EXACT ALGORITHMS FOR MOTIF SEARCH*

S. RAJASEKARAN, S. BALLA, C. HUANG, AND V. THAPAR
Dept. of CSE, Univ. of Connecticut,
Storrs, CT 06269, USA

M. GRYK, M. MACIEJEWSKI, AND M. SCHILLER
Dept. of Neuroscience, Univ. of Connecticut,
Storrs, CT 06030, USA

In this paper we study the problem of identifying meaningful patterns (i.e., motifs) from biological data. The general version of this problem is NP-hard. Numerous algorithms have been proposed in the literature to solve this problem. Many of these algorithms fall under the category of heuristics. We concentrate on exact algorithms in this paper. In particular, we concentrate on two different versions of the motif search problem and offer exact algorithms for them.

1 Introduction

The human genome project has resulted in the generation of voluminous biological data. Novel computational techniques are called for to extract useful information from this data. One such technique is that of finding patterns that are repeated over many sequences (and possibly over many species). Several variants of this motif search problem have been identified in the literature (see e.g., [3][6][13]). In this section we define two versions of motif search and list some related publications.

Problem 1. A *pattern* is a string of symbols (also called *residues*) and ?'s. A "?" refers to a wild card character. A pattern cannot begin or end with ?. AB?D, EB??DS?R, etc. are examples of patterns. The *length* of a pattern is the number of characters in it (including the wildcard characters). This problem takes as input a database *DB* of sequences. The goal is to identify all the patterns of length at most *P* (with anywhere from 0 to $\lfloor P/2 \rfloor$ wild card characters). In particular, the output should be all the patterns together with a count of how many times each pattern occurs. Optionally a threshold value for the number of occurrences could be supplied.

The above motif model has been derived as follows. We have generated a list of 312 minimotifs (i.e., motifs of short length) that have defined biological functions. We have used this list to select parameters for a de novo analysis of novel minimotifs in the human proteome. In this paper we choose to analyze novel motifs with a length (*P*) of 10 amino acids because 92% of the previously characterized minimotifs in our list are less than 10 amino acids in length. Another reason for choosing a length of 10 amino acids is based

* This research was supported in part by the NSF Grants CCR-9912395 and ITR-0326155.

on the function of minimotifs. Most minimotifs are in binding domains or substrates of enzymes. The peptide ligand binding surfaces on proteins in the Protein Data Bank is usually no longer than 35 angstroms. A 10 amino acid peptide would achieve a maximum of 35 angstroms in length if it were in a random coil or a beta-sheet structure. Thus, the selection of a length of 10 amino acids is consistent with the length of peptides that interact with binding surfaces on protein domains.

The average minimotif in our list has 2.1 wildcard positions for any amino acid. Wildcards signify any of the 20 amino acids. Since only 13 % of minimotifs in our list have more than 50% wild card positions, we chose $P/2$ or 5 wild cards as the maximal number in the algorithm.

The second version of motif search considered in this paper is defined next as Problem 2. The main focus of this paper is Problem 1. We provide algorithms, analyses, and experimental results for Problem 1. For Problem 2 we provide algorithms and analyses. Our algorithms for Problem 1 as well as Problem 2 employ radix sorting as the basic technique. We believe that similar techniques can be used for solving other variations of the motif search problem as well. For example, we have solved a variant called the *planted motif search problem* using similar techniques.

Problem 2. The input is a database *DB* of sequences $S_1, S_2, \ldots, S_n$. Input also are integers *P, D,* and *q*. Output should be all the patterns in *DB* such that each pattern is of length *P* and it occurs in at least *q* of the *n* sequences. A pattern *U* is considered an occurrence of another pattern *V* as long as the *edit distance* between *U* and *V* is at most *D*.

The TEIRESIAS algorithm of Floratos and Rigoutsos [6] addresses a problem similar to Problem 1. The run time of this algorithm *is* $\Omega(P^{P/2}M\log M)$, where *M* is the size of the database (i.e., the number of characters (or residues) in the database). An algorithm for Problem 2 has been given by Sagot [13] that has a run time of $O(n^2mP^D|\Sigma|^D)$ where *m* is the average length of the sequences in *DB*. The space requirement of this algorithm is $O(n^2m)$. An algorithm with an expected run time *of* $O(nm + D(nm)^{1+pow(\varepsilon)}\log nm)$ where $\varepsilon = D / P$ and $pow(\varepsilon)$ is an increasing concave function has been proposed by Adebiyi and Kaufmann [2]. The value of $pow(\varepsilon)$ is roughly 0.9 for protein and DNA sequences.

