MoTeX: A word-based HPC tool for MoTif eXtraction

Solon P. Pissis
Florida Museum of Natural History, University of Florida, USA &
Heidelberg Institute for Theoretical Studies, Germany
solon.pissis@h-its.org

Alexandros Stamatakis
Heidelberg Institute for Theoretical Studies, Germany
alexandros.stamatakis@h-its.org

Pavlos Pavlidis
Foundation for Research and Technology – Hellas
Institute of Molecular Biology and Biotechnology, Greece

ABSTRACT

Motivation: Identifying repeated factors that occur in a string of letters or common factors that occur in a set of strings represents an important task in computer science and biology. Such patterns are called *motifs*, and the process of identifying them is called *motif extraction*. In biology, motifs may correspond to functional elements in DNA, RNA, or protein molecules. Motifs may also correspond to whole loci whose sequences are highly similar because of recent duplication (e.g., transposable elements or recently duplicated genes). A DNA motif is a nucleic acid sequence that has a specific biological function, for instance encoding the DNA binding sites for a regulatory protein (transcription factor).

Results: In this article, we introduce MoTeX, the first high-performance computing (HPC) tool for MoTif eXtraction from large-scale datasets. It uses state-of-the-art algorithms for solving the fixed-length approximate string matching problem. MoTeX comes in three flavors: a standard CPU version; an OpenMP-based version; and an MPI-based version. We show that MoTeX produces similar and partially identical results to current state-of-the-art tools with respect to accuracy as quantified by statistical significance measures. Moreover, we show that it matches or outperforms competing tools in terms of runtime efficiency. The MPI-based version of MoTeX requires only one hour to process all human genes on 1056 processors, while current sequential programmes require more than two months for this task.

Availability: `http://www.exelixis-lab.org/motex` (open-source code)

Categories and Subject Descriptors

G.4 [**Mathematical Software**]: Algorithm design and analysis

General Terms

Algorithms

Keywords

motif extraction; DNA motif; high-performance computing; text mining

1. INTRODUCTION

Identifying repeated factors that occur in a string of letters or common factors that occur in a set of strings represents an important task in computer science and biology. Such patterns are called *motifs*, and the process of identifying them is called *motif extraction*. Motif extraction has numerous direct applications in areas that require some form of *text mining*, that is, the process of deriving reliable information from text [17]. Here we focus on its application to molecular biology.

In biological applications, motifs correspond to functional and/or conserved DNA, RNA, or protein sequences. Alternatively, they may correspond to (recently, in evolutionary terms) duplicated genomic regions, such as transposable elements or even whole genes. It is mandatory to allow for a certain number of mismatches between different occurrences of the same motif since both single nucleotide polymorphisms as well as errors introduced by wet-lab sequencing platforms might have occurred. Hence, molecules that encode the same or related functions do not necessarily have *exactly* identical sequences.

A DNA motif is defined as a sequence of nucleic acids that has a specific biological function (e.g., a DNA binding site for a regulatory protein). The pattern can be fairly short, 5 to 20 base-pairs (bp) long, and is known to occur in different genes [19], or several times within the same gene [21]. The DNA motif extraction problem is the task of detecting overrepresented motifs as well as conserved motifs in a set of orthologous DNA sequences. Such conserved motifs may, for instance, be potential candidates for transcription factor binding sites.

A single *motif* is a string of letters (word) on an alphabet Σ. Given an integer error threshold e, a motif on Σ is said to *e-occur* in a string s on Σ, if the motif and a factor (substring) of y differ by a distance of e. In accordance with the pioneering work of Sagot [22], we formally define the repeated motifs problem and the common motifs problem as follows:

The *repeated motifs* problem takes as input a string s, the quorum $q \geq 2$, the maximal allowed distance (error threshold) $e \geq 0$, and the length k for the motifs. It consists in finding all motifs of length k, such that each motif e-occurs at least q times in s. Such motifs are called *valid*. The values for k, e, and q are determined empirically.

The *common motifs* problem is a generalization of the repeated motifs problem. It takes as input a set $s_1, \dots, s_N$ of strings on Σ, where $N \geq 2$, the quorum $1 \leq q \leq N$, the maximal allowed distance e, and the length k for the motifs. It consists in determining all motifs of length k, such that each motif e-occurs in at least q input strings. Such motifs are called *valid*.

1.1 Motif extraction algorithms

In accordance with [7], motif extraction algorithms can be divided into two major classes: (1) *word-based* (string-based) methods that mostly rely on exhaustive enumeration, that is, counting and comparing oligonucleotide sequence (k-mer) frequencies; and (2) *probabilistic sequence* models, where the model parameters are estimated using maximum-likelihood or Bayesian inference methods.

Word-based methods guarantee global optimality and are suitable for detecting short motifs ranging between 5 to 20 bp. Therefore, they are useful for motif finding in eukaryotic genomes where motifs are generally shorter than in prokaryotes [7]. Word-based methods are also relatively fast, when implemented using efficient data structures, such as *suffix trees* [26]. They also represent a good choice for finding totally *constrained* motifs, that is, all motif instances are exactly identical (i.e., $e := 0$). However, for typical transcription factor motifs that often have several weakly constrained positions (several observed mutations), word-based methods can be problematic. The results often need to be post-processed using clustering methods [27]. Word-based methods also suffer from the problem of producing too many spurious motifs (see Section 5 for details).

In probabilistic sequence models, motifs are modeled by a position weight matrix [3]. Position weight matrices are often visualized as pictograms where each position is represented by a stack of letters whose height is proportional to the observed occurrence frequency of the respective letter at the specific position [23]. Probabilistic methods have the advantage of tying their search parameters by incorporating background knowledge (e.g., accounting for palindromes). The disadvantage is that they rely on probabilistic models of the regulatory regions. These models are sensitive to small changes in the input data [7]. The majority of probabilistic motif extraction algorithms has been designed to find longer or more general motifs than required for identifying transcription factor binding sites. Therefore, they are more appropriate for motif finding in prokaryotes, where the motifs are generally longer than in eukaryotes. However, these algorithms are not guaranteed to find globally optimal solutions. This is because they employ local searches, such as Gibbs sampling, expectation maximization, or greedy algorithms, that may converge to locally optimal solutions only.

Here, we focus on word-based methods for motif extraction. A plethora of word-based tools for motif extraction, such as RISO [18, 22], MITRA [9], YMF [24], Weeder [19], FLAME [10], and RISOTTO [4], have already been released. The comprehensive study by Tompa *et al.* [25] compared thirteen different word-based and probabilistic methods on real and synthetic datasets, and identified Weeder and YMF—which are both word-based—as the most effective methods for motif extraction.

