E. Ikeda. A fast word search algorithm for the representation of sequence similarity in genomic DNA. *Nucleic Acids Res.*, 22:404–411, 1994.
12. E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23:262–272, 1976.
13. E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12:345–374, 1994.
14. E. W. Myers. 1997. personal communication.
15. M.-F. Sagot, V. Escalier, A. Viari, and H. Soldano. Searching for repeated words in a text allowing for mismatches and gaps. In R. Baeza-Yates and U. Manber, editors, *Second South American Workshop on String Processing*, pages 87–100, Viñas del Mar, Chili, 1995. University of Chili.
16. M.-F. Sagot and E. W. Myers. Identifying satellites in nucleic acid sequences. 1998. submitted to RECOMB 1998.
17. M.-F. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. In D. Hirschberg and G. Myers, editors, *Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 186–208. Springer-Verlag, 1996.
18. M.-F. Sagot, A. Viari, and H. Soldano. Multiple comparison: a peptide matching approach. *Theoret. Comput. Sci.*, 180:115–137, 1997. presented at *Combinatorial Pattern Matching 1995*.
19. E. Ukkonen. Constructing suffix trees on-line in linear time. pages 484–492. IFIP'92, 1992.
20. E. Ukkonen. Approximate string matching over suffix trees. In Z. Galil A. Apostolico, M. Crochemore and U. Manber, editors, *Combinatorial Pattern Matching*, volume 684 of *Lecture Notes in Computer Science*, pages 228–242. Springer-Verlag, 1993.
21. M. S. Waterman. Multiple sequence alignments by consensus. *Nucleic Acids Res.*, 14:9095–9102, 1986.
22. M. S. Waterman. Consensus patterns in sequences. In M. S. Waterman, editor, *Mathematical Methods for DNA Sequences*, pages 93–116. CRC Press, 1989.
23. M. S. Waterman, R. Arratia, and D. J. Galas. Pattern recognition in several sequences: consensus and alignment. *Bull. Math. Biol.*, 46:515–527, 1984.
24. S. Wu and U. Manber. Agrep - a fast approximate pattern-matching tool. pages 153–162, San Francisco, CA, 1992. USENIX Technical Conference.
25. S. Wu and U. Manber. Fast text searching allowing errors. *Commun. ACM*, 35:83–91, 1992.