A First Approach to Finding Common Motifs With Gaps

Costas S. Iliopoulos[1*], James McHugh[2], Pierre Peterlongo[3], Nadia Pisanti[4†], Wojciech Rytter[5] and Marie-France Sagot[6,1‡]

[1] Dept. Computer Science, King's College London, London WC2R 2LS, England, and School of Computing, Curtin University of Technology, GPO Box 1987 U, WA.
e-mail: csi@dcs.kcl.ac.uk http://www.dcs.kcl.ac.uk/staff/csi

[2] New Jersey Institute of Technology College of Computing Sciences 323 M.L.King Blvd. University Heights Newark, NJ 07102-1982, USA
e-mail: mchugh@oak.njit.edu http://www.cs.njit.edu/~mchugh

[3] Institut Gaspard-Monge, Universite de Marne-la-Valle, Cite Descartes, Champs sur Marne, 77454 Marne-la-Valle CEDEX 2, France
e-mail: pierre.peterlongo@univ-mlv.fr

[4] Department of Computer Science, University of Pisa, Italy
e-mail: pisanti@di.unipi.it http://www.di.unipi.it/~pisanti

[5] New Jersey Institute of Technology College of Computing Sciences 323 M.L.King Blvd. University Heights Newark, NJ 07102-1982, USA
e-mail: rytter@oak.njit.edu http://www.cs.njit.edu/~rytter

[6] Inria Rhône-Alpes, UMR 5558 Biométrie et Biologie Évolutive Université Claude Bernard Lyon 1. 43, Bd du 11 novembre 1918 69622 Villeurbanne cedex France
e-mail: Marie-France.Sagot@inria.fr
http://www.inrialpes.fr/helix/people/sagot

Abstract. We present three linear algorithms for as many formulations of the problem of finding motifs with gaps. The three versions of the problem are distinct in that they assume different constraints on the size of the gaps. The outline of the algorithm is always the same, although this is adapted each time to the specific problem, while maintaining a linear time complexity with respect to the input size. The approach we suggest is based on a re-writing of the text that uses a new alphabet made of labels representing words of the original input text. The computational complexity of the algorithm allows to use it also to find long motifs. The algorithm is in fact general enough that it could be applied to several variants of the problem other those suggested in this paper.

*The work was partially supported by a NATO grant PST.CLG.977017, a Marie Curie Fellowship, a Royal Society and a Wellcome Foundation grant.

†Partially supported by Programme Bioinformatique inter EPST and the French Ministry of Research through the ACI NIM.

‡Partially supported by Programme Bioinformatique inter EPST and the French Ministry of Research through the ACI NIM, and by Royal Society, Nato and Wellcome Foundation Grants.

1 Introduction

The inference of common motifs is an interesting problem in a variety of text algorithm applications [1, 2, 4]. Given several input strings, the goal is to find words that are shared by all (or by some) of them. In many applications, such as for instance biology, the requirement should be a certain degree of *similarity*, rather than identity, between a motif and its occurrences in the different input strings. This is what makes the problem computationally difficult and algorithmically challenging. Such similarity can be expressed in various ways. Among the most classical, in one case similarity is modelled by means of a limited edit (or Hamming) distance between a motif and its occurrences while in another don't care symbols are allowed in a motif. Our approach falls under this latter category. We are therefore interested in motifs with don't cares. A don't care is a special symbol that matches any letter of the alphabet. We further focus our attention on statements of the problem where don't cares appear concentrated in distinct sets of contiguous positions, that is, we are interested in finding common motifs with *gaps*. In computational biology, this particular problem may have applications to the inference of co-regulated genes, that is, to the detection of common binding sites whose structure often consists of motifs separated by gaps of approximatively known size. The hypothesis is that genes whose promoter regions share a binding site are co-regulated. Common motifs located in the regions upstream of genes are good candidates to be such common binding sites that could then be experimentally tested. Figure 1 shows an example of gapped motif.