Weeder initially indexes the set of N sequences using a generalized suffix tree [26], and subsequently searches for non-exact occurrences of motifs along different paths of the suffix tree. This approach was first introduced by Sagot in RISO [22]. For every valid motif, one has to walk along at most $N \times n$ different paths in the suffix tree, where n is the average sequence length. For every word of length k induced by a path in the tree, there are at most $|\Sigma|^e k^e$ valid motifs, where $|\Sigma|$ is the size of the alphabet Σ, and e is the maximal allowed number of mismatches. Hence, the overall time complexity of this approach is $\mathcal{O}(|\Sigma|^e k^e N^2 n)$. The additional factor N is required to annotate the suffix tree nodes [19]. Weeder further assumes that the mismatches are distributed uniformly along the motif. Because of this assumption, the search space explored by Weeder can be substantially decreased. Thus, the time complexity is reduced to $\mathcal{O}(\lceil 1/\epsilon \rceil^e |\Sigma|^e N^2 n)$, where $\epsilon < 1$ is an initial error ratio determined by the algorithm, such that $e = \lceil \epsilon k \rceil$. RISOTTO, the successor of RISO, improved upon the average-case time complexity. It also runs more than two times faster than RISO in practice [4]. Moreover, RISOTTO can extract *structured motifs*, which are motifs composed of several disjoint single motifs placed at given distances from each other.

YMF is based on an empirical probabilistic motif model that was derived from a study of known transcription factor binding sites in *yeast*. The input of the algorithm is a set of N DNA sequences, the number of letters in the motifs to be enumerated, and the transition matrix for a Markov chain of order m constructed from the $(m+1)$-mer frequencies of the full yeast genome. The algorithm finds all motifs that occur more frequently than the expected value in the set of DNA sequences. It is thus guaranteed to report the *best* motifs according to the YMF scoring criterion. It simply maintains a counter that corresponds to each possible motif. It scans the set of DNA sequences once using a sliding window and increments the counter for each motif that matches the sliding window. YMF scales linearly with the total length of the input sequences, but scales poorly with motif length, since it maintains a counter for each possible motif. It has been shown that, in cases where the known motif cannot be captured by the YMF's derived motif model, YMF is not as accurate [24].

1.2 Our Contribution

All aforementioned motif extraction algorithms exhibit a part of the following disadvantages: First, their time complexity depends on the motif length k. Hence, they can only be used for finding very short motifs. For instance, YMF allows only up to $k := 8$ and Weeder up to $k := 12$. Second, their time complexity depends on the size $|\Sigma|$ of the alphabet. Hence, they are not suitable for detecting motifs drawn from large alphabets (e.g., amino acids, where $|\Sigma| = 20$). Third, their time complexity grows exponentially with the maximal allowed number of mismatches e. Thus, they are not suitable for detecting longer motifs with high error rates, say $k := 13$ and $e := 4$. Finally, the probabilistic sequence models are not guaranteed to find globally optimal solutions.

There are two additional disadvantages: First, although some theoretical work has been done [1, 2, 15], existing tools are only designed for identifying motifs under the Hamming distance model (mismatches) but not under the edit distance model (indels). Indels in biological sequences may occur because of insertions or deletions of genomic segments at various genomic locations or due to sequencing errors. Second, they are not designed or implemented for high-performance computing (HPC). For instance, Weeder and RISOTTO, which are currently two of the most widely used tools for motif extraction, require more than two months to process the full upstream *Homo sapiens* genes dataset with $k := 12$ and $e := 4$, making this kind of analyses intractable.

To alleviate these shortcomings, we introduce `MoTeX`, the first word-based HPC tool for MoTif eXtraction from large-scale datasets. It can be used to tackle both the repeated motifs problem and the common motifs problem by making a stricter assumption on motif validity, which we will elaborate later on. `MoTeX` is based on string algorithms for solving the so-called fixed-length approximate string matching problem under the edit distance model [14] and under the Hamming distance model [6]. Given that $k \leq w$, where w is the size of the computer word (in practice, $w = 64$ or $w = 128$), the time complexity of this approach is $\mathcal{O}(n^2)$ for the repeated motifs problem and $\mathcal{O}(N^2n^2)$ for the common motifs problem. The parallel time complexity is $\mathcal{O}(n^2/p)$ for the repeated motifs problem and $\mathcal{O}(N^2n^2/p)$ for the common motifs problem, where p is the number of available processors.

Hence, `MoTeX` exhibits the following advantages: under the realistic assumption that $k \leq w$, time complexity does not depend on (i) the length k for the motifs (ii) the size $|\Sigma|$ of the alphabet, or (iii) the maximal allowed distance e. Given the stricter assumption on motif validity, it is guaranteed to find globally optimal solutions. Furthermore, the size of the output is linear with respect to the size of the input. In addition, `MoTeX` can identify motifs either under the edit distance model or the Hamming distance model. Finally, apart from the standard CPU version, `MoTeX` comes in two HPC flavors: the OpenMP-based version that supports the *symmetric multiprocessing programming* (SMP) paradigm; and the MPI-based version that supports the *message-passing programming* (MPP) paradigm. For instance, the MPI-based version of `MoTeX` requires about one hour to process the full upstream *Homo sapiens* genes dataset using 1056 processors, for *any* value of k and e.

2. DEFINITIONS AND NOTATION

In this section, in order to provide an overview of the algorithms used later on, we give a few definitions, generally following [5].

An *alphabet* Σ is a finite non-empty set whose elements are called *letters*. A *string* on an alphabet Σ is a finite, possibly empty, sequence of elements of Σ. The zero-letter sequence is called the *empty string*, and is denoted by ε. The *length* of a string x is defined as the length of the sequence associated with the string x, and is denoted by $|x|$. We denote by $x[i]$, for all $1 \leq i \leq |x|$, the letter at index i of x. Each index i, for all $1 \leq i \leq |x|$, is a position in x when $x \neq \varepsilon$. It follows that the ith letter of x is the letter at position i in x, and that

$$x = x[1\,..\,|x|].$$

A string x is a *factor* of a string y if there exist two strings u and v, such that $y = uxv$. Let the strings x, y, u, and v, such that $y = uxv$. If $u = \varepsilon$, then x is a *prefix* of y. If $v = \varepsilon$, then x is a *suffix* of y.

Let x be a non-empty string and y be a string. We say that there exists an (exact) *occurrence* of x in y, or, more simply, that x *occurs* (exactly) in y, when x is a factor of y. Every occurrence of x can be characterised by a position in y. Thus we say that x occurs at the *starting position* i in y when $y[i\,..\,i + |x| - 1] = x$. It is sometimes more suitable to consider the *ending position* $i + |x| - 1$.

The *edit distance*, denoted by $\delta_E(x, y)$, for two strings x and y is defined as the minimum total cost of operations required to transform string x into string y. For simplicity, we only count the number of edit operations and consider that the cost of each edit operation is 1. The allowed operations are the following:

- `Ins`: insert a letter in y, not present in x; $(\varepsilon, b),\ b \neq \varepsilon$;
- `Del`: delete a letter in y, present in x; $(a, \varepsilon),\ a \neq \varepsilon$;
- `Sub`: substitute a letter in y with a letter in x; $(a, b),\ a \neq b,\ a, b \neq \varepsilon$.

