

Extracting Common Motifs under the Levenshtein Measure: Theory and Experimentation

Ezekiel F. Adebiyi and Michael Kaufmann

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
72076 Tübingen, Germany
{adebiyi,mk}@informatik.uni-tuebingen.de

Abstract. Using our techniques for extracting approximate non-tandem repeats[1] on well constructed maximal models, we derive an algorithm to find common motifs of length P that occur in N sequences with at most D differences under the Edit distance metric. We compare the effectiveness of our algorithm with the more involved algorithm of Sagot[17] for Edit distance on some real sequences. Her method has not been implemented before for Edit distance but only for Hamming distance[12,20]. Our resulting method turns out to be simpler and more efficient theoretically and also in practice for moderately large P and D .

1 Introduction

The problem we consider here, called common motifs extraction, is stated as follows: Given a set of $N \geq 2$ subject strings $\{S_i, | i = 1..N\}$, each of average length n over a finite alphabet Σ and a threshold D , find common motifs of length P that occur with at most D differences (or simply D -occurrence) in $1 \leq q \leq N$ distinct sequences of the set. We consider this problem with respect to the Levenshtein metric (or simply Edit distance). We consider here the simple unit measure, whereby, the Edit distance between two strings $S_1[1..i]$ and $S_2[1..j]$, $\delta(i, j)$ is defined as follows:

$$\delta(i, j) = \min[\delta(i - 1, j) + 1, \delta(i, j - 1) + 1, \delta(i - 1, j - 1) + t(i, j)],$$

where $t(i, j)$ is 0, if $S_1(i) = S_2(j)$ else 1. As far as we know, this work will be the first that discusses this problem with respect to Edit distance not only theoretically (as been done before) but also experimentally.

The above problem has found applications in finding the DNA binding sites. There are various methods to extract common motifs. See Adebiyi[2] for an overview. Most of the approach made use of the Hamming distance metric. It is only the work of Roche and Tompa[15] and Sagot[17] that considered extracting common motifs under Edit distance but did so theoretically only. Various pointers[13,9] have indicated that extracting common motifs under Edit distance is also an important biological problem. It is this open problem that we consider in this paper.

Our contributions are in two folds: First, we have implemented the interesting but involved algorithm of Sagot[17] for motifs finding for Edit distance. The theoretical running time of this algorithm is $O(N^2 n P^D |\Sigma|^D)$. It has only been implemented before for Hamming distance[20,12]. Second, we present an alternative algorithmic approach also for Edit distance based on the concept described by Adebiyi et al.[1]. The resulting algorithm is simpler and more efficient for moderately large P and D in theory as well as in practice. It runs in $O(N \cdot n + D(N \cdot n)^{1+pow(\varepsilon)})$ expected time for $D \leq 2 \log_{|\Sigma|}(N \cdot n)$, where $\varepsilon = D/P$ and $pow(\varepsilon)$ is an increasing, concave function that is 0 when $\varepsilon = 0$ and about 0.9 for DNA and protein sequences. We implemented this algorithm in C and compare its effectiveness using our implementation of Sagot's algorithm on sequences that include B. Subtilis[7], yeast *S. cerevisiae*[10] and H. Pylori[18].

This paper is structured as follows. In sections 2 and 3, we present the algorithm of Sagot and our algorithm for Edit distance, and in section 4, we discuss our experimental experience with the algorithms from sections 2 and 3 and also, with our implementation of Sagot's algorithm for Hamming distance.

2 Basic Concepts and Sagot's Algorithm for Edit Distance

Here, we will describe in detail the algorithm of Sagot[17] for Edit distance, since we use some of the concepts later on for the new algorithm, and start with some definitions. A *model* of length P is a pattern over Σ^P for some positive $P \geq 1$ and a *valid model* is a model that present a single representation for all reoccurrences of a motif. A valid model is elsewhere also known as a consensus. The distance between a valid model and the reoccurrences of the motif that it represents is not more than D . In particular, we refer to a motif and its reoccurrences in the sequences using its valid model.

Sagot's method is basically based on the concept of validating all words/models (they are elsewhere known as *external objects*, and are the candidates for motifs) over Σ^P that occur in $1 \leq q \leq N$ distinct sequences of the set $\{S_i, | i = 1 \dots N\}$. The generation or *SPELLING* after Sagot of each character of the models is done simultaneously as we search them in the preprocessed generalized suffix tree (henceforth known as *GST*, see below for the definition) of the sequences. *GST* is the suffix tree that represent the suffixes of multiple sequences. The *SPELLING* of a model stops, if we have reached the length required for models or the models may not be further extended while remaining valid. If we introduce errors (Hamming or Edit distance), for any model, we may also stop the descent down a path in *GST* as soon as too many insertions, deletions, or substitutions have been accumulated along it. The models of length P , which have a chance to be valid, are those in the D -neighborhood of the words present in the sequences of the set. Let u be a word present in the sequences. Model X is in the D -neighborhood of u , if the distance between X and u is not more than D . For Hamming distance, Sagot[17] proved that the D -neighborhood of a word of a length P contains

$$V(D, P) = \sum_{i=1}^D \binom{P}{i} (|\Sigma| - 1)^i \leq P^D |\Sigma|^D$$

elements. Recently, Crochemore and Sagot[5] indicated that this also bounds the number of D -neighborhood for Edit distance from above.

To introduce the concept of generalized suffix tree, for brevity, we adopt the combinatorial definitions given by Blumer[3].