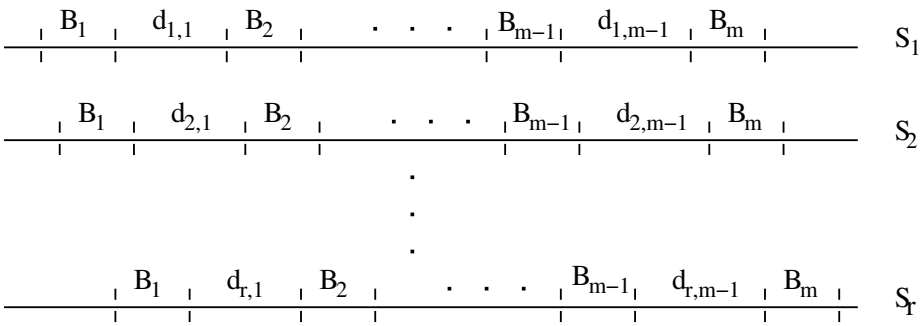

Figure 1: An example of a motif with gaps that occurs in every string, where by $d_{i,j}$ we mean a gap of size $d_{i,j}$.

Section 2 formally introduces the general problem and describes the data structure used in the algorithms presented in this paper. Section 3 defines the simplest formulation of the problem, that is, when all gaps have the same fixed size, and shows a linear solution for this problem. This will serve as a starting point for the solution to two variants of the problems where the sizes of the gaps are more flexible. Section 4 presents the first variant where each gap inside a motif has variable size but for each occurrence the total sum of such sizes is upper bounded by a fixed value. A solution for this problem is then provided that does not increase the complexity of the first algorithm. Finally, Section 5 addresses, and solves by keeping the same time complexity, another variant of the problem where each gap is of variable size but each such size is upper bounded by a fixed value.

2 Preliminaries

A *string* is a sequence of zero or more symbols from an alphabet Σ. A string s of length n is represented by $s_1 s_2 \cdots s_n$, where $s_i \in \Sigma$ for $1 \leq i \leq n$. A string w is a *substring*, or *word* of s if $s = uwv$ for $u, v \in \Sigma^*$; in this case we also say that the string w occurs at position $|u| + 1$ of the string s. We denote by $s[i..j]$ with $1 \leq i \leq j \leq n$ the word $s_i s_{i+1} \cdots s_j$ of s. A string w is a *prefix* of s if $s = wu$ for $u \in \Sigma^*$. Similarly, w is a *suffix* of s if $s = uw$ for $u \in \Sigma^*$. The don't care symbol that matches any other symbol of Σ is denoted by $*$.

The general problem addressed in this paper can be formalized in the following way.

We are given a set of r input strings $\{S_1, S_2, \ldots, S_r\}$ and we want to find gapped motifs $B = B_1 *^{d_1} B_2 *^{d_2} \cdots *^{d_{m-1}} B_m$ occurring in each of them. An example in shown in Fig. 1.

We consider three variants of the problem that basically differ in the way the size of the gaps are constrained. Assuming that the number m of motifs as well as the gap sizes (bounded by input parameters) are constants, we present linear algorithms for all three variants.

The proposed algorithms make use of a *generalised* and *truncated* suffix tree. A generalised suffix tree [5] is a suffix tree for a set of more than one input string where the information of which string each indexed suffix belongs to is stored in the nodes of the tree. A truncated suffix tree [6] is a suffix tree that stores only suffixes up to a certain length l. In practice, it is a suffix tree pruned at level l.

3 Finding motifs with fixed gaps

We start by considering the most constrained version of the problem, that is when the size of the gaps is always the same. More flexible formulations of the problem will be addressed in the next sections.