2 Results of this Paper

In this paper we present algorithms for Problems 1 and 2. All of our algorithms are elegant and use only such simple data structures as arrays.

The goal of Problem 1 is to identify all the patterns of length at most *P* (with anywhere from 0 to $|P/2|$ wild card characters) in a given database. In particular, the output should be all the patterns together with a count of how many times each pattern occurs. Optionally a threshold value for the number of occurrences could be supplied. Determining this threshold is a challenging task. One way of determining this threshold is to rank the motifs in the order of the number of their occurrences and choosing certain

number of them (either because they are over-represented or because they are under-represented). Another way of determining the threshold is by analyzing the table of occurrences of all the patterns in the database together with a model for the biological sequences under concern. This differs from the first way in that here the number of occurrences of any motif will be weighted as dictated by the model. A typical value for P is 10. We present a simple sorting based algorithm for Problem 1 (in Section 4). This algorithm has been coded and experimental results have been obtained. The run time of this algorithm is $O(P^{P/2}M)$ for a given pattern length P, the number of wild cards being at most $P/2$. The number of residues in the database is M. In Section 5 we illustrate the usefulness of sampling in motif search. In Section 6 we present a randomized algorithm for Problem 2 that has the potential of performing better than the algorithms of [13] and [2]. This is a Monte-Carlo algorithm with a run time of $O((Dn^2m^2\log n)/q + gmnD)$, where g is the number of P-mers in the database DB that occur in around q or more sequences in DB. When q is large, the above run time could be $o(mn+D(mn)^{1+pow(\varepsilon)}\log mn)$. In Section 7 we present our experimental results. Section 8 concludes the paper.

3 Previous Sorting Based Algorithm (Problem 1)

The algorithm of Martinez [10] addresses a variant of Problem 1. In particular, the input is just one sequence. The output consists of all repeated patterns. The matches of interest are **exact.** Even if the input has many sequences, they can be concatenated to get a single sequence.

The algorithm of [10] works as follows. Let $S = x_1x_2...x_n$ be the input sequence. This sequence can be thought of as n sequences where each sequence corresponds to a `suffix' of S. I.e., S_1 is the same as S; $S_2 = x_2x_3...x_n$; and so on. These n sequences are then sorted one residue at a time. At any level of the sorting we have groups of sequences. In particular, after k levels of sorting, two sequences are in the same group, if the first k residues are the same in these two sequences. Sorting at any level is local to groups. A group will not be sorted further if it has only one element.

The expected run time of the above algorithm is $O(n \log n)$ whereas its worst case run time is $\Omega(n^2)$.

The above algorithm can be modified to have an expected run time of $O(n)$ by performing radix sorting with respect to the first $\Omega(\log n)$ residues of the n sequences (see e.g., [8]).

As another variant consider a problem where the input are a sequence S and an integer k. The goal is to report all repeats of length k. This variant can be solved in the worst case in time $O(nk/w)$, where w is the word length of the computer as follows. 1) Form all k-mers of S. There are less than n such k-mers; 2) Sort these k-mers lexicographically in time $O(nk/w)$; and 3) Scan through the sorted list to identify the repeats.

Instead of the above algorithm one could also employ a prefix tree or a suffix array to get a run time of O(n). Depending on the underlying constant and the values of k and w, the above algorithm could be faster.

4 Simple Motif Search (SMS) (Problem 1)

As has been pointed out before, we are interested in identifying **all** the patterns of length at most P (with anywhere from 0 to $|P/2|$ wild card characters). For every pattern, the number of occurrences should be output. How does a biologist identify *biologically important* patterns? This is a challenging task for biologists and will not be addressed in this paper.

Define a (u, v)-class as a class of patterns where each pattern has length u and has exactly v wild card characters. For example, GA??C?T belongs to (7, 3)-class. Note that there are

$\binom{u-2}{v} |\Sigma|^{u-v}$ patterns in a (u, v)-class. Similar notations have been used before.