The Hamming distance δ_H is only defined on strings of the same length. For two strings x and y, $\delta_H(x, y)$ is the number of positions in which the two strings differ, that is, have different letters. Formally

$$\delta_H(x, y) = \sum_{i=1}^{|x|} 1_{x[i]\neq y[i]}\,,\text{ where } 1_{x[i]\neq y[i]} = \begin{cases} 1, & \text{if } x[i] \neq y[i] \\ 0, & \text{otherwise} \end{cases}$$

For the sake of completeness, we define $\delta_H(x, y) = \infty$ for strings x, y such that $|x| \neq |y|$.

3. ALGORITHMS

In this section, we first formally define the *fixed-length approximate string matching* problem under the edit distance model and under the Hamming distance model. We then provide a brief description and analysis of the sequential algorithms to solve it. Finally, we show how both the repeated motifs problem *and* the common motifs problem can be reduced to the fixed-length approximate string matching problem, by using a stricter assumption than the one in the initial problem definition for the validity of motifs.

PROBLEM 1 (EDIT DISTANCE). *Given a string x of length m, a string y of length n, an integer k, and an integer $e < k$, find all factors of y, which are at an edit distance less than, or equal to, e from every factor of fixed length k of x.*

PROBLEM 2 (HAMMING DISTANCE). *Given a string x of length m, a string y of length n, an integer k, and an integer $e < k$, find all the factors of y, which are at a Hamming distance distance less than, or equal to, e from every factor of fixed length k of x.*

Let $\mathsf{D}[0\,..\,n, 0\,..\,m]$ be a dynamic programming (DP) matrix, where $\mathsf{D}[i, j]$ contains the edit distance between some factor $y[i'\,..\,i]$ of y, for some $1 \leq i' \leq i$, and factor $x[\max\{1, j - k + 1\}\,..\,j]$ of x, for all $1 \leq i \leq n$, $1 \leq j \leq m$. This matrix can be obtained through a straightforward $\mathcal{O}(kmn)$-time algorithm by constructing DP matrices $\mathsf{D}^s[0\,..\,n, 0\,..\,k]$, for all $1 \leq s \leq m - k + 1$, where $\mathsf{D}^s[i, j]$ is the edit distance between some factor of y ending at $y[i]$ and the prefix of length j of $x[s\,..\,s + k - 1]$. We obtain D by collating D^1 and the last row of D^s, for all $2 \leq s \leq m - k + 1$. We say that $x[\max\{1, j - k + 1\}\,..\,j]$ e-occurs in y ending at $y[i]$ *iff* $\mathsf{D}[i, j] \leq e$, for all $1 \leq j \leq m$, $1 \leq i \leq n$.

Table 1: Matrix D for $x := y :=$ GGGTCTA and $k := 3$

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	1	2	2	2	3	3
2	G	0	0	0	1	1	2	3	3
3	G	0	0	0	0	1	2	3	3
4	T	0	1	1	1	0	1	2	2
5	C	0	1	2	2	1	0	1	2
6	T	0	1	2	3	2	1	0	1
7	A	0	1	2	3	3	2	1	0

Example 1. *Let the string $x :=$ GGGTCTA, the string $y := x$, and $k := 3$. Table 1 illustrates matrix D. Consider, for instance, the case where $j = 6$. Column 6 contains all e-occurrences of factor $x[4 \mathinner{.\,.} 6] =$ TCT, that is the factor of length $k = 3$ ending at position 6 of x, in y. Cell $\mathsf{D}[4, 6] = 2$, tells us that there exists some factor $y[i' \mathinner{.\,.} 4]$ of y, such that, for $i' = 2$, $\delta_E(y[2 \mathinner{.\,.} 4], x[4 \mathinner{.\,.} 6]) = 2$.*

Iliopoulos, Mouchard, and Pinzon devised MaxShift [14], an algorithm with time complexity $\mathcal{O}(m\lceil k/w \rceil n)$, where w is the size of the computer word. By using word-level parallelism MaxShift can compute matrix D efficiently. The algorithm requires constant time for computing each cell $\mathsf{D}[i, j]$ by using word-level operations, assuming that $k \leq w$. In the general case it requires $\mathcal{O}(\lceil k/w \rceil)$ time. Hence, algorithm MaxShift requires time $\mathcal{O}(mn)$, under the assumption that $k \leq w$. The space complexity is $\mathcal{O}(m)$ since each row of D only depends on the immediately preceding row.

Theorem 1 ([14]). *Given a string x of length m, a string y of length n, an integer k, and the size of the computer word w, matrix D can be computed in time $\mathcal{O}(m\lceil k/w \rceil n)$.*

Let $\mathsf{M}[0 \mathinner{.\,.} n, 0 \mathinner{.\,.} m]$ be a DP matrix, where $\mathsf{M}[i, j]$ contains the Hamming distance between factor $y[\max\{1, i-k+1\} \mathinner{.\,.} i]$ of y and factor $x[\max\{1, j-k+1\} \mathinner{.\,.} j]$ of x, for all $1 \leq i \leq n$, $1 \leq j \leq m$. Crochemore, Iliopoulos, and Pissis devised an analogous algorithm [6] that solves the analogous problem under the Hamming distance model with the same time and space complexity.

Theorem 2 ([6]). *Given a string x of length m, a string y of length n, an integer k, and the size of the computer word w, matrix M can be computed in time $\mathcal{O}(m\lceil k/w \rceil n)$.*

Example 2. *Let the string $x :=$ GGGTCTA, the string $y := x$, and $k := 3$. Table 2 illustrates matrix M.*

Example 3. *Let the string $x :=$ GTGAACT, the string $y :=$ GTCACGT, and $k := 3$. Table 3 illustrates matrix M. Consider, for instance, the case where $j = 7$. Column 7 contains all e-occurrences of factor $x[5 \mathinner{.\,.} 7] =$ ACT, that is, the factor of length $k = 3$ ending at position 7 of x, in y. Cell $\mathsf{M}[6, 7] = 1$, tells us that $\delta_H(y[4 \mathinner{.\,.} 6], x[5 \mathinner{.\,.} 7]) = 1$.*

By making the following stricter assumption for motif validity, both the repeated motifs problem and the common motifs problem can be directly and efficiently solved using the above algorithms.

Table 2: Matrix M for $x := y :=$ GGGTCTA and $k := 3$

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	1	2	3	3	3	3
2	G	0	0	0	1	2	3	3	3
3	G	0	0	0	0	1	2	3	3
4	T	0	1	1	1	0	2	2	3
5	C	0	1	2	2	2	0	3	2
6	T	0	1	2	3	2	3	0	3
7	A	0	1	2	3	3	2	3	0

Table 3: Matrix M for $x :=$ GTCACGT, $y :=$ GTGAACT, and $k := 3$

		0	1	2	3	4	5	6	7
		ϵ	G	T	G	A	A	C	T
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	2	2	3	3	3	3
2	T	0	1	0	3	2	3	3	2
3	C	0	1	2	1	3	2	2	3
4	A	0	1	2	3	1	2	3	2
5	C	0	1	2	3	3	2	1	3
6	G	0	0	2	2	3	3	2	1
7	T	0	1	0	3	2	3	3	2

Definition 1. *A valid motif is called* strictly valid *iff it occurs* exactly*, at least once, in (any of) the input string(s).*

Consider, for instance, the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. The number of possible DNA motifs of length $k := 10$ is $|\Sigma|^k = 1{,}048{,}576$. Given a dataset with a size of $\approx$ 1MB, the possibility that there exists a motif that is valid, but not strictly valid, is rather unlikely. In other words, given such a dataset, the possibility that there exists a pattern which does not occur exactly, at least once, in the dataset *and* it also satisfies *all* the restrictions imposed by the input parameters, is rather unlikely.