Formally, given a set of strings $\{S_1, S_2, \ldots, S_r\}$ and integers k, m, d, the problem consists in finding words $B_1, B_2, \ldots, B_m$ such that:

1. $|B_1| = |B_2| = \ldots = |B_m| = k$;
2. $B_1 *^d B_2 *^d \cdots *^d B_m$ occurs in S_i for $i = 1..r$.

Observe that technically we can allow d to be zero. This possibility becomes more interesting when dealing with gaps of variable size. The strings $B_1 *^d B_2 *^d \cdots *^d B_m$ are said to form *a gapped motif*; in the literature the sequence of words $B_1, B_2, \ldots, B_m$ is also called *a chain* and each word B_i a *block* of a gapped motif. The size of the blocks B_j's is also assumed to be fixed and the same for all blocks.

We now give an algorithm that solves the problem stated above.

Step 1 Build the generalised suffix tree, truncated at depth k, for the input strings $S_1, S_2, \ldots, S_r$. This data structure indexes all words of length k (which we call the *"k-words"*) of the set of input strings. We label each "k-word" by the number of the corresponding leaf in the suffix tree (the number of distinct such k-words is at most linear in the input size). The k-words are thus labelled in lexicographic order.

STEP 2 Build a new set of strings $\{\tilde{S}_1, \ldots \tilde{S}_r\}$ as follows. The new string $\tilde{S}_i$ is obtained from S_i by replacing each symbol with the label of the k-word that starts there. In other words, $\tilde{S}_i[j]$ becomes the label assigned to the k-word $S_i[j..j+k-1]$.
STEP 3 Search for all possible chains of m blocks by doing $m-1$ skips of length $d+k$ in $\tilde{S}_i$ (the motifs have now length $|B_1 *^d B_2 *^d \ldots *^d B_m| = m(d+k)-d$). The *chains* sought are the following:

$$C_i[j] = \tilde{S}_i[j]\tilde{S}_i[j+d+k]\tilde{S}_i[j+2(d+k)]\ldots\tilde{S}_i[j+(m-1)(d+k)],$$
$$\forall j \in \{1..n-m(d+k)+d\}$$

STEP 4 Let $\hat{S}_i$ be the set of chains $C_i[j],\ \forall j \in \{1..n-m(d+k)+d\}$. Obtain the sets $\hat{S}_i$ for all i.
STEP 5 Build the generalised suffix tree for the set of sets of chains $\{\hat{S}_1, \ldots \hat{S}_r\}$. This tree has depth m. The leaves are labelled by the set of strings S_i each chain in $\{\hat{S}_1, \ldots \hat{S}_r\}$ is derived from.
STEP 6 The motifs sought correspond to the leaves of this second generalised suffix tree with labels covering all strings.

Observe that in general the $\hat{S}_i$'s are multisets and not sets (some of the chains may appear repeated). The redundancy may be detected at Step 4 (by not including in a set $\hat{S}_i$ a chain that is already there), or left to be detected at Step 5.

	1	2	3	4	5	6	7	8	9	10	11	12
$S_1 =$	A	C	A	A	A	A	C	A	C	A	A	A
$S_2 =$	A	C	A	C	C	A	A	C	C	A	C	A
$S_3 =$	C	A	C	A	A	A	C	C	A	C	C	A

Figure 2: An example of input

As an example, consider the set of strings of Fig. 2 and the input parameters $k = 2, d = 1$ and $m = 3$. We want to find gapped motifs of the form $b_{1,1}b_{1,2}*b_{2,1}b_{2,2}*b_{3,1}b_{3,2}$ that are common to all three strings. We first construct the truncated suffix tree as shown in Fig. 3.