Definition 1. *A substring, x , of a string S , occurs in m left (right) contexts if x occurs at least m times, with these occurrences preceded (followed) by at least m distinct characters.*

Definition 2. *Given a finite set s of strings and x, y in s , y is a minimal extension of x (in s) if x is a prefix of y and there is no z in s shorter than y such that x is a prefix of z and z a prefix of y .*

We can now define a suffix tree for a single string S of size n as follows,

Definition 3. *Let s be the set of all substrings of S . The suffix tree (henceforth known as ST) for S is the directed graph (V, E) , where V is the set of substrings of S that are either suffixes of S or that occur in S in at least two right contexts, and E is the set of all directed edges (x, y) , x, y in s , such that y is a minimal extension of x .*

In particular, ST of a string S of size n is a tree with $O(n)$ edges and internal vertices and n external vertices (the leaves). Each leaf in the suffix tree is associated with an index i ($1 \leq i \leq n$). The edges are labeled with characters so that the concatenation of the labeled edges on the path from the root to the leaf with index i is the i th suffix of S . Here, ST is used to represent a single string. This can be extended to represent multiple strings $\{S_i, | i = 1 \dots N\}$. The resulting tree is called GST. When two or more input strings have the same suffix, GST has several leaves corresponding to that suffix, but each corresponds to a different input string.

Before giving a detailed description of Sagot algorithm for Edit distance, note that SPELLING all occurrences of a model if errors are not allowed leads to a node x in the suffix tree, while it leads to more than one node $\{x_1, x_2, \dots\}$, once errors are allowed. Her algorithm is based on the following recurrence formula.

Lemma 1. [5] *Let x be a node in the suffix tree and $err \leq D$. (x, err) is a node-occurrence of $m' = m\alpha$ with $m \in \Sigma^k$ and $\alpha \in \Sigma$ (i.e node x is a substring approximate to m' within distance err) if and only if, one of the following conditions is verified:*

- (match)** *(father(x), err) is a node-occur. of m and the label of father(x) to x is α ;*
- (substitution)** *(father(x), $err-1$) is a node-occurrence of m and the label of the arc from father(x) to x is $\beta \neq \alpha$;*
- (deletion)** *(x , $err-1$) is a node-occurrence of m ;*
- (insertion)** *(father(x), $err-1$) is a node-occurrence of $m\alpha$.*

To perform the SPELLING step, some additional information has to be added to the nodes of the *GST*. The first preprocessing task, is to calculate for each node x , the number CSS_x [8] of different sequences of the set $\{S_i, | i = 1 \dots N\}$, that the leaves in the subtree rooted at x refer to. This has been shown to be solvable in $O(N \cdot n)$ time by Hui. Furthermore, to each x , a boolean array $Colors_x$ of dimension N is associated, whose entries are used to determine which sequences are common to two or more nodes of the tree. The i -th entry of $Colors_x$ is 1 if at least one leaf in the subtree rooted at x represents a suffix of string S_i and 0 otherwise. $Colors_x$ for all nodes x may be obtained by a simple traversal of the tree.

The pseudo-code of the Edit distance algorithm, as outlined by Sagot[17] but implemented by us, are presented below. The following auxiliary structures are used:

1. a set Ext_m of symbols by which a model may be extended at the next step;
2. a set Occ_m of node-occurrences of a model m . Recall that these correspond to classes of occurrences. Each node-occurrence x is represented by a pair (x, x_{err}) where x_{err} is the number of insertions, deletions or mismatches between m and the label of the path leading from the root to x in the tree;
3. a variable CSS_x as defined above;
4. a boolean array $Colors_x$ as also defined above;
5. $minseq$ that indicates the minimum of CSS_x for all node-occurrences x of the extended model;
6. a variable $maxseq$ that indicates the sum of CSS_x for all node-occurrences x of the extended model;
7. a $Color_m$, whose i -th entry is 1 if m occurs in string S_i and 0 otherwise;
8. a function $KeepModel(m)$, that either stores all information concerning a valid model of required length for printing later, or immediately prints this information.

In procedure *SPELLMODELS* (fig. 1), the models m are spelled as they are searched against the words in the sequences via the paths that encoded them in *GST*. Note that the Edit operations can be rewritten in any of the following forms: $a \rightarrow \varepsilon$ (a deletion), $\varepsilon \rightarrow a$ (an insertion), or $a \rightarrow b$ (a change or substitution), where $a, b \in \Sigma$, $a \neq b$, and ε is the empty string. Let the characters at the left-hand side of the binary operator \rightarrow come from the substrings of the node occurrences, while that on the right-hand side come from the models. To spell the models, while allowing its re-occurrences to contain gaps, sub-procedure *TREAT* in *SPELLMODELS* of fig. 1 simulates the adding of the last row of a dynamic programming matrix of the model m against the nodes of *GST*[19].

TREAT is called recursively as $TREAT(Occ_m, Occ_{m\alpha}, Ext_{m\alpha}, Colors_{m\alpha}, minseq, maxseq, x, x_{err}, \alpha, level + 1)$ and works as follows. At $level = 0$, the edit script of deleting a character from the model $m\alpha$ is carried out if $x_{err} < D$ and parameters $maxseq, minseq, Colors_{m\alpha}, Ext_{m\alpha}$ and $Occ_{m\alpha}$ are updated accordingly using the node x in concern. At $level > 0$, the nodes x' of the subtree rooted at x are pruned accordingly using their min_{err} , i.e., the minimum error that exist between them and the model considering insertions, deletions

and substitutions. And if $min_{err} \leq D$, parameters $maxseq$, $minseq$, $Colors_{m\alpha}$, $Ext_{m\alpha}$ and $Occ_{m\alpha}$ are again updated accordingly using the nodes x and x' in concern.