The repeated motifs problem (detecting strictly valid motifs) can be directly solved by first setting $x := s$ and $y := s$, where s is the input string of length n, and subsequently solving the fixed-length approximate string matching problem for x and y. Consider, for example, the repeated motifs problem under the Hamming distance model. We define an array v of size n, such that $v[j]$, for all $k \leq j \leq n$, denotes the total number of e-occurrences of motif $s[j-k+1 \mathinner{.\,.} j]$ in s; we set $v[j] := 0$, for all $0 \leq j < k$. We can compute array v in a straightforward way: as soon as we have computed $\mathsf{M}[i, j]$, we increment $v[j]$ by one *iff* $\mathsf{M}[i, j] \leq e$, for all $k \leq i \leq n$. As soon as we have computed matrix M, it suffices to traverse array v and report $s[j-k+1 \mathinner{.\,.} j]$, for all $k \leq j \leq n$, as a strictly valid motif *iff* $v[j] \geq q$. Maintaining array v does not induce additional costs. Therefore, the repeated motifs problem can be solved in time $\mathcal{O}(n^2)$.

Example 4. *Let the input string $s :=$ GGGTCTA, $e := 1$, $q := 2$, and $k := 3$. Table 2 illustrates matrix M. Array v*

is:

j :	0	1	2	3	4	5	6	7
$v[j]$:	0	0	0	2	2	1	1	1

and so the strictly valid motifs are $s[1\,..\,3]$ = `GGG` *and* $s[2\,..\,4]$ = `GGT`.

The common motifs problem (detecting strictly valid motifs) can be directly solved by solving the fixed-length approximate string matching problem for all N^2 pairs of the N input strings. Consider, for example, the common motifs problem under the Hamming distance model. We use an array u_r for each input string s_r, such that for all $1 \le r \le N$, $k \le j \le |s_r|$, $u_r[j]$ contains the total number of strings in which motif $s_r[j-k+1\,..\,j]$ e-occurs; we set $u_r[j] := 0$, for all $0 \le j < k$. Array u_r, for all $1 \le r \le N$, can easily be computed, by computing matrix M for pair (s_r, s_t), for all $1 \le t \le N$. While computing matrix M, we increment $u_r[j]$ only once *iff* $\mathsf{M}[i,j] \le e$, for some $k \le i \le |s_t|$; as soon as we have computed the N different matrices M for s_r, it suffices to iterate over array u_r and report $s_r[j-k+1\,..\,j]$, for all $k \le j \le |s_r|$, as a strictly valid motif *iff* $u_r[j] \ge q$. An array v_r, such that $v_r[j]$, for all $1 \le j \le |s_r|$, denoting the total number of e-occurrences of motif $s_r[j-k+1\,..\,j]$ in $s_1, \ldots, s_N$ can also be maintained, in analogy to the repeated motifs case. Maintaining arrays u_r and v_r does not induce additional costs. Therefore, the common motifs problem can be solved in time $\mathcal{O}(n^2)$ per matrix, where n is the average length of the N strings, thus $\mathcal{O}(N^2n^2)$ in total.

Example 5. *Let the input strings* s_r := `GTGAACT`, s_t := `GTCACGT`, $e := 1$, $q := 2$, *and* $k := 3$. *Further, let the current state of arrays* u_r *and* v_r *be:*

j :	0	1	2	3	4	5	6	7
$u_r[j]$:	0	0	0	0	1	0	1	0
$v_r[j]$:	0	0	0	0	2	0	2	0

Table 3 illustrates matrix M. *Arrays* v_r *and* u_r *are:*

j :	0	1	2	3	4	5	6	7
$u_r[j]$:	0	0	0	1	2	0	2	1
$v_r[j]$:	0	0	0	1	3	0	3	1

and so the strictly valid motifs are $s_r[2\,..\,4]$ = `TGA` *and* $s_r[4\,..\,6]$ = `AAC`.

4. IMPLEMENTATION

We implemented `MoTeX` as a programme that solves the repeated motifs problem and the common motifs problem for strictly valid motifs. A valid motif is strictly valid *iff* it occurs exactly, at least once, in (any of) the input string(s). `MoTeX` was implemented in the C programming language under Linux. It is distributed under the GNU General Public License (GPL). The open-source code and documentation are available at `http://www.exelixis-lab.org/motex`. The mandatory input parameters are:

- a file with the input string(s) in FASTA format (sequences);
- the distance (edit/Hamming) to be used for extracting the motifs;
- the length k for the motifs;
- the maximal allowed distance e;
- the quorum $q' \le 100$ (%) as the proportion of sequences in the input dataset in which a reported motif must e-occur at least once.

There are additional input parameters. For example, an integer parameter *occ*, such that the total number of e-occurrences of a reported motif in (any of) the input sequence(s) is at least *occ*.

Given these parameters, `MoTeX` outputs a text file containing the strictly valid motifs. For each reported motif, it also outputs the total number of sequences in which the motif e-occurs at least once and the total number of its e-occurrences. To ensure that the output of `MoTeX` is linear with respect to the size of the input dataset, we need to report each distinct motif only once. This is done by inserting a strictly valid motif into a *trie* data structure [11]—a deterministic finite automaton—only if it does not already exist in it. The time needed to traverse the trie from the root to the leaf does not depend on the trie size. It is only proportional to the motif length. Trivially, `MoTeX` reports a motif only if it does not already exist in the trie, thus, avoiding duplicates.

Apart from the standard CPU version, `MoTeX` comes in two HPC flavors: the OpenMP-based version for shared memory systems and the MPI-based version for distributed memory systems. The user can choose the best-suited version depending on:

- the total size of the input sequence(s);
- the nature of the input dataset, for instance, one (or a few) very long sequence(s), or many relatively short sequences;
- the available HPC architecture;
- the number $p > 1$ of available processors.

In our implementation, we distinguish among two cases: $p \le N$ and $p > N$, where N is the number of input sequences.

4.1 Case 1: $p \le N$

This is the common motifs problem, that is, we have a large number of relatively short sequences. The user can choose any of the two HPC versions. In this case, `MoTeX` evenly distributes the computation of the N^2 distinct DP matrices for the N input sequences in a straightforward manner across the p processors. Therefore, if $p \le N$, the common motifs problem for strictly valid motifs can be solved in parallel in time $\mathcal{O}(N^2n^2/p)$.

4.2 Case 2: $p > N$

This case refers to either the repeated motifs problem with one very long sequence or the common motifs problem with just a few very long sequences. The user is advised to chose the MPI-based version. Because there are fewer embarrassingly parallel tasks than for Case 1 above, we need to exploit fine-grain parallelism in the DP calculation. For this purpose, `MoTeX` deploys an efficient cost-optimal parallelization scheme [16, 6] for fixed-length approximate string-matching. In the following, we describe this parallelization scheme for computing matrix D in more detail.