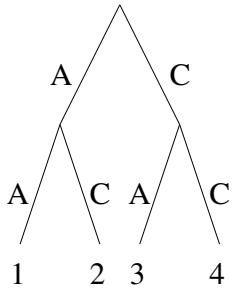

Figure 3: Truncated suffix tree for the strings of Fig. 2 and with $k = 2$

We then compute $\tilde{S}_1, \tilde{S}_2, \tilde{S}_3$ by re-writing the S_i's using the labels of the leaves. From $\tilde{S}_1, \tilde{S}_2, \tilde{S}_3$, we obtain the set of chains $\hat{S}_1, \hat{S}_2, \hat{S}_3$. The result is shown in Fig. 4

Finally, we construct the generalised suffix tree at depth $m = 3$ of the three sets of chains $\hat{S}_1, \hat{S}_2, \hat{S}_3$. The result is shown in Fig. 5. The leaf circled spells a chain that

	1	2	3	4	5	6	7	8	9	10	11	12
$\tilde{S}_1 =$	2	3	1	1	1	2	3	2	3	1	1	–
$\tilde{S}_2 =$	2	3	2	4	3	1	2	4	3	2	3	–
$\tilde{S}_3 =$	3	2	3	1	1	2	4	3	2	4	3	–

$\hat{S}_1 =$	2	1	3,	3	1	2,	1	2	3,	1	3	1,	1	2	1
$\hat{S}_2 =$	2	4	2,	3	3	4,	2	1	3,	4	2	2,	3	4	3
$\hat{S}_3 =$	3	1	4,	2	1	3,	3	2	2,	1	4	4,	1	3	3

Figure 4: Strings of Fig. 2 after steps 3 ($\tilde{S}_i$) and 4 ($\hat{S}_i$) of the algorithm

belongs to each one of the sets $\hat{S}_i$. Hence, it corresponds to a gapped motif with the required structure that occurs in every input string. This is the (only) output of our example, corresponding to the gapped motif $AC*AA*CA$ occurring in S_1 at position 1, in S_2 at position 3, and in S_3 at position 2 as shown in Fig. 6.

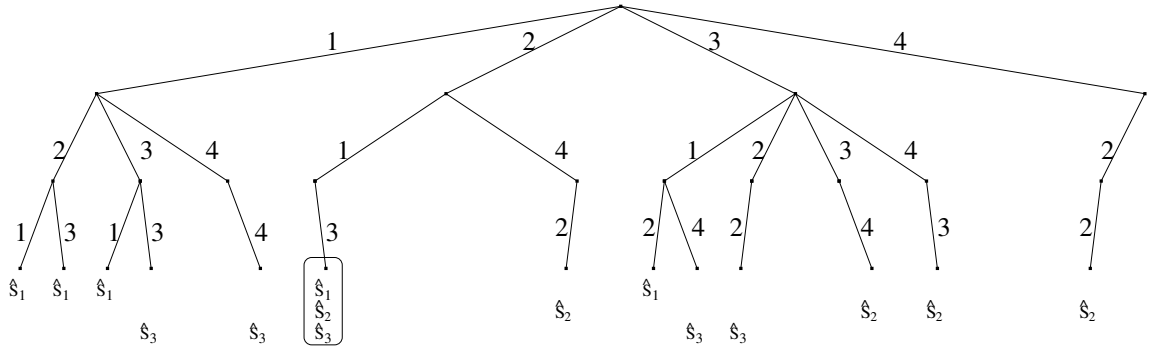

Figure 5: The generalised truncated suffix tree for $\hat{S}_1, \hat{S}_2, \hat{S}_3$. The circled leaf presents a common motif occurring in the three strings

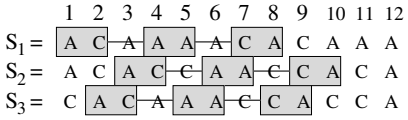

Figure 6: A motif with fixed gaps with $k = 2, m = 3$ and $D = 1$

The correctness of the algorithm is straightforward. The string $C_i[j]$ given as output in Step 6 is an encoding of a gapped motif that satisfies the input parameters. Furthermore, the construction of the generalised suffix tree of Step 5 provides the evidence that $C_i[j]$ is common to all the strings of the input set $\{S_1, S_2, \ldots, S_r\}$.