Note that in all cases, the number of distinct sequences, the model occurs in, is in the range between $minseq$ and $maxseq$. For simplicity, we assumed that motifs of a fixed length P are been sought for. The function *SPELLMODELS* is called with the arguments $(0, \lambda, Occ_\lambda = \{(root, 0)\}, Ext_\lambda)$, where $Ext_\lambda =$

$$\begin{cases} \Sigma & : D > 0 \\ label_b \text{ for branches } b \text{ from root} & : otherwise \end{cases}$$

Note that the number of all possible models in line 4 that does not necessarily represent a motif is $|\Sigma|^P$ and the loop of line 8 is executed $O(N \cdot n)[3]$ times. In fact, the number of nodes x , considered in loop of line 8, are further reduced in sub-procedure *TREAT*. The models which might be valid (those representing motifs) are those in the D -neighborhood of the words in the sequences of the set $\{S_i, | i = 1 \dots N\}$. Therefore, the nested loops starting in line 4 and line 8 are executed $O(N \cdot n \cdot V(D, P))$ times, where $V(D, P) \leq P^D |\Sigma|^D$. Within these loops, the dominating operation is the operation of line 14 and this can be carried out in $O(N)$ time. Therefore, it follows that for all possible models in Σ^P , the running time of the algorithm *SPELLMODELS*(l, P, Occ_m, Ext_m) is $O(N^2 n P^D |\Sigma|^D)$.

1. Procedure *SPELLMODELS*(l, P, Occ_m, Ext_m)
2. if ($l = P$) then *KeepModel*(m)
3. else if ($l < P$)
4. for each symbol α in Ext_m
5. $maxseq = 0, minseq = \infty$
6. $Colors_{m\alpha}$ is initialized with no colors
7. $Ext_{m\alpha} = 0, Occ_{m\alpha} = 0$
8. for each pair (x, x_{err}) in Occ_m
9. remove(x, x_{err}) from Occ_m
10. TREAT($Occ_m, Occ_{m\alpha}, Ext_{m\alpha}, Colors_{m\alpha}, minseq, maxseq, x, x_{err}, \alpha, level = 0$)
11. if ($maxseq < q$) return (no hope)
12. else if ($minseq \geq q$)
13. *SPELLMODELS*($l + 1, m\alpha, Occ_{m\alpha}, Ext_{m\alpha}$)
14. else if the number of bits at 1 in $Colors_{m\alpha}$ is not less than q
15. *SPELLMODELS*($l + 1, m\alpha, Occ_{m\alpha}, Ext_{m\alpha}$)

Fig. 1. Pseudocode of the procedure for SPELLING models corresponding to common motifs when gaps as well as mismatches are allowed.

3 An Alternative Algorithmic Approach

We develop another algorithmic approach by a careful generation of potential models using the adapted techniques of Adebiyi et al.[1]. Any string S of size n is a combination of exact repeated and non-repeated substrings respectively. Two types of exact repeats exist: maximal and non-maximal repeats. An exact repeat β is maximal if $a\beta c$ and $b\beta d$ occur in S for some $a \neq b$ and $c \neq d$, $a, b, c, d \in \Sigma$. Otherwise it is non-maximal. Note that a maximal repeat might be a proper substring of another maximal repeat. Therefore, a supermaximal repeat is a maximal repeat that does not occur as a substring of any other maximal repeat. Models are constructed by partitioning them into repeats-oriented equivalence classes and avoiding those models that are contained in another maximal models or that will extend to the same motif. Here, we use the ideas developed in Adebiyi et al.[1]. The number of such models is linear in the total length of the input strings.

Using the foregone details, the algorithm is based on a clever construction of models that are extended into common motifs.

3.1 The Algorithm

The algorithm is divided into two steps. First, we extract the models, second, we extend these models to derive common motifs. We now discuss how the models are constructed. We begin with formal definitions[6] for identifying *maximal and supermaximal repeats respectively* in a suffix tree, ST .

Definition 4. *For each position i in string S , character $S(i - 1)$ is called the left character of i . The left character of a leaf of ST is the left character of the suffix position represented by that leaf.*

Definition 5. *A node v of ST is called left diverse, if at least two leaves in v 's subtree have different left characters. By definition, a leaf cannot be left diverse.*

Lemma 2. *The string α , labeling the path to a node v of ST is a maximal repeat if and only if v is left diverse. A left diverse internal node v represents a supermaximal repeat α , if and only if all of v 's children are leaves and each has a distinct left character.*

Lemma 3. [1] *Each maximal or non-maximal repeat is contained in some supermaximal repeats.*

We have the following observations, the first one is straight forward.

Observation 1. A model substring must be extendable in all strings in $\{S_i, | i = 1 \dots N\}$. That is, its left index in each string shifted by $P - 1$ positions to the right, must be less than or equal to the end index of each string.

Observation 2. A substring that is completely non-maximal in all strings in $\{S_i, | i = 1 \dots N\}$ cannot be a model, since it is contained in some supermaximal repeats.

Our models now should satisfy the two observations above and are defined using the following definitions. Note that the keywords *maximal* used in the naming of our models, have different meaning from their use with *repeat* as given in Lemma 2 above.

Definition 6. A repeat model is a repeat (i.e., non-maximal, maximal or supermaximal) in at least two of the subject strings.

Definition 7. A maximal model is a non repeated pattern occurring in at least q of the subject strings and a maximal repeat when considering its occurrences in at least q of the subject strings.

Let R and M be the sets of *repeat and maximal models* respectively. In general, we will call R and M models, the set of maximal models. Blumer[3] showed that the minimum expected length of a repeated pattern is $f \log_{|\Sigma|} n$ for $f \leq 1$. In Adebisi et al.[1], we derived that the expected length of the longest repeat is $2 \log_{|\Sigma|} n + o(\log n)$.

This implies that, the length of each model l defined above satisfied $l = \Theta(\log_{|\Sigma|} n)$. This information will be useful for the efficient extension step of each model.

3.2 Step 1: Extracting the Maximal Models

We will use the above definitions to extract models from a given set of $N \geq 2$ subject strings $\{S_i, | i = 1 \dots N\}$, each of average length n using a generalized suffix tree. To do this, we extend the algorithm for finding exact maximal and supermaximal repeats as described by Gusfield[6], using the ideas of Hui[8] for solving the CSS_x problem described in section 2. This is encapsulated in procedure FINDMODELS(N, n).