To identify the patterns in a (u, v)-class, we perform

$\binom{u-2}{v}$ sorts. More specifically, for each possible placement of v wild card characters (excluding at the end positions) in a sequence of length u, we perform a sorting. As an example, consider a case where u=5 and v=2. There are three possible placements: C??CC, CC??C, and C?C?C, where C corresponds to any residue. Call every placement as a (u, v)-*pattern type.*

For every (u, v)-pattern type, we perform the following steps.

Algorithm SMS

For every (u, v)-pattern type do

1. If R is a pattern type in (u, v)-class, we generate all possible u-mers in all the sequences of DB. If the sequences in DB have lengths $l_1, l_2, \ldots, l_n$, respectively, then the number of u-mers from S_i is $l_i - u + 1$, for $1 \le i \le n$.
2. Sort all the u-mers generated in step 1 only with respect to the non-wild card positions of R. For example, if the pattern type under concern is CC??C?C, we generate all possible 7-mers in DB and sort the 7-mers with respect to positions 1,2,5, and 7. Employ radix sort (see e.g., [8]).
3. Scan through the sorted list and count the number of occurrences of each pattern.

The run time of the above algorithm is

$O\left(\binom{u-2}{v} M \frac{u}{w}\right)$ for a (u, v)-class, where M is the total number of residues in *DB* and w is the word length of the computer.

Now we consider the problem of identifying all of the following patterns: The maximum length is 10. Pattern lengths of interest are: 3, 4, 5, 6, 7, 8, 9 and 10. The maximum number of wild cards are 1, 2, 2, 3, 3, 4, 4 and 5, respectively. In other words we are interested in: (10, 5)-class, (10, 4)-class, …, (10, 1)-class, (9, 4)-class, (9, 3)-class, …, (9, 1)-class, …, (4, 2)-class, (4, 1)-class, and (3,1)-class.

Thus the total number of sorts done is

$$\sum_{i=0}^{5}\binom{8}{i}+\sum_{i=0}^{4}\binom{7}{i}+\sum_{i=0}^{4}\binom{6}{i}+\sum_{i=0}^{3}\binom{5}{i}+\sum_{i=0}^{3}\binom{4}{i}+\sum_{i=0}^{2}\binom{3}{i}+\sum_{i=0}^{2}\binom{2}{i}+\sum_{i=0}^{1}\binom{1}{i}=429.$$

Theorem 4.1. SMS algorithm runs in time $O(P^{P/2}M)$.

5 Random Sampling

Random sampling can be employed to speedup computations. In this section we describe a simple form of sampling as it applies to Problem 1. We prove the following Lemma, the proof of which is omitted due to space constraints.

Lemma 5.1. Consider the problem of identifying patterns in a database of n sequences. Each pattern of interest should occur in at least q of the input sequences. To solve this problem it suffices to use a random sample of size εn and a sample threshold of $(1-\alpha)\varepsilon q$. In this case, with high probability, no pattern that has an occurrence of less than $(1-\alpha)q/(1+\alpha)$ in *DB* will pass the sample threshold, provided

$$q \geq \frac{3}{\alpha^2 \varepsilon}\frac{1+\alpha}{1-\alpha}(\beta \ln n + \ln M). \square$$

6 Motif Search with Edit Distance (Problem 2)

In this section we consider Problem 2. Here the input is a database *DB* of sequences S_1, S_2, …, S_n. Input also are integers *P, D,* and *q*. The output should be all the patterns in the *DB* such that each pattern is of length *P* and it occurs in at least *q* of the *n* sequences. A pattern *U* is considered an occurrence of another pattern *V* as long as the *edit distance* between *U* and *V* is at most *D*.

An algorithm for the above problem has been given by Sagot [13] that has a run time of $O(n^2mP^D|\Sigma|^D)$ where m is the average length of the sequences in *DB*. An algorithm with an expected run time of $O(nm+D(nm)^{1+pow(\varepsilon)}\log nm)$ where $\varepsilon = D/P$ and $pow(\varepsilon)$ is an increasing concave function has been given in [2]. The value of $pow(\varepsilon)$ is roughly 0.9 for protein and DNA sequences.

In the following discussion the word "occurrence" is used to denote occurrence within an edit distance of *D,* and the word "presence" is used to denote exact occurrence (i.e., occurrence with an edit distance of zero).