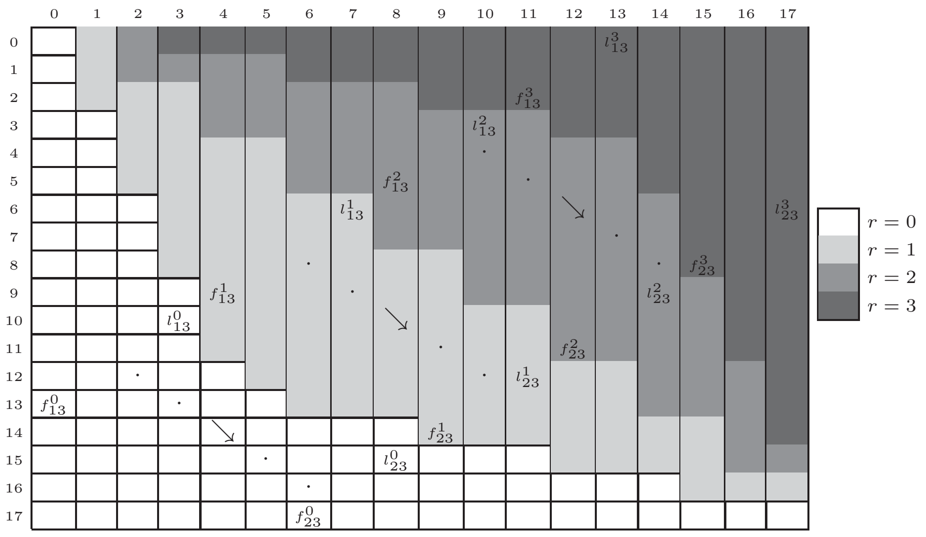


Figure 1: Mapping of the cells of matrix D to the available processors for $n := m := 17$ and $p := 4$

Decomposition technique.

We use the *functional decomposition* technique. The initial focus is on the actual computation, rather than on the data manipulated by this computation. Consider, for example, the repeated motifs problem under the edit distance model. The main observation for parallelisation is that cell $\mathsf{D}[i, j]$ is computed by using cells $\mathsf{D}[i-1, j]$, $\mathsf{D}[i-1, j-1]$, and $\mathsf{D}[i, j-1]$ [14]. Based on this, we can partition the problem of computing matrix $\mathsf{D}[0\,..\,n, 0\,..\,m]$ into a set $\{\Delta_0, \Delta_1, \ldots, \Delta_{n+m}\}$ of anti-diagonal arrays. This is also known as a sequence of parallel *wavefronts* of a computation. As shown in Equation 1, the computation of each cell in each diagonal (parallel step) is independent from the other cells of the same diagonal, and can thus be executed concurrently. The size of Δ_ν, that is, the number of cells of Δ_ν, is denoted by δ_ν.

$$\Delta_\nu[x] = \begin{cases} \mathsf{D}[\nu - x, x] : 0 \le x \le \nu & \text{(a)} \\ \mathsf{D}[\nu - x, x] : 0 \le x < m+1 & \text{(b)} \\ \mathsf{D}[n - x, \nu - n + x] : & \\ 0 \le x < n + m - \nu + 1 & \text{(c)} \end{cases} \quad (1)$$

where

(a) if $0 \le \nu < m$
(b) if $m \le \nu < n$
(c) if $n \le \nu < n + m + 1$

Mapping technique.

Regarding the mapping of tasks to the available processors, we distribute the computation of the cells for each diagonal among the available processors. Without loss of generality, we assume that for each diagonal array Δ_ν, each processor r, for $0 \le r < p$, is assigned δ_ν/p contiguous cells of Δ_ν. The mapping is described by Equation 2.

$$\begin{array}{ll} f^r_\nu = r\lceil \delta_\nu/p \rceil & \\ l^r_\nu = f^r_\nu + \lceil \delta_\nu/p \rceil - 1 & \text{if } r < \delta_\nu \ mod\ p \\ & \\ f^r_\nu = r\lfloor \delta_\nu/p \rfloor + \delta_\nu \ mod\ p & \\ l^r_\nu = f^r_\nu + \lfloor \delta_\nu/p \rfloor - 1 & \text{otherwise} \end{array} \quad (2)$$

It relies on calculating both the indices f^r_ν and l^r_ν of the first and the last cell of Δ_ν mapped to processor r, respectively. In other words, processor r is assigned the computation of $\Delta_\nu[f^r_\nu\,..\,l^r_\nu]$ (see Fig. 1 for $\nu = 13$ and $\nu = 23$). The auxiliary space required is $\mathcal{O}(m/p)$, as each diagonal only depends on the two immediately preceding diagonals.

Processor communication.

The data exchange between the processors involves a constant number of point-to-point message transfers (neighboring cells) at each parallel step [13].

Since the number of cells to be computed is $\Theta(nm)$, the number of parallel steps is $\Theta(n)$—exactly $n+m+1$ diagonals—and each processor is assigned the computation of $\mathcal{O}(m/p)$ cells—the size of each diagonal is $\mathcal{O}(m)$—at each parallel step. Therefore, if $p > N$, the repeated motifs problem for strictly valid motifs can be solved in parallel time $\mathcal{O}(n^2/p)$. The common motifs problem for strictly valid motifs can be solved in parallel time $\mathcal{O}(N^2n^2/p)$.

5. STATISTICAL SIGNIFICANCE

On the one hand, word-based methods guarantee global optimality; on the other hand, they suffer from the problem of producing too many spurious motifs. Consider, for instance, the case when $k := 12$ and $e := 3$. When a valid motif is expanded by one symbol, it becomes a valid motif for $k := 13$ and $e := 4$, as all its e-occurrences are also valid e-occurrences of the longer one. Thus, we need a method for sorting the reported motifs according to their statistical significance.

For this purpose we use the z-score, a statistical measure of significance, that compares the actual number of occurrences of a motif with the expected value:

$$\mathcal{Z}_{\text{score}} = \frac{Occ_x - nN\pi(x,e)}{\sqrt{nN\pi(x,e)(1-\pi(x,e))}}$$

where Occ_x is the observed number of occurrences of motif x, $\pi(x,e)$ is the probability that x occurs in a sequence of the dataset with at most e mismatches, and nN is the total accumulated length of the sequences in the input dataset.

SMILE [18] is a post-analysis programme that, given the output of a motif extractor, calculates the z-score and other statistical measures for assessing the statistical significance of the reported motifs. The significance of the reported motifs is computed from their occurrence frequency in a random subset of the input data. Finally, SMILE sorts the motifs by their z-scores in descending order. `MoTeX` can produce a SMILE-compatible output file, which can then directly be used as input for SMILE.

6. EXPERIMENTAL RESULTS

All experiments were conducted on a Linux-based Infiniband-connected cluster using 1 up to 1056 2.4 GHz AMD 6136 processors.