The main auxiliary structures and functions used by Hui include LeafList and lastleaf(x), where x is a leaf in GST . LeafList is the list of leaves ordered according to a post-order traversal of GST . For each leaf that belongs to a sequence in $\{S_i, | i = 1 \dots N\}$, lastleaf(x) is the last leaf which precedes x in LeafList and belongs to the same sequence. If no leaf precedes x in LeafList, Nil=lastleaf(x). We add the following preprocessing to extract our maximal models.

Let $lca(x, y)$ be the least common ancestor of x and y , where x and y are vertices in GST . An internal vertex v is a *repeat representative pair-lca* (of a sequence in $\{S_i, | i = 1 \dots N\}$, if there exists a leaf x of a sequence S_i , such that $v = lca(y = lastleaf(x), x)$ and lastleaf(y)=Nil. Let $RRPcount(v)$ denotes the number of such situations for each node v . Therefore, to verify the conditional part of Definition 6 for a node u , the problem to solve is equivalent to the calculation of its number of Non-Nil lastleaf that refers to distinct sequences in $\{S_i, | i = 1 \dots N\}$. This value for a node u , is called *distinct(u)* and is recursively calculated as $distinct(u) = \sum_{v \in subtree(u)} RRPcount(v) = RRPcount(u) + \sum_{w \in W} distinct(w)$, where W is the set of children of u .

1. **procedure** FINDMODELS(N, n)
2. Build GST
3. Create the LeafList
4. Compute the lastleaf() values
5. Compute the RRPcount() values
6. Compute the distinct() values
7. **For** each node x (in bottom-up format)
8. if(!non-maximal $_x$)
9. if($distinct \geq 2$ and $CSS_x \geq 2$)
10. Append x to R
11. else if($distinct == 0|1$ and $true == TM$)
12. Append x to M

Fig. 2. Extracting maximal models.

The property expressed in Observation 2 above, can be verified by adding a boolean variable non-maximal $_x$ to each node x . The boolean variable non-maximal $_x$ is 0, if the node is at least once maximal in a sequence and 1 if non-maximal in all strings in $\{S_i, | i = 1...N\}$. This can be done by a simple combination of the original algorithm of Gusfield and the auxiliary structure and function LeafList and lastleaf. To verify the properties from Definition 7 for a node u , distinct(u) must be zero or one and what remains is to test its maximality (henceforth TM) with respect to its occurrences in at least q strings in $\{S_i, | i = 1...N\}$. This involves collecting and scanning of the left characters of the suffixes represented by its children. Using the above facts, we can now compactly present FINDMODELS(N, n).

Lemma 4. All the models defined in Definitions 6 and 7 can be extracted from GST in $O(N \cdot n)$ expected time.

Proof. The preprocessing tasks before extracting models include Creating LeafList, computing lastleaf(), RRPcount() and distinct() values respectively. lastleaf() values are calculated via a single scan of LeafList, therefore, since the set of leaves is not more than $N \cdot n$, creating LeafList and computing lastleaf values can be done in $O(N \cdot n)$ time. Given an arbitrary pair of leaves, their lca can be computed in $O(1)$ time[16]. After initializing all $RRPcount()$ values by zero, for every leaf x , we compute $u = lca(x, y = lastleaf(x))$ and if $lastleaf(y) = Nil$, which means, it has not been counted, we increment $RRPcount(u)$ by 1. Therefore $RRPcount()$ values can be computed in $O(N \cdot n)$ time for all leaves. What then follows is the computation of all distinct() values by its recursive definition given above via a post-order traversal. This is also achieved in $O(N \cdot n)$ time.

In lines 8-12, we now extract our maximal models. This is also achievable in $O(N \cdot n)$ time because the total number of nodes and leaves in a GST is bounded by $O(N \cdot n)$ [3]. □

3.3 Step 2: Extending the Maximal Models

The algorithm we use to extend the models is similar to the algorithm used in Adebisi et al.[1] to extend seeds to find approximate repeats in one long subject string of length n . To extend a seed W , given P , the length of short substrings sought for, that repeat with at most D -differences, *i.e.* insertions, deletions, and mismatches, we split W into a left and right part W^l and W^r , where $|W^l| = \log_{|\Sigma|} n$. Next, using SAM (Sublinear Algorithm of Myers[11,1]), we find all locations of all approximate repeats $(W^l)'$ of W^l . Their right indices are stored in set H . Next, we check whether we can extend $(W^l)'$ to the right to get approximate repeats for $W = W^l W^r$. Therefore, for every index H_i in H corresponding to a word W_i^l with differences D_i , we consider the word W_i^r starting from H_i and having length $P - \log_{|\Sigma|} n$. Let E_W denote the set of possible rightward extensions of W^r to a word of length $P - \log_{|\Sigma|} n$. Note that $|E_W|$ is the number of occurrences of the seed W in string S , which is, fortunately, at most $|\Sigma|$ for supermaximal repeats. We compute the pairwise alignments of W_i^r with all elements E_j of E_W and, if the difference between any of the two substrings is at most $D'_i = D - D_i$, then the extension works and we can add $W_i = W_i^l W_i^r$ to the output list SA for the short approximate repeats. The procedure is called $\text{ADDWR}(D'_i, H_i, P)$.