Let us now discuss the complexity of the algorithm. The construction of the generalised suffix tree at Step 1 requires $O(nr \log |\Sigma|)$ time, that is $O(nr)$ assuming $|\Sigma|$ is a constant and $n = |S_i|$ for all S_i's. The transformation of the strings S into $\tilde{S}$ requires linear $O(nr)$ time. In Step 3, we construct $n - m(d+k) + d$ chains of length $m(d+k)-d$, thus Step 3 is bounded by $O(rn)$ assuming that m, k and d are constants. Since at Step 5 the alphabet has size $O(rn)$, we resort to the suffix tree construction

of [3] whose time complexity is independent of the alphabet's size. Therefore, the construction of the generalised suffix tree of Step 5 requires $O(rn)$ time, and Step 6 is also linear. Thus the overall complexity of the algorithm is $O(rn)$.

4 Finding motifs with bounded sum of variable gaps

We now consider the problem of finding motifs with gaps of variable sizes, but whose sum is upper bounded by a user defined parameter. Again, the gapped motif must occur at least once in each one of the input strings. Given a set of strings $\{S_1, S_2, \ldots, S_r\}$, and given integers k, m, d, D, the problem is then to find strings $B_1, B_2, \ldots, B_m$ such that:

1. $|B_1| = |B_2| = \ldots = |B_m| = k$;
2. $B_1 *^{d_{i,1}} B_2 *^{d_{i,2}} \ldots *^{d_{i,m-1}} B_m$ occurs in S_i for $i = 1..r$;
3. $\sum_{j=1}^{m} d_{i,j} \leq D$ for $i = 1..r$;
4. $0 \leq d_{i,j}$.

$\hat{S}_1$ = 2 1 3, 3 1 2, 1 2 3, 1 3 1, 1 2 1, ($d_{1,1}$=1, $d_{1,2}$=1)
2 1 2, 3 2 3, 1 3 1, 1 2 1, ($d_{1,1}$=2, $d_{1,2}$=1)
2 1 2, 3 1 3, 1 2 1, 1 3 1, ... ($d_{1,1}$=1, $d_{1,2}$=2)
$\hat{S}_2$ = 2 4 2, 3 3 4, 2 1 3, 4 2 2, 3 4 3, ($d_{2,1}$=1, $d_{2,2}$=1)
2 3 4, 3 1 3, 2 2 2, 4 4 3, ($d_{2,1}$=2, $d_{2,2}$=1)
2 4 4, 3 3 3, 2 1 2, 4 2 3, ... ($d_{2,1}$=1, $d_{2,2}$=2)
$\hat{S}_3$ = 3 1 4, 2 1 3, 3 2 2, 1 4 4, 1 3 3, ($d_{3,1}$=1, $d_{3,2}$=1)
3 1 3, 2 2 2, 3 4 4, 1 3 3, ($d_{3,1}$=2, $d_{3,2}$=1)
3 1 3, 2 1 2, 3 2 4, 1 4 3, ... ($d_{3,1}$=1, $d_{3,2}$=2)

Figure 7: The strings $\{\hat{S}_1, \ldots \hat{S}_r\}$ after step 4. The end of each line shows the gap lengths applied to obtain the values (omitted if equal to zero).

What follows is a solution to the problem just stated.

Step 1 As was done in the first step of the previous algorithm, build the truncated generalised suffix tree for the input set of strings, and label each "k-word" with the number of the corresponding leaf in the suffix tree.

Step 2 Build the strings $\{\tilde{S}_1, \ldots, \tilde{S}_r\}$.