Although `MoTeX` is based on an exact and deterministic algorithm, we initially evaluated its accuracy. The reasons for doing this are twofold: first, to ensure that our implementation is correct; and, second, to evaluate the impact of our stricter motif validity assumption.

As basic dataset, we used 1200 1000 bp-long upstream sequences of *Homo sapiens* genes with a total size of 1.2MB, retrieved from the ENSEMBL [8] database. We extracted synthetic datasets, denoted by $< \alpha, \beta_{\min}, \beta_{\max}, \gamma >$, from this basic dataset as follows. We initially generated 100 random DNA motifs, each of length α; then, we implanted each motif into a randomly selected fraction γ % of the sequences in the basic dataset, by inserting a random number of mismatches, ranging from $\beta_{\min} := 0$ to $\beta_{\max}$, in each different occurrence of the motif. We executed `MoTeX` with dataset $< \alpha, \beta_{\min}, \beta_{\max}, \gamma >$ as input and also set $k := \alpha$, $e := \beta_{\max}$, and $q' := \gamma$ as input arguments. Finally, we counted how many out of the 100 motifs we had initially implanted were found. We used different values for α, $\beta_{\max}$, and γ. The results in Table 4 demonstrate the high accuracy of `MoTeX`. It was able to identify all implanted motifs.

In order to evaluate the accuracy of `MoTeX` under more difficult conditions, we repeated the same set of experiments by setting $\beta_{\min} := 1$. In other words, we made sure that, for each randomly generated motif, no exact occurrence was implanted in any sequence. Again, the results in Table 5 demonstrate the high accuracy of `MoTeX`, which was able to identify all implanted motifs.

We also make available, on the website of `MoTeX`, the programme and the basic dataset used to generate the aforementioned synthetic datasets (Table 4 and Table 5) for reproducing the results on `MoTeX` accuracy.

To further evaluate the accuracy of `MoTeX` under statistical measures of significance, we compared its output to the corresponding output of RISOTTO using SMILE (see Section 5 for details). The reason for choosing RISOTTO for this comparison is that RISOTTO is based on an exact and deterministic algorithm and guarantees global optimality. As input dataset for the two programmes, we used the 1200 upstream sequences of *Homo sapiens* genes. The results obtained by the two programmes using SMILE were similar and partially even identical. The highest scoring motifs for $k := 8$, $e := 1$, $q' := 10$ and $k := 10$, $e := 2$, $q' := 10$ are given in Table 6 and Table 7, respectively. Note that, SMILE reports the same set of highest scoring motifs with the two programmes in both cases. The small differences observed in Table 7 between the rankings of the highest scoring motifs reported for the two programmes are due to the randomization in SMILE.

We also evaluated the efficiency of `MoTeX` in comparison to Weeder (version 1.4.2) and RISOTTO. We did not include YMF in this comparison because of the input parameter restrictions; the maximum allowed length k for the motifs is 8. We compared the standard CPU version and the OpenMP-based version of `MoTeX` against Weeder and RISOTTO for

Table 4: Number of identified motifs by `MoTeX` for $\beta_{\min} := 0$

Dataset	Implanted motifs	Identified implanted motifs	Extracted motifs
$< 8, 0, 2, 1 >$	100	100	64,712
$< 8, 0, 2, 10 >$	100	100	64,789
$< 8, 0, 2, 25 >$	100	100	64,712
$< 8, 0, 2, 50 >$	100	100	64,728
$< 10, 0, 3, 1 >$	100	100	540,930
$< 10, 0, 3, 10 >$	100	100	545,980
$< 10, 0, 3, 25 >$	100	100	544,412
$< 10, 0, 3, 50 >$	100	100	541,965
$< 12, 0, 4, 1 >$	100	100	1,013,738
$< 12, 0, 4, 10 >$	100	100	1,029,950
$< 12, 0, 4, 25 >$	100	100	1,023,758
$< 12, 0, 4, 50 >$	100	100	1,018,555

Each of the datasets consists of 1200 upstream sequences of *Homo sapiens* genes.

Table 5: Number of identified motifs by `MoTeX` for $\beta_{\min} := 1$

Dataset	Implanted motifs	Identified implanted motifs	Extracted motifs
$< 8, 1, 2, 1 >$	100	100	64,724
$< 8, 1, 2, 10 >$	100	100	64,834
$< 8, 1, 2, 25 >$	100	100	64,745
$< 8, 1, 2, 50 >$	100	100	64,717
$< 10, 1, 3, 1 >$	100	100	540,730
$< 10, 1, 3, 10 >$	100	100	545,851
$< 10, 1, 3, 25 >$	100	100	544,516
$< 10, 1, 3, 50 >$	100	100	542,168
$< 12, 1, 4, 1 >$	100	100	1,018,579
$< 12, 1, 4, 10 >$	100	100	1,029,801
$< 12, 1, 4, 25 >$	100	100	1,023,429
$< 12, 1, 4, 50 >$	100	100	1,018,280

Each of the datasets consists of 1200 upstream sequences of *Homo sapiens* genes.

Table 6: Highest scoring motifs for $k := 8$, $e := 1$, and $q' := 10$ extracted from 1200 upstream sequences of *Homo sapiens* genes

Motif	$\mathcal{Z}_{\text{score}}$
`AAAAATAA`	31.96
`TATTTATA`	30.70
`TTATTTTT`	27.86
`TTTATTTT`	26.92
`TATAAATA`	26.06
`TATTTTTA`	25.90
`TTATATTT`	25.07

(a) RISOTTO

Motif	$\mathcal{Z}_{\text{score}}$
`AAAAATAA`	31.96
`TATTTATA`	30.70
`TTATTTTT`	27.86
`TTTATTTT`	26.92
`TATAAATA`	26.06
`TATTTTTA`	25.90
`TTATATTT`	25.07

(b) `MoTeX`

Table 7: Highest scoring motifs for $k := 10$, $e := 2$, and $q' := 10$ extracted from 1200 upstream sequences of *Homo sapiens* genes

Motif	$\mathcal{Z}_{\text{score}}$
`TATTTTTATA`	36.91
`AAATAAATAT`	35.07
`AATAAATATT`	33.16
`TTTATATTTT`	32.38
`AAATATATAT`	32.35
`TATTTTTATT`	31.61
`AAAATAAACA`	31.55

(a) RISOTTO

Motif	$\mathcal{Z}_{\text{score}}$
`AAATAAATAT`	34.93
`TATTTTTATA`	32.96
`TTTATATTTT`	32.39
`AACATATAAA`	32.26
`AAAATAAACA`	31.86
`AATAAATATT`	31.75
`TATTTTTATT`	31.71

(b) `MoTeX`

the common motifs problem. As input sequences, we used 1062 upstream sequences of *Bacillus subtilis* genes of total size 240KB, obtained from the RISOTTO website [20]. We measured the elapsed time for each programme for different combinations of input parameters. In particular, we provided different values for the motif length k, the maximal allowed Hamming distance e, and the quorum q', as a proportion of sequences in the input dataset. As depicted in Fig. 2, the performance of `MoTeX` is independent of the aforementioned input parameters and corroborates our theoretical findings. The standard CPU version of `MoTeX` is