In this section we present a simple randomized algorithm that has the potential of performing better than the algorithms of [13] and [2]. The algorithms in [13][2] employ suffix trees and our algorithm uses arrays.

Before presenting the randomized algorithm we present a very simple algorithm. The randomized algorithm is based on this simple algorithm. This algorithm works as follows.

1. Generate all possible *P*-mers in *DB*. Let the collection of these *P*-mers be *C*. There are at most *mn* elements in *C*. Duplicates in *C* could be eliminated by a simple radix sort.
2. For every *P*-mer *U* in *C*, compute the number of occurrences of *U* in *DB*. This can be done in time $O(nmD)$ using the algorithm of Galil and Park [7]. (See also [1] [9] [11] [12] [15]).

   Thus we get the following Theorem.

   Theorem 6.1. Problem 2 can be solved in time O(n^2m^2D). □

A Randomized Algorithm. A randomized algorithm can be developed based on the above algorithm.

1. Generate all possible *P*-mers in *DB*. Let *C* the collection of these *P*-mers. *C* has at most *nm* elements.
2. For each element *U* in *C,* pick a random sample S_U from *DB* of $(16\alpha n \ln n)/q$ sequences where α is the probability parameter (assumed to be a constant). Count the number of occurrences N_U of *U* in the sample. This will take time $|S_U|mD$ (using the algorithm of Galil and Park [7]) for a single *U*.
3. For each *U* in *C* such that $N_U > 10.34\alpha \ln n,$ compute the occurrences of *U* in the entire input *DB*. If the number of occurrences of *U* in *DB* is *q* or more, then output *U*.

Theorem 6.2. The above algorithm runs in time

$$O\left(\frac{n^2m^2 \log n}{q} D + gmnD\right)$$

where *g* is the number of *P*-mers that pass the test in step 3. Also, the probability of an incorrect answer is no more than $n^{-\alpha}nm$. The space used is linear in the input size.

Proof. The run time is easy to see. Note that if a *P*-mer occurs in less than *q* input sequences, it will never be output. If a *P*-mer *U* occurs in at least *q* sequences of *DB*, then the number of occurrences of *U* in S_U (i.e., the value of N_U) is lower bounded by a binomial random variable with mean $16\alpha \ln n$. An application of the Chernoff bounds (second equation) with $\varepsilon = 1/(2\sqrt{2})$ shows that the probability that N_U is less than

$10.34\alpha \ln n$ is no more than $n^{-\alpha}$. On the same token, let *U'* be a *P*-mer that occurs in at most (3/8)*q* of the input sequences. The number of occurrences $N_{U'}$ of *U'* in the sample is a binomial with mean $6\alpha \ln n$. Using Chernoff bounds equation 3 with $\varepsilon = 1/\sqrt{2}$, probability that $N_{U'}$ exceeds $10.25\alpha \ln n$ is at most $n^{-\alpha}$.

In summary, if a pattern occurs in q or more input sequences, it will pass the test of step 3 with high probability. Moreover, not many spurious patterns will pass the test of step 3. Also note that there at most nm patterns of interest.□

Note that this algorithm has the potential of performing better than that of [2], especially for large values of *q*. When *q* is large (εn, for some constant fraction ε, for instance), *g* can be expected to be small and hence the entire run time could be $o(D(nm)^{1+pow(\varepsilon)} \log nm)$ We have shown that the expected value of *g* is very small. The proof is omitted due to space constraints.

7 Experimental Results

We have implemented the algorithm SMS (for Problem 1) both sequentially and in parallel. In this section we describe the experimental results.