To extract common motifs, the situation is different, the subject string is of size $N \cdot n$. We assume in the following exposition that $P = 2 \log_{|\Sigma|}(N \cdot n)$. Note that if $P \leq \log_{|\Sigma|}(N \cdot n)$, we just directly apply SAM and we do not need any extension by subprocedure ADDWR described below. For each model, after we ran SAM on their left part, we sort the set H , so that we can partition them into N subclasses, depending on where they occur in each subject string S_j , $j = 1 \dots N$. That is, $H_{ij} \in H$, $i = 1 \dots |H|$ and $j = 1 \dots N$. Then in subprocedure ADDWR, for every index H_{ij} in H corresponding to a word W_{ij}^l with differences D_i , we consider the words W_{ij}^r starting from H_{ij} of length $P - \log_{|\Sigma|}(N \cdot n)$. Next, we can no longer assume that $|E_W|$ is at most $|\Sigma|$, since supermaximal, maximal and non-maximal repeats are all used to construct our models. Note that the concept of maximality is still paramount in our models construction, so that $|E_W|$ in expectation, is the number of maximal substring occurrences divided by the number of internal nodes in a generalized suffix tree. Therefore $|E_W|$ is constant in expectation. The only exceptional case is when a model appears as maximal in one sequence but non-maximal in the others. In this case, its logarithmic length can be used to show that in expectation, its number of occurrences is still bounded by the number of maximal substring occurrences. Finally, we compute the pairwise alignment of W_{ij}^r with all elements E_o of E_W and, if the difference between substrings of W_{ij}^r s.t. $j = 1 \dots t$, $t \geq q$ and one element of E_o is at most $D'_i = D - D_i$, then $W_{ij} = W_{ij}^l W_{ij}^r$ are the locations of a common motif in the subject strings $\{S_j \mid j = 1 \dots t\}$. This one element of E_o is stored in the set of Common Motifs, CM .

The operation described above is encapsulated in figure 3 below.

1. **procedure** FINDSMOTIF(D, P, N, n)
2. $R, M \leftarrow$ FINDMODELS(N, n)
3. **for** $W_i, i = 1 \dots |R \cup M|$ **do**
4. $W^l, W^r \leftarrow$ SPLITW(W_i)
5. $H \leftarrow$ SAM(D^l, W^l)
6. **Sort** H
7. **for each** $H_{ij}, i = 1 \dots |H|, j = 1 \dots N$ **do**
8. $CM \leftarrow$ ADDWR(D^r, H_{ijr}, P)

Fig. 3. Extracting common motifs.

Theorem 1. *Given a set of $N \geq 2$ subject sequences $\{S_i, | i = 1 \dots N\}$, each of average length n over a finite alphabet Σ and threshold D , all common motifs, each of size $P = O(\log(N \cdot n))$ can be found in $O(N \cdot n + D(N \cdot n)^{1+pow(\varepsilon)} \log(N \cdot n))$ expected time for $D \leq 2 \log_{|\Sigma|} N \cdot n$, where $\varepsilon = D/P$, $pow(\varepsilon) = \log_{|\Sigma|}(c + 1)/(c - 1) + \varepsilon \log_{|\Sigma|} c + \varepsilon$ and $c = \varepsilon^{-1} + \sqrt{1 + \varepsilon^{-2}}$.*

Proof. The subprocedure FINDSMODELS includes the building of the generalized suffix tree and finding the models. *GST* can be built in $O(N \cdot n)$ time. Lemma 4 shows that we can also extract the maximal models in $O(N \cdot n)$ time. Therefore the expected running time of FINDMODELS(N, n) is $O(N \cdot n)$.

Note that the number of maximal models is bounded by $O(N \cdot n)$ and the operation perform in SPLITW can be done for each model in constant time. SAM running time for each model is $O(D(N \cdot n)^{pow(\varepsilon)} \log(N \cdot n))$. It is known[11] that the size of the output H of SAM for each query model is $O(N \cdot n^{pow(\varepsilon)})$. Therefore the time to sort set H is at most $O(N \cdot n^{pow(\varepsilon)} \log(N \cdot n))$, while ADDWR requires for all indices in H at most $O(N \cdot n^{pow(\varepsilon)} \log_{|\Sigma|}^2(N \cdot n))$ time. Therefore, the total running time for calling SAM and ADDWR for all models is bounded by $O(D(N \cdot n)^{1+pow(\varepsilon)} \log(N \cdot n))$, for all values of N, n , and Σ . The bound stated in the lemma then follows. \square

We show the correctness of our method using the following proposition.

Proposition 1. *It is true that the above algorithm in Fig. 3 with the above complexity solves the problem of extracting common motifs under the Levenshtein measure.*

Proof. In our construction of maximal models, we optimally ensure that all possible valid node-occurrences of common motifs are covered. To specifically establish it, we make use again of the algorithm of Sagot[17]. For simplicity, we assume that motifs of fixed length P are sought for. The basic ingredients in the validating algorithm include $CSS_x, Colors_x$ (see section 2 for their definitions) and the simultaneous SPELLING of each characters of the 'external objects', the models, over Σ^P and the recursively *grouping and validating* of its reoccurrences in each sequence using the given basic parameters P, D and q .

In the following, when we refer to repeats, we mean the reoccurrences of a pattern in the same sequence. The nodes in a generalized suffix tree can be

categorized into two: 1) The ones that represent patterns that are repeated but also occur (or not) in other sequences. 2) The ones that represent patterns that are non-repeated but also occur (or not) in other sequences. The set of *Repeat* and *maximal models* (see Definition 6 and 7) covers the first and the second group respectively. This is done in line 2.

The calculation of CSS_x is performed in line 6, while $Colors_x$ is technically performed in lines 3-5. Finally, the grouping and validating processes are performed in line 7-8. These lines also check to make sure that too many insertions, deletions and substitutions have not been accumulated. The basic reason of using 'external objects' as models in Sagot's algorithm, is because the real motif (in some literature, consensus) itself may not exist in the set of sequences, but it may be represented by one of its occurrences. This implies that the set of occurrences of a motif may share approximately a common pattern. In our algorithm above, the concept of D -neighborhood also used in SAM but in a different way, allows us to find approximate motif reoccurrences that are related via an approximately common pattern instead of an exact pattern. \square

Remark: Comparison of the running times $O(N^2nP^D|\Sigma|^D)$ of Sagot algorithm and $O(N \cdot n + D(N \cdot n)^{1+pow(\epsilon)} \log(N \cdot n))$ of our algorithm shows that our algorithm is a major improvement for moderately large P and D that normally arise in practice. Note further that for small values of D (for example, $D = 1$) as in some cases in practice, the complexity of Sagot's algorithm will therefore scale linearly and thus its complexity is better than our algorithm in this case.