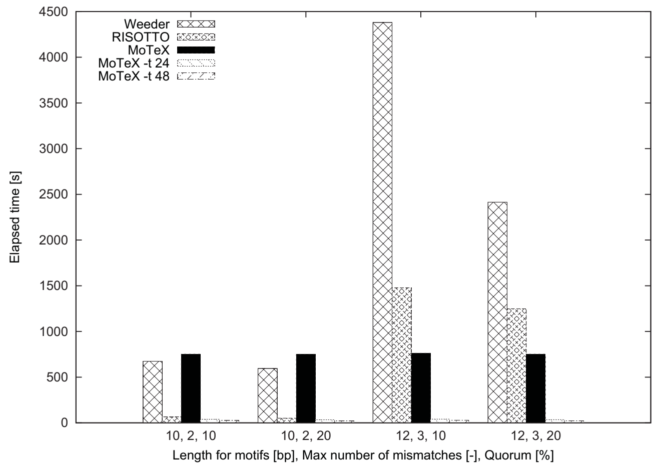


Figure 2: Elapsed-time comparison of Weeder, RISOTTO, and `MoTeX` for the common motifs problem using 1062 upstream sequences of *Bacillus subtilis* genes

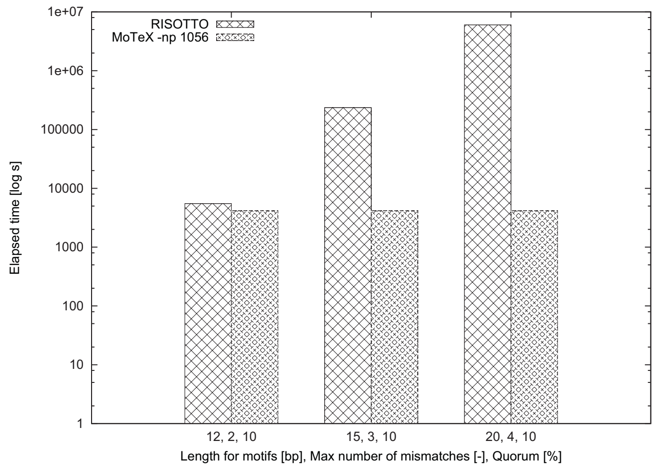
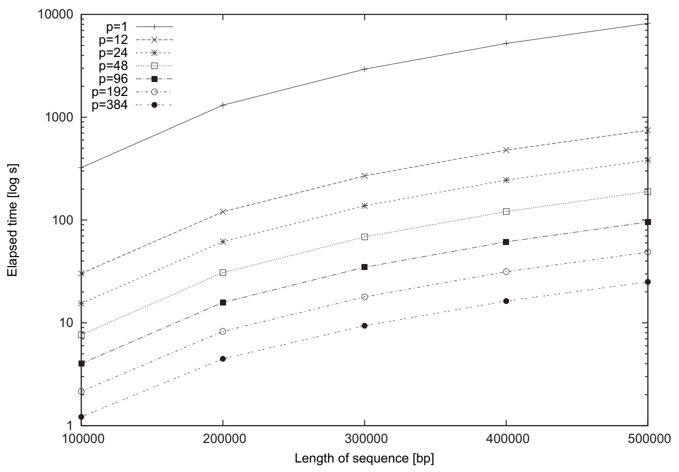


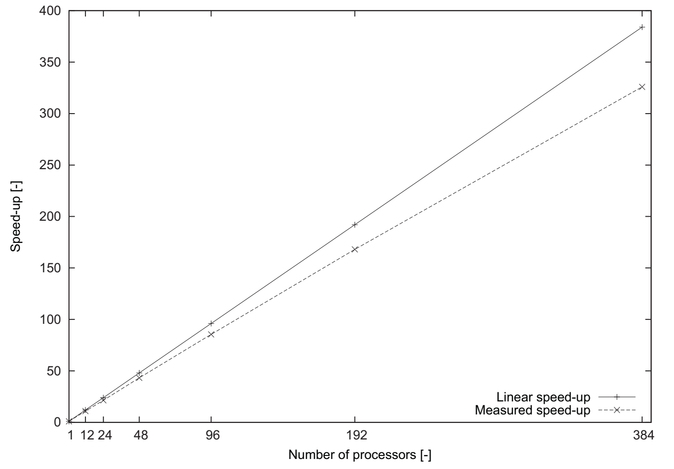
Figure 3: Elapsed-time comparison of RISOTTO and `MoTeX` for the common motifs problem using the full upstream *Homo sapiens* genes dataset

competitive for short motifs and becomes the fastest as the length k for the motifs and the maximal allowed Hamming distance e increase. As expected, the OpenMP-based version of `MoTeX` with 48 processing threads (`-t 48`) is always the fastest.

As a next test, we compared the MPI-based version of `MoTeX` against RISOTTO. We did not include Weeder in this comparison since the maximum allowed length k for the motifs in Weeder is 12. As input sequences, we used the full upstream *Homo sapiens* genes dataset obtained from ENSEMBL [8]. We measured the elapsed time for each programme for different combinations of input parameters. Although a direct comparison between the MPI-based version of `MoTeX` and RISOTTO is unfair, we believe that it is critical as it highlights the fact that real full-length datasets cannot be processed by current state-of-the-art tools for motif extraction in a reasonable amount of time. In other words,



(a) Elapsed time of MoTeX



(b) Measured speed-up of MoTeX for $n := 500$KB

Figure 4: Elapsed time and measured speed-up of MoTeX for the repeated motifs problem using sequences obtained from *Escherichia coli*

the time-to-solution *is* an important property. As depicted in Fig. 3, the performance of MoTeX is completely independent of the aforementioned input parameters. The MPI-based version of MoTeX with 1056 processors (`-np 1056`) finishes the assignment in a reasonable amount of time (69 mins), as opposed to RISOTTO, which requires more than two months.

As a last test, we evaluated the parallel efficiency of the MPI-based version of MoTeX, for the repeated motifs problem under the edit distance model. We used DNA sequences of the single chromosome of *Escherichia coli* str. K-12 substr. MG1655, obtained from GenBank [12] as input. Experimental results regarding the elapsed time of MoTeX for different problem sizes are provided in Fig. 4a. The measured relative speed-up of MoTeX is illustrated in Fig. 4b. The relative speed-up was calculated as the ratio of the runtime of the parallel algorithm on 1 processor to the runtime of the parallel algorithm on p processors—the runtime of the parallel algorithm on 1 processor was identical to the sequential one. The results highlight the *good* scalability of MoTeX, even for small problem sizes. When increasing the problem size, MoTeX achieves linear speed-ups, confirming our theoretical findings.

7. FINAL REMARKS

In this article, we introduced MoTeX, the first word-based HPC tool for MoTif eXtraction from large-scale datasets. MoTeX considers the repeated motifs problem and the common motifs problem for strictly valid motifs. A valid motif is strictly valid *iff* it occurs exactly, at least once, in (any of) the input sequence(s). By making this stricter assumption for motif validity, we showed how both the repeated motifs problem and the common motifs problem can be solved directly and efficiently using algorithms for fixed-length approximate string matching.