7.1. *Sequential Implementation*

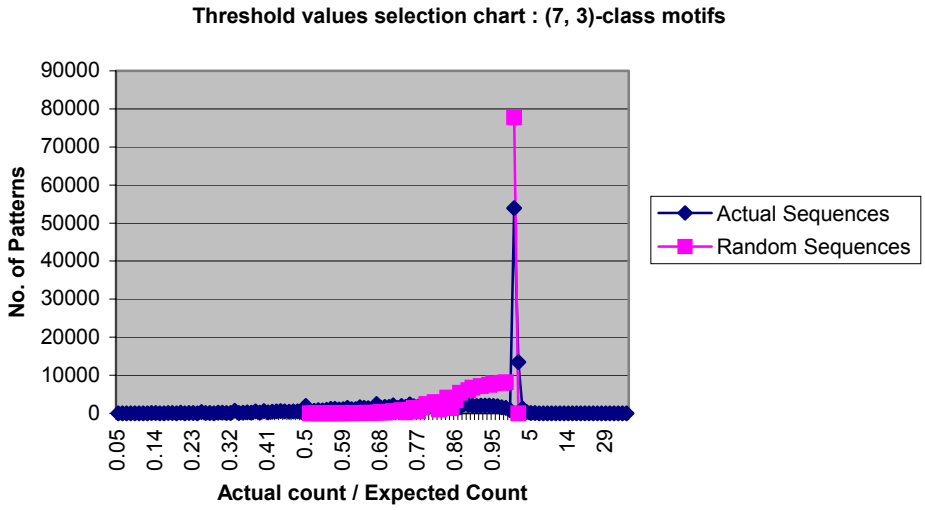
We employ the following form of radix sort: Let $k_1, k_2, \ldots, k_N$ be a sequence of keys to be sorted where each key is a string of residues. Sorting is done with respect to *d* residues at a time. The optimal value for *d* can be decided empirically. For the proteome database, a value of *d*=3 proved to be optimal.

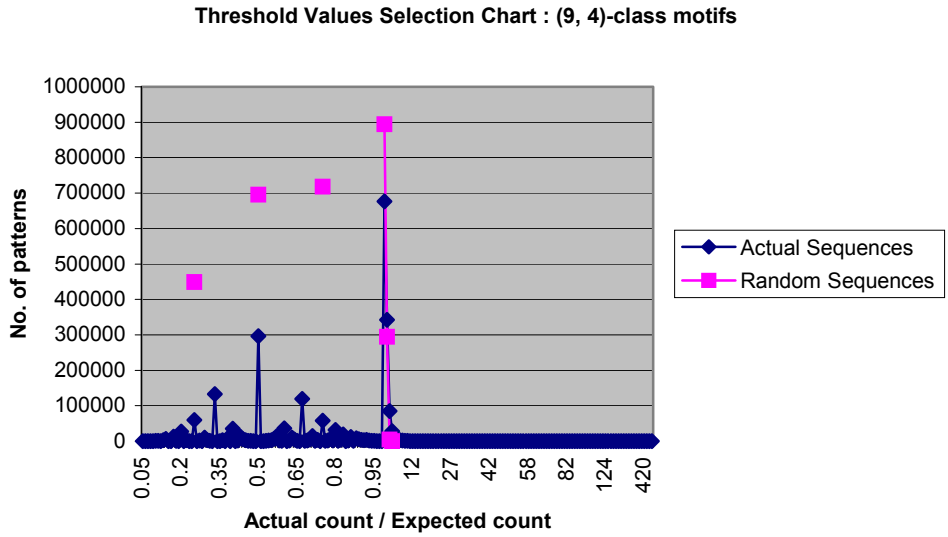
When *d*=3, the algorithm runs as follows. The only data structures used are arrays. For the proteome database, $|\Sigma| = 20$. We can think of each key to be sorted as an integer in the range [1, 8000]. An array *c*[1:8000] whose entries are initialized to zeros is employed. There are two phases in the algorithm each phase involving a scan through the input sequence. In the first phase input keys are processed one at a time starting from k_1. When key k_i is processed, $c[k_i]$ is incremented by one. Thus at the end of the first phase, *c*[*j*] has the number of input keys whose value is *j* (for $1 \le j \le 8000$). In the second phase prefix sums of the array *c* are computed. A second scan through the input sequence is done and each key is output in an array in an appropriate (stable-sorted) place. The prefix sums are useful in deciding an appropriate output index for each key.

We have employed our algorithm SMS on various proteome sequences. As an example, we report the results pertaining to the Human proteome sequences (RefSeq database). This database has 10,046,356 residues and 19,244 sequences. The average length of the sequences in the database is 522.

On a Pentium 4, 2.4 Ghz machine with 1 GB RAM, SMS takes around 7.25 hours. The following graphs show the distribution of patterns in RefSeq and a database of random sequences (of a comparable size). The patterns shown are for (7,3)-class and (9,4)-class. In the x-axis we show the ratio of the actual number of occurrences of a

pattern to the expected number of occurrences. In the y-axis we show the number of patterns that have a specific ratio.