4 Experimental Experience

We implemented Sagot's algorithm[17] for Hamming and also for Edit distance. The three real sequences we used here consist of sequences from B. Subtilis[7], collection of promoter regions upstream the genes SWI4, CLN3, CDC6, CDC46 and CDC47 of yeast *S. cerevisiae* that is known to contain a shared cell-cycle dependent promoter[10], obtained from NCBI database, and those extracted from *H. pylori*[18] obtained from *ftp : //ftp.tigr.org/pub/data/h_pylori*. We found minor inconsistencies in the extraction of sequences of B. Subtilis[20]. We found out that the required set of B. Subtilis contains 131 sequences of average length 100bp (instead of 99bp), for a total of 13073 (instead of 13099) nucleotides. The set of extracted sequences of yeast we used, contains 5 sequences and the largest of them is 2565bp. The total nucleotides for this set is 8639. The set (called Set C in Vanet et al.[20]) of the extracted sequences of *H. pylori* consists of non-coding regions upstream from genes coding for either a ribosomal RNA or a ribosomal protein, or from operons including ribosomal protein genes. This set consists of 26 sequences, the largest sequence has 2997bp and the total size of the set is 29324 nucleotides.

4.1 Efficiency Discussion

Let A and B denote the algorithms of Sagot[17] for Hamming and Edit distance respectively and C be our new algorithm for Edit distance. We compare here the running times and the number of common motifs returned by each algorithm¹.

Figures 4 to 7 of the appendix show that the algorithms of Sagot are clearly inefficient for moderately large values of P and D as theoretically analyzed before.

Note that algorithms A and B report much more redundant motifs than C . This is due to the fact that many of the motifs found by A and B correspond approximately to the same words in the sequences[20]. Algorithm C does not have this effect. In table 1 of the appendix, for $D \geq 4$, the numbers of resulting motifs for Hamming and Edit distances do not differ because of the lack of sufficient node-recurrences. This behavior depends on the set of sequences under consideration as we observed different behaviors for the sequences sets of *H. pylori* and Yeast. Finally for $P \geq 10$, $D = 5$ and $q = 65$ in table 2 of the appendix, the sensitivity of Edit distance compared to Hamming distance becomes obvious as no motif was found with Hamming distance (Algorithm A). We will analyze in the next section the common motifs extracted for the Edit distance. Here also for algorithm B , the number of node-reoccurrences decreases dramatically, so that the number of models returned decreased from 1126894 to 50680. These effects lead to the flatness observed in Figures 7 for the run-times of algorithms A and B . We predict also that this behavior depends on the set of sequences under consideration.

4.2 Effectiveness Discussion

We begin this section by presenting the common motifs found by algorithm A in comparison to the results of Vanet et al.[20]. We found no significant error in the common motifs reported, because all common motifs reported by them were also found with differences only in the number of times they occur. All the other common motifs found by our algorithm A correspond to variants of those reported. Vanet et al.[20] also observed this in their report. This result is given in table 3 below.

Algorithm C compares favorably with the validating algorithm B . The common motifs that played the pivoting role here, are those returned from *B. Subtilis* and set C of *H. pylori* in Vanet et al.[20]. Table 3 shows the results on *B. Subtilis*. An elaborate experimental results on the sequences of *H. pylori* can be found in table 5.6 of [2]. There, we ran algs. B and C to find the pivoter motifs using $D = 1$, $q = 13$, and $P = 6, 7, 8, 9$. Note that, if the common motifs (more than 3) returned do not exactly match the pivoting common motifs, we represent them with one of their variant. Note that G,C-GTAAAA means GGTA AAA, CG-TAAAA, TATA-TT,AT means also TATATT, TATAAT and blank entry means

¹ The programs were run on a SUN-sparc computer under Solaris 2.5.1 with a 366 MHz-Processor and 360Mbytes of main memory

the pivoter pattern or its variants are not found. We observe that in table 3 (for Algs. B and C), a slightly smaller q provides the pivoter motifs that both algorithms cannot find at $q = 65$. This is not the case in table 5.6[2] as both algorithms find either exact or variants of the pivoter motifs. Therefore we concluded that the differences (although mostly small) in the number of the pivoter motifs returned by the two algorithms, when both returned them exactly, is simply due to various optimizations performed in SAM with the ones, we did in deriving our models in algorithm C. This does not affect the effectiveness of algorithm C[2].

Algorithm A failed to predict the corresponding motif in yeast as we can see in table 4. Vanet et al.[20] recognized this problem as they stated that suggestions from biologists concerning possible extension of their algorithm application to eukaryotes are being sought. Algorithm B ran for more than 16 hours. We stopped its execution due to lack of storage to store all the motifs found. Algorithm C got two corresponding motifs within a reasonable time. We observed that its predicted motifs distance themselves away from the published motif by a Hamming distance of 4.

Lastly, in table 5 below, we analyze and present some common motifs extracted in the case, where the Hamming distance failed. Our example here is for $D = 3$, $P = 10$ and $q = 65$ and they can be found around the TATA-box and known patterns like (TATAAT and TTGACT)[7] and (TAAAT and GAAAAA)[12].

We use a combination of the data shuffling methods[9] and the χ^2 -test with one degree of freedom to evaluate the statistical significance of the common motifs found. The statistical significance of each motif is evaluated using a contingency table[14]. The first table listed the number of occurrences of each motif per sequence without shuffling the sequences. This is the observed hypothesis. The second table corresponds to what is expected when the sequences are shuffled. This is the expected hypothesis otherwise known as the null hypothesis. For each motif, we are now to find the probability of getting this motif, given the null hypothesis. To obtain the χ^2 values in table 5, we performed a thousand random shufflings preserving both the mono- and dinucleotide frequency distributions of the original sequences.