Step 3 In order to compute the different chains, we now use a window of width $D + km$. Let $W_{i,j}$ be the window appearing in string i at position j, and let $C_i[j]$ be the set of all non-overlapping words of length m that occur in $W_{i,j}$:

$$C_i[j] = \Big\{ \tilde{S}_i[p_1], \tilde{S}_i[p_2], \ldots, \tilde{S}_i[p_m] \;\; | \;\; p_1 = j, \; \forall \;\; l > 1, k \leq p_l - p_{l-1} < k + D \text{ and } p_m - p_1 \leq D + (m-1)k \text{ and } p_m < n - k \Big\}$$

Without the condition $(p_m - p_1 \leq D + (m-1)k)$, $C_i[j]$ would contain D^{m-1} elements. Thus there are at most D^{m-1} elements in $C_{i,j}$.

Step 4 Obtain the sets of chains $\hat{S}_i$.

Step 5 Construct the generalised suffix tree for the set of sets of chains $\{\hat{S}_1, \ldots, \hat{S}_r\}$.

Step 6 The motifs sought correspond as before to the leaves of this second generalised suffix tree with labels covering all strings.

Consider the same input strings as in the previous example (see Fig. 2), the same values for k and m, and $D = 3$. The $\tilde{S}_i$ strings are thus the same as before, while the $\hat{S}_i$ strings for motifs with bounded sum of variable gaps are shown in Fig. 7.

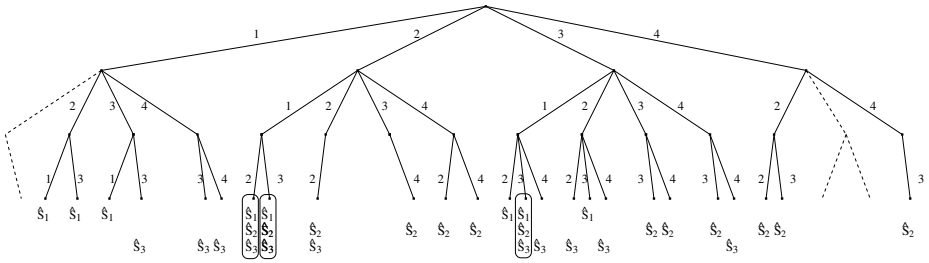

Figure 8: Part of the generalised truncated suffix tree for $\hat{S}_1, \hat{S}_2, \hat{S}_3$. Each circled leaf represents a common motif occurring in the three strings. Dashed paths indicate simply that all the tree is not drawn.

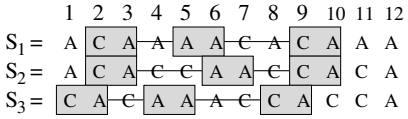

Figure 9: A gapped motif found for the bounded sum of variable gaps problem with $k = 2, m = 3$ and $D = 3$

Part of the generalised suffix tree for the set of chains $\hat{S}_1, \hat{S}_2, \hat{S}_3$, is shown in Fig. 8.

In this tree, three leaves present motifs common to all the strings. For instance, the one given by the path 313 corresponds to the gapped motif $CA *^{d_{i,1}} AA *^{d_{i,2}} CA$ with $0 \leq d_{i,j}$ and $\sum_{j=1}^{m} d_{i,j} \leq D$, $\forall i = 1..r$ which occurs in all the strings as shown in Figure 9.

The correctness of the algorithm is straightforward. The output string $C_i[j]$ in Step 6 is an encoding of a motif with bounded sum of gaps according to the input parameters that is common to all the strings $\{S_1, S_2, \ldots, S_r\}$.

If one assumes that m and D are constants, the construction of the generalised suffix tree at Step 1 requires $O(nr \log |\Sigma|)$ time, where $n = |S_i|$. The transformation of the strings S into the set of chains $\tilde{S}$ requires linear $O(nr)$ time.

For each position i in a string, the number of motifs that may have an occurrence starting at i is at most D^{m-1}. Thus, if one considers all positions in the strings, there are at most $r \cdot n \cdot D^{m-1}$ motifs. Hence, the overall cost of Step 3 is bounded by $O(rn)$ assuming that m and D are constants.