7.2. *Speedup Techniques*

The run time of SMS can be improved using the fact that $\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}$.

We omit details due to space constraints.

We can speedup SMS further and also reduce the memory requirements as follows. Let $S_1, S_2, ..., S_n$ be the input sequences where $S_i = s(i,1), s(i,2), ..., s(i, l_i)$, l_i being the length of the sequence S_i. Each residue in the database can be uniquely identified by a combination of the sequence index and the position of the residue in that sequence.

Definition 7.1. : A *position pair* is defined as a pair of indices *(i, j)* and this pair represents the *j*th residue in S_i.

The basic idea is to save memory by not explicitly generating all possible *u*-mers (as mentioned in step 1 of Algorithm SMS). We work with the position pairs to perform the sorts (mentioned in step 2 of Algorithm SMS). In particular, each position pair corresponds to a *u*-mer. Whenever any portion of a *u*-mer is needed, this portion is obtained from the list of position pairs. Details have been omitted due space limits.

7.3. *Experimental Data*

We have employed our algorithm SMS on various proteome sequences. As an example, to report novel motifs in the Human proteome sequences of RefSeq database, that has 10,046,356 residues and 19,244 sequences, with the average length of the sequences as 522 amino acids, on a Pentium 4, 2.4 GHz machine with 1 GB RAM, SMS takes around seven hours.

7.4. *Parallelism*

SMS is amenable to parallel implementations. One possibility is to partition the number of sorts equally among the processors. A second possibility is to partition the sequences as equally among the processors as possible and finally merge the occurrence numbers of patterns. A third possibility is to treat each sort as a job. We have employed the third approach to parallelize SMS. The speedups obtained have been very close to linear. We omit details due to space constraints.

8 Conclusions

In this paper we have considered two versions of the motif search problem and offered exact solutions for these versions. The algorithms presented have the potential of performing well in practice. An interesting open problem is to implement all the algorithms that have been proposed for Problem 2 in the literature and determine under what conditions which algorithms will perform better.

References

1. E. F. Adebiyi, T. Jiang and M. Kaufmann, An efficient algorithm for finding short approximate non-tandem repeats, *Bioinformatics* 17, Supplement 1, 2001, pp. S5-S12.

2. E. F. Adebiyi and M. Kaufmann, Extracting common motifs under the Levenshtein measure: theory and experimentation, *Proc. Workshop on Algorithms for Bioinformatics (WABI),* Springer-Verlag LNCS 2452, 2002, pp. 140-156.
3. J. Buhler and M. Tompa, Finding motifs using random projections, *Proc. Fifth Annual International Conference on Computational Molecular Biology (RECOMB),* April 2001.
4. H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, *Annals of Math. Statistics* 23, 1952, pp. 493-507.
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.
6. A. Floratos and I. Rigoutsos, On the time complexity of the TEIRESIAS algorithm, Research Report RC 21161 (94582), IBM T.J. Watson Research Center, April 21, 1998.
7. Z. Galil and K. Park, An improved algorithm for approximate string matching, *SIAM Journal of Computing* 19(6), 1990, pp. 989-999.
8. E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms,* W. H. Freeman Press, 1998.
9. G. M. Landau and U. Vishkin, Introducing efficient parallelism into approximate string matching and a new serial algorithm, *Proc. ACM Symposium on Theory of Computing,* 1986, pp. 220-230.
10. H. M. Martinez, An efficient method for finding repeats in molecular sequences, *Nucleic Acids Research* 11(13), 1983, pp. 4629-4634.
11. E.W. Myers, Incremental alignment algorithms and their applications, Technical Report 86-22, Department of Computer Science, University of Arizona, Tucson, AZ 85721, 1986.
12. E. W. Myers, A sublinear algorithm for approximate keyword searching, *Algorithmica* 12, 1994, pp. 345-374.
13. M. F. Sagot, Spelling approximate repeated or common motifs using a suffix tree, Springer-Verlag LNCS 1380, pp. 111-127, 1998.
14. E. Ukkonen, Finding approximate patterns in strings, *Journal of Algorithms* 6, 1985, pp. 132-137.