The runtime of MoTeX does not depend on (i) the length for motifs, (ii) the size of the alphabet, or (iii) the maximal allowed distance. Given the stricter assumption on motif validity, it is guaranteed to find globally optimal solutions. Furthermore, the size of the output is linear with respect to the size of the input. In addition, it can identify motifs under the edit distance model or the Hamming distance model. Finally, MoTeX also comes in two HPC flavors: the OpenMP-based version and the MPI-based version.

Suffix-tree-based motif extractors produce globally optimal solutions but exhibit many disadvantages. We demonstrated that MoTeX can alleviate these shortcomings for motif extraction from large-scale datasets. For instance, we showed how the quadratic time complexity of MoTeX can be slashed, in theory and in practice, by using parallel computations. The extensive experimental results presented are promising, both in terms of accuracy under statistical measures of significance as well as efficiency; a fact that suggests that further research and development of MoTeX is desirable.

The stricter assumption for motif validity trivially suggests that MoTeX is not the best solution in the case where the input dataset is rather small. Consider, for instance, the case when Σ is the DNA alphabet and $k := 10$. If the total size of the input sequences is only a few KB, MoTeX might not be able to extract all statistically significant motifs, since some of them might be valid but not strictly valid. The best solution, in this case, will most probably be an exhaustive enumeration.

Our main goal is to accurately detect motifs over a massive set of biological sequences. We are especially interested in discovering transcription factor binding sites whose conservation is decreasing as the evolutionary distance between species increases. We plan to employ our motif extraction algorithm in a phylogenetic framework to incorporate evolutionary information in the motif extraction process. In addition, extending MoTeX to be able to extract not only simple but also structured motifs is already under way.

8. REFERENCES

[1] P. Antoniou, M. Crochemore, C. S. Iliopoulos, and P. Peterlongo. Application of suffix trees for the acquisition of common motifs with gaps in a set of strings. In *Proceedings of the 1st International Conference on Language and Automata Theory and Applications (LATA 2007)*, volume Report 35/07, pages 57–66. Research Group on Mathematical

Linguistics, Universitat Rovira i Virgili, Tarragona, 2007.

[2] P. Antoniou, J. Holub, C. S. Iliopoulos, B. Melichar, and P. Peterlongo. Finding common motifs with gaps using finite automata. In *Implementation and Application of Automata, 11th International Conference, CIAA 2006, Taiwan*, volume 4094 of *Lecture Notes in Computer Science*, pages 69–77. Springer, 2006.

[3] P. Bucher. Weight matrix descriptions of four eukaryotic RNA polymerase II promoter elements derived from 502 unrelated promoter sequences. *Journal of Molecular Biology*, 212(4):563–578, 1990.

[4] A. Carvalho, L. Marsan, N. Pisanti, and M.-F. Sagot. RISOTTO: Fast extraction of motifs with mismatches. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06)*, Lecture Notes in Computer Science, pages 757–768. Springer, 2006.

[5] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

[6] M. Crochemore, C. S. Iliopoulos, and S. P. Pissis. A parallel algorithm for fixed-length approximate string-matching with k-mismatches. In T. Elomaa, H. Mannila, and P. Orponen, editors, *Algorithms and Applications*, volume 6060 of *Lecture Notes in Computer Science*, pages 92–101. Springer, 2010.

[7] M. Das and H. K. Dai. A survey of DNA motif finding algorithms. *BMC Bioinformatics*, 8(Suppl 7):S21+, 2007.

[8] Ensembl Genome Browser. `www.ensembl.org`, Apr. 2013.

[9] E. Eskin and P. A. Pevzner. Finding composite regulatory patterns in DNA sequences. In *ISMB*, pages 354–363, 2002.

[10] A. Floratou, S. Tata, and J. Patel. Efficient and accurate discovery of patterns in sequence data sets. *Knowledge and Data Engineering, IEEE Transactions on*, 23(8):1154–1168, 2011.

[11] E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, 1960.

[12] GenBank. `http://www.ncbi.nlm.nih.gov/genbank/`, Apr. 2013.

[13] C. Hadjinikolis, C. S. Iliopoulos, S. P. Pissis, and A. Stamatakis. Minimising processor communication in parallel approximate string matching. Technical Report Exelixis-RRDR-2012-8, Heidelberg Institute for Theoretical Studies, 2012.

[14] C. Iliopoulos, L. Mouchard, and Y. Pinzon. The Max-Shift algorithm for approximate string matching. In G. Brodal, D. Frigioni, and A. Marchetti-Spaccamela, editors, *Proceedings of the fifth International Workshop on Algorithm Engineering (WAE 2001)*, volume 2141 of *Lecture Notes in Computer Science*, pages 13–25, Denmark, 2001. Springer.

[15] C. S. Iliopoulos, J. A. M. McHugh, P. Peterlongo, N. Pisanti, W. Rytter, and M.-F. Sagot. A first approach to finding common motifs with gaps. *Int. J. Found. Comput. Sci.*, 16(6):1145–1154, 2005.

[16] C. S. Iliopoulos, L. Mouchard, and S. P. Pissis. A parallel algorithm for the fixed-length approximate string matching problem for high throughput sequencing technologies. In B. Chapman, F. Desprez, G. R. Joubert, A. Lichnewsky, F. Peters, and T. Priol, editors, *Proceedings of the International Conference on Parallel Computing (PARCO 2009)*, volume 19 of *Advances in Parallel Computing*, pages 150–157, France, 2010. IOS Press.

[17] M. Lothaire, editor. *Applied Combinatorics on Words*. Cambridge University Press, 2005.

[18] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of computational biology : a journal of computational molecular cell biology*, 7(3-4):345–362, 2000.

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

[20] RISOTTO motif discovery tool (Supplementary material). `http://www.lx.it.pt/~asmc/software/risotto.html`, Apr. 2013.

[21] S. Rombauts, P. Déhais, M. Van Montagu, and P. Rouzé. PlantCARE, a plant cis-acting regulatory element database. *Nucleic acids research*, 27(1):295–296, 1999.

[22] M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics (LATIN'98)*, pages 374–390, London, UK, 1998. Springer-Verlag.

[23] T. D. Schneider and R. M. Stephens. Sequence logos: a new way to display consensus sequences. *Nucleic Acids Res*, 18(20):6097–6100, 1990.

[24] S. Sinha and M. Tompa. YMF: A program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Res*, 31(13):3586–3588, 2003.

[25] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nat Biotechnol*, 23(1):137–144, 2005.

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

[27] J. Vilo, A. Brazma, I. Jonassen, A. J. Robinson, and E. Ukkonen. Mining for putative regulatory elements in the yeast genome using gene expression data. In P. E. Bourne, M. Gribskov, R. B. Altman, N. Jensen, D. A. Hope, T. Lengauer, J. C. Mitchell, E. D. Scheeff, C. Smith, S. Strande, and H. Weissig, editors, *ISMB*, pages 384–394. AAAI, 2000.