Step 5 requires $O(rn)$ time, and Step 6 is linear. The overall complexity of the algorithm is therefore again $O(rn)$, and hence linear in the input size.

5 Finding motifs with variable gaps of bounded size

In this section, we address a third variant of the problem. We have again that the size of the gaps is variable, but this time we set an upper bound on the size of each individual gap.

Given a set of strings $\{S_1, S_2, \ldots, S_r\}$ and given integers k, m, D, the problem consists now in finding strings $B_1, B_2, \ldots, B_m$ such that:

1. $|B_1| = |B_2| = \ldots = |B_m| = k$
2. $B_1 *^{d_{i,1}} B_2 *^{d_{i,2}} \ldots *^{d_{i,m-1}} B_m$ occurs in S_i for $i = 1..r$;
3. $1 \leq d_{i,j} \leq D$ for $i = 1..r, j = 1..m$.

Notice that putting an upper bound on the size of each gap implies also putting a bound on the size of their sum, as is the case of the previous variant of the problem.

$\hat{S}_1$ =
2 2 3, 3 3 1, 1 2 1, ($d_{1,1}$=3, $d_{1,2}$=1)
2 1 3, 3 2 1, 1 3 1, ($d_{1,1}$=2, $d_{1,2}$=2)
2 1 1, 3 2 1, ... ($d_{1,1}$=2, $d_{1,2}$=3)

$\hat{S}_2$ =
2 1 3, 3 2 2, 2 4 3, ($d_{2,1}$=3, $d_{2,2}$=1)
2 3 3, 3 1 2, 2 2 3, ($d_{2,1}$=2, $d_{2,2}$=2)
2 3 2, 3 1 3, ... ($d_{2,1}$=2, $d_{2,2}$=3)

$\hat{S}_3$ =
3 2 2, 2 4 4, 3 3 3, ($d_{3,1}$=3, $d_{3,2}$=1)
3 1 2, 2 2 4, 3 4 3, ($d_{3,1}$=2, $d_{3,2}$=2)
3 1 4, 2 2 3, ... ($d_{3,1}$=2, $d_{3,2}$=3)

Figure 10: Part of the strings $\hat{S}_i$ after step 4. The end of each line shows the gap lengths applied to obtain the values.

What follows is an algorithm that solves this last variant of the problem.

Step 1 Compute the truncated generalised suffix tree for the k-words of the set of input strings, and as before label each "k-word" with the number of the corresponding leaf in the suffix tree.

Step 2 Construct the strings $\{\tilde{S}_1, \ldots, \tilde{S}_r\}$ as in the previous algorithms.

Step 3 Let $C_i[j]$ be the set of all the possible chains $\tilde{S}_i[p_1], \tilde{S}_i[p_2], \ldots, \tilde{S}_i[p_m]$ in the string $\tilde{S}_i$ such that $p_1 = j$ and $\forall \;\; j > 1, k \geq p_j - p_{j-1} < k + D$.

Formally,

$$C_i[j] = \Big\{ \tilde{S}_i[p_1], \tilde{S}_i[p_2], \ldots, \tilde{S}_i[p_m] \;\; | \;\; p_1 = j \\ \text{and } \forall \;\; l > 1, k \leq p_l - p_{l-1} < k + D \text{ and } p_m < n - k \Big\}.$$

Notice that $C_i[j]$ contains at most D^{m-1} elements.

Step 4 Obtain the set of chains $\hat{S}_i$.

Step 5 As for the first algorithm, build the generalised suffix tree for $\{\hat{S}_1, \ldots \hat{S}_r\}$.

step 6 The motifs sought correspond as before to the leaves of this second generalised suffix tree with labels covering all strings.

For example, with the strings shown in Fig. 2, the same values for k and m (i.e., $k = 2, m = 3$), and $D = 3$, part of the $\hat{S}_i$ strings obtained for this last variant of the problem is shown in Fig. 10.