Using the Hamming distance, Vanet et al.[20] obtained statistical significance common motifs whose length are greater than six, with at most one substitution and a quorum of $q = 65$ from B. Subtilis sequences. The length of the motifs they obtained is actually either six or seven. This confirm our results in table 2 that algorithm A failed to predict common motifs for $P \geq 10$. The χ^2 values of the motifs that Vanet et al.[20] obtained for B. Subtilis is in the range of 11-31. Note that the χ^2 values above 10.8 means that the corresponding motifs that have such χ^2 values, have the probability of happening by chance below 10^{-3} .

The χ^2 values of the common motifs for B. Subtilis under the Edit distance that we show in table 5 is in the range of 50-400. These are higher than the χ^2 values of Vanet et al.[20] on B. Subtilis sequences for Hamming distance. These higher values are expected, since the number of occurrences of statistical

significant motifs per sequence is higher, under the Edit distance than under the Hamming distance.

5 Conclusion

We have implemented and tested the more involved algorithm of Sagot[17] for Edit distance. Furthermore, we have also presented an algorithm that is efficient both in theory and in practice for finding moderately long motifs that distance itself away from its reoccurrences under the Edit distance metric. Its effectiveness compares favorably with the algorithm of Sagot[17]. Our algorithm reports also less false positive motifs. Additionally, we have demonstrated the relevance (in practice) of the Edit distance metric in the extraction of motifs. The sensitivity of Edit distance and the significance of its common motifs, presented in Table 5, that eluded the Hamming distance gives evidence that some important motifs may have eluded existing motifs finder algorithms that are mostly based on Hamming distance.

Our results are important and significant but left some important problems open. The results of table 5 below are preliminary. More random shufflings in thousands need to be performed on sequences of organisms like the Human and the ground Squirrel, e.t.c, that may have common motifs caused by substitutions, insertions, and deletions[9]. Another statistic, like the Z -score should be used to certified the results obtained. The motifs, we consider here are "consensus sequences" that must occur at specific positions. The main limitation of consensus sequences is that they do not allow for different positions to be conserved to different degrees, or for some differences to the consensus to be penalized more than others. An interesting work will be to introduce a better model that have some contribution from each base at each position, and the sum of all the contributions to be above some threshold. This model is nicely handled by a weight matrix representation, called Position Weight Matrices (PWM)[4]. Finally, it will be interesting to apply our ideas here to the extraction of structured motifs[12,20].

Acknowledgement

We thank J. D. Helmann for providing pointer to the sequences of *B. subtilis*. We also thank Tao Jiang and Kay Nieselt-Struwe for some useful discussions.

References

1. E. F. Adebiyi, T. Jiang, and M. Kaufmann. *An efficient algorithm for finding short approximate non-tandem repeats (Extended Abstract)*. Bioinformatics, 17(1):S5-S13, 2001.
2. E. F. Adebiyi. *Pattern Discovery in Biology and Strings Sorting: Theory and Experimentation*. Ph. D Thesis, 2002.

3. A. Blumer and A. Ehrenfeucht and others. *Average size of suffix trees and DAWGS*. Discrete Applied Mathematics, 24, 37-45, 1989.
4. J.-M. Claverie and S. Audic. *The Statistical significance of nucleotide position-weight matrix matches*. Computer Applications in Biosciences 12(5), 431-439, 1996.
5. M. Crochemore and M.-F. Sagot. *Motifs in sequences: localization and extraction*. In Handbook of Computational Chemistry, Crabbe, Drew, Konopka, eds., Marcel Dekker, Inc., 2001. To appear.
6. D. Gusfield. *Algorithms on strings, trees and sequences*. Cambridge University Press, New York, 1997.
7. J. D. Helmann. *Compilation and analysis of Bacillus Subtilis σ^A -dependent promoter sequences: evidence for extended contact between RNA polymerase and upstream promoter DNA.*, Nucleic Acids Research, 23(13): 2351-2360, 1995.
8. L. C. K. Hui. *Color set size problem with applications to string matching*. In CPM Proceeding, vol. 644 of LNCS, 230-243, 1992.
9. S. Karlin, F. Ost, and B. E. Blaisdell. *Patterns in DNA and amino acid sequences and their statistical significance*. In M. S. Waterman, editor, Mathematical Methods for DNA Sequences, 133-158, 1989.
10. C. J. McNerny, J. F. Patridge, G. E. Mikesell, D. P. Creemer, and L. L. Breen. *A novel Mcm1-dependent element in the SWI4, CLN3, CDC6, CDC46, and CDC47 promoters activates M/G₁-specific transcription*. Genes and Development, 11: 1277-1288, 1997.
11. E. Myers. *A sub-linear algorithm for approximate keyword matching*. Algorithmica 12, 4-5, 345-374, 1994.
12. L. Marsan and M. F. Sagot. *Extracting structured motifs using a suffix tree algorithms and application to promoter consensus identification*. RECOMB 2000.
13. P. Pevzner and S.-H. Sze. *Combinatorial approaches to finding subtle signals in DNA sequences*. ISMB, 269-278, 2000.
14. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes*. In The Art of Scientific Computing, Cambridge University Press, Cambridge.
15. E. Rocke and M. Tompa. *An algorithm for finding novel gaped motifs in DNA sequences*. RECOMB, 228-233, 1998.
16. B. Schieber and U. Vishkin. *On Finding Lowest Common Ancestors: Simplification and Parallelization*. SIAM Journal on Computing, 17:1253-1262, 1988.
17. M.-F. Sagot. *Spelling approximate repeated or common motifs using a suffix tree*. LNCS 1380: 111-127, 1998.
18. J. F. Tomb et al. *The complete genome sequence of the gastric pathogen Helicobacter pylori*. Nature, 388, 539-547, 1997.
19. E. Ukkonen. *Approximate string matching over suffix trees*. LNCS 684: 228-242, 1993.
20. A. Vanet, L. Marsan, A. Labigne and M.-F. Sagot. *Inferring regulatory elements from a whole genome. an analysis of Helicobacter pylori σ^{80} family of promoter signals*. J. Mol. Biol., 297, 335-353, 2000.

Appendix: Figures and Tables

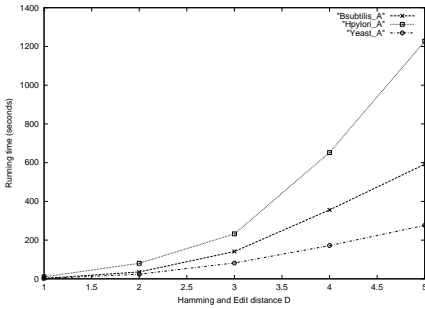


Fig. 4. The running times of algorithm A for $P = 7, 8, 8, q = 65, 13, 3$ and various values of D on B. Subtilis, H. Pylori and Yeast sequences.

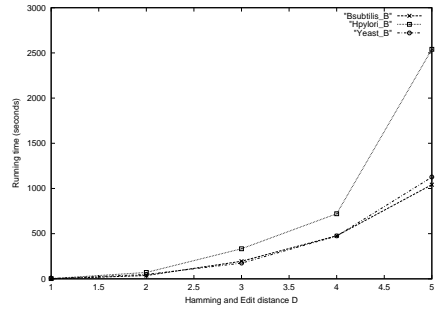


Fig. 5. The running times of algorithm B for $P = 7, 8, 8, q = 65, 13, 3$ and various values of D on B. Subtilis, H. Pylori and Yeast sequences.

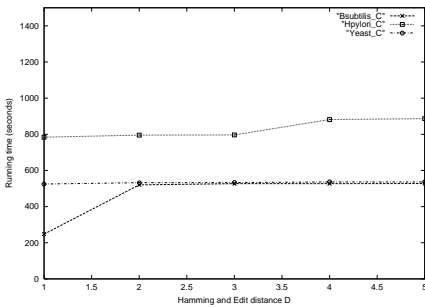


Fig. 6. The running times of algorithm C for $P = 7, 8, 8, q = 65, 13, 3$ and various values of D on B. Subtilis, H. Pylori and Yeast sequences.

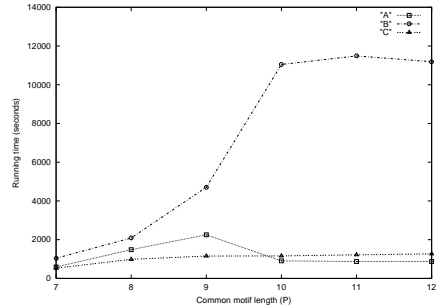


Fig. 7. The running times of algorithms A, B, C for $D = 5, q = 65$ and various values of P on B. Subtilis sequences.

Table 1. The number of common motifs returned by each algorithm for $P=7, q=65$ and various values of D on B. Subtilis sequences.

D/Algorithms	A	B	C
1	129	41	34
2	6895	12225	2520
3	15493	16384	2572
4	16384	16384	2572
5	16384	16384	2572

Table 2. The number of common motifs returned by each algorithm for $D=5, q=65$ and various values of P on B. Subtilis sequences.

P/Algorithms	A	B	C
7	16384	16384	2572
8	65283	65536	4945
9	152941	262144	6618
10	0	1008661	7425
11	0	1126894	7687
12	0	50680	7626

Table 3. The common motifs returned for $D = 1$, $q = 65$ and $P = 6, 7$ from *B. Subtilis*. Columns 3 and 4 give the results for Hamming distance, while columns 5 and 6 give the results for Edit distance.

	B. Subtilis	Alg. A	Alg.[20]	Alg. B	Alg. C
Family 1	TATAATA	75	94	(TATAAAA)67	102
	GTATAAT	(GTAAAAA) 68	74	(G,C-GTAAAA) 69,72	82
	TGTTATA	67	66		86
	ATAATAT	73	82		95
	ACAATA	81	108	77	
	TAAAATA	79	95	72	(TAAAATG) 74
	TATTATA	70	76	(TATAAAA) 67	85
	GATATA	79	98	81	(GAAATA) 73
	TATAGT	77	95	(TATA-TT,AT,AA)80,81,81	(TATACA) 76
Family 2	TTTTACA	70	76		84
	GTGACA	(GTGAAA) 72	68		
	GTTGAC	72	66	(CGTTGA) 78	
	TTTACAA	73	75		(TTTACAT) 67
Classical Motifs[7]	TATAAT			81	76
	TTGACA				

Table 4. Performance of algorithms *A*, *B*, and *C* on an eukaryotic promoter sequences, Yeast for $P = 16$.

Algorithm	D	Running time	Published motif is TTtCCcnnntnaGGAAA[10]
		for last D	Correspond Motif(s) predicted
A	2-5	111.06 sec	-
B	2-8	>16.7 hrs	-
C	4	1440.48 sec	TTTACCACCTAGAAGA, TTCAAAAATGAGGAAA

Table 5. Some common motifs found in *B. Subtilis* sequences for Edit distance ($P = 10$, $D = 3$, and $q = 65$). The second column listed the number of distinct sequences, the patterns occur. The third column listed the same for the shuffled versions of *B. Subtilis* sequences, average over 1000 simulations. The last column gives each pattern χ^2 value. All motifs have the probability of happening by chance below 10^{-3} .

Common Motifs	# of distinct sequences each Common Motifs occurs:		
	Observed	Expected	χ^2
AAGTATAATG	72	27	218.33
ACGGTATATA	65	10	54.42
CCTATATTTA	65	13	54.05
GATTATAATG	65	15	170.41
GGGTGCTATA	65	15	44.75
CCGTGAAAAA	78	23	332.21
AGTTATAATG	72	27	218.33
ACGTTGAATA	66	18	65.97
CCGTAAAAATG	66	18	123.11
CCGTGAAAAA	78	23	332.21
TTATATAATG	66	12	47.45