Part of the generalised suffix tree for the set of chains $\hat{S}_1, \hat{S}_2, \hat{S}_3$, is shown in Fig. 11.

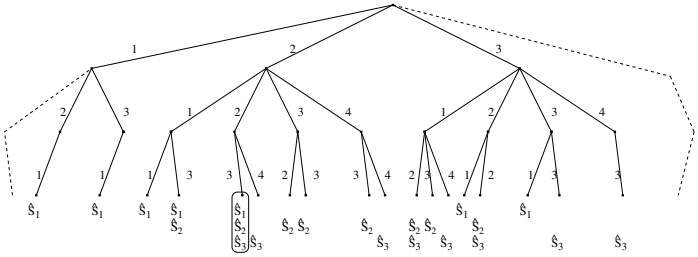

Figure 11: A part of the generalised truncated suffix tree for $\hat{S}_1, \hat{S}_2, \hat{S}_3$. The circled leaf presents a common motif occurring in the three strings. The dashed paths indicate simply that all the tree is not drawn.

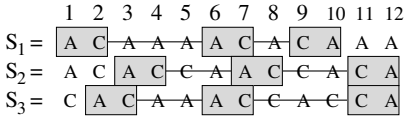

Figure 12: A gapped motif found for the bounded variable gaps problem with $k = 2, m = 3$ and $D = 3$

In this tree, one of the leaves corresponds to a motif common to all the strings. It denotes the path 223 corresponding to the gapped motif $AC *^{d_{i,1}} AC *^{d_{i,2}} CA$ with $0 \leq d_{i,j} \leq D \ \ \forall i = 1..r$ which is present in all the strings as shown in Figure 12.

The string $C_i[j]$ output in Step 6 is an encoding of a structured motif with each gap bounded by a fixed value that is common to all the input strings $\{S_1, S_2, \ldots, S_r\}$.

The construction of the generalised suffix tree at Step 1 requires $O(nr \log |\Sigma|)$ time, where $n = |S_i|$. The transformation of the strings S into $\tilde{S}$ requires linear $O(nr)$ time. For each position i in each string, the number of motifs that may have an occurrence at i is at most D^{m-1}. Hence, there are overall $r * n * D^{m-1}$ motifs. Step 3 takes $O(rn)$ assuming that m and D are constants. Again, the construction of the generalised suffix tree of Step 5, as well as Step 6, can be done in linear time. The overall complexity of the algorithm for the third variant of the problem is therefore $O(rn)$.

6 Conclusion

We have presented algorithms for three different variants of the problem of finding motifs with gaps. The approach we suggest allows to find motifs with gaps maintaining a linear time complexity with respect to the input size. The technique we applied is general enough that it can be used for several other variants of the problem besides those addressed in this paper.

Another interesting further direction to explore would be to allow errors (*i.e.*, substitutions and insertions deletions) inside the blocks.

References

[1] M. Crochemore, W. Rytter, *Text algorithms*, Oxford Press, 1994.

[2] T. Crawford, C.S. Iliopoulos, R. Raman, String matching techniques for musical similarity and melodic recognition, *Computing in Musicology*, Vol. 11, pp. 73–100, 1998.

[3] M. Farach, Optimal suffix tree construction with large alphabets, in *Foundations of Computer Science (FOCS '97)*, 137–143, 1997.

[4] C. Charras, T. Lecroq, *Handbook of Exact String Matching Algorithms*, King's College London publications 2004.

[5] P. Bieganski, J. Riedl, J. V. Carlis, Generalized Suffix Trees for Biological Sequence Data: Applications and Implementation, in *Proc. Hawaii International Conference on System Sciences*, 1994.

[6] J. Allali, M.-F. Sagot, The at-most k-deep factor tree, Internal Report IGM 2004-03 (Institut Gaspard Monge), 2004.