*AMO - Advanced Modeling and Optimization, Volume* 11*, Number* 2*,* 2009

# Finding Motifs Based on Suffix Trie

F. Zare-Mirakabad[a], P. Davoodi[b], H. Ahrabian[a,b,1], A. Nowzari-Dalini[a,b], M. Sadeghi[c,d], B. Goliaei[a]

[a]*Department of Bioinformatics, Institute of Biochemistry and Biophysics, University of Tehran, Tehran, Iran.*
Email: {*zare, goliaei*}*@ibb.ut.ac.ir.*

[b]*Center of Excellence in Biomathematics, School of Mathematics, Statistics, and Computer Science, University of Tehran, Tehran, Iran.*
Email: {*ahrabian,davoodi,nowzari*}*@ut.ac.ir.*

[c]*National Institute of Genetic Engineering and Biotechnology, Tehran, Iran.*
Email: *sadeghi@nrcgeb.ac.ir.*

[d]*School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics (IPM), Tehran, Iran.*

**Abstract**

Pattern discovery or motif finding is one of the most challenging problems in both molecular biology and computer science. In this paper we present an exact exhaustive method, for finding motifs of length $\ell$ in a set of $t$ sequences of length $n$ with a limited number of mutations $d$. The algorithm is based on the Depth First Search on a suffix trie with maximum nodes $O(tn)$ and is performed in $O(t^2n^2\ell^2)$ time complexity. The proposed algorithm is tested on yeast and human transcription factor binding site data sets and the obtained results are compared to the other well-known algorithms. The experimental results demonstrate that the proposed method is working analogous to them algorithms.

**Keywords***:* Bioinformatics, Motif finding, Suffix tree, Suffix trie.

[1]Corresponding author.

**AMO - Advanced Modeling and Optimization. ISSN: 1841-4311.**

# 1 Introduction

Problem of pattern discovery appears in different areas of biology. Patterns that we want to find usually correspond to functionally or structurally important elements in proteins or DNA sequences. There is an assumption that the important regions are better conserved in the evolution and therefore they occur more frequently than expected [Pavesi et al., 2001]. These regions are called *patterns* or *motifs* and problem of finding them is called motif (or pattern) finding problem. Pattern finding can be used in multiple sequence alignment, protein structure and function prediction, categorization of protein families, promoter signal detection and family DNA sequences.

In a simple form, the motif finding problem can be formulated as follows [Eskin and Pevzner, 2002]. Given the sample sequence $S$, find all $\ell$-mers occurring with up to $d$ mismatches (each $\ell$-mer have at most $d$ mismatches with another $\ell$-mer), at least $k$ times in the sample $S$. Such $\ell$-mers are called $(\ell, d)$-$k$ patterns or motifs. A variant of this problem assumes that the sample $S$ is split into several sequences i.e. consider $S = \{s_1, s_2, \ldots, s_t\}$, we want to find all $\ell$-mers that occur with up to $d$ mismatches in at least $k$ sequences in the sample set $S$ [Eskin and Pevzner, 2002]. In this problem a sequence is a string on a given alphabet $\Sigma$; thus $\Sigma = \{A, C, G, T\}$ for DNA sequences, $\Sigma = \{A, C, G, U\}$ for RNA sequences, and $\Sigma$ is the set of all 20 amino acids for a protein. Most of the algorithms can be easily adapted to work with any finite alphabet, and can be used also outside bioinformatics, or an other type of biological data. There are two common models for motif representation: String (or consensus) representation and Position Frequency Matrix (PFM) (or profile) representation [Stormo, 2000]. String representation uses an $\ell$-mer sequence of symbols (or nucleotides) A, C, G and T to describe a motif. PFM representation contains the probability for each nucleotide at each position.

Despite many studies, motif finding in unaligned DNA sequences is far from being solved, most motifs in DNA sequences are so complicated that we do not have good models or reliable algorithms for their recognitions. The first motif in DNA was found in 1970 by Hamilton Smith after the discovery of HindII restriction enzyme [Eskin and Pevzner, 2000]. Finding the palindromic site of this motif was not a simple problem in 1970. Since then, many algorithms are presented for solving motif or pattern finding problem in general case [Rigoutsos and Floratos, 1998; Stormo, 2000; van Helden et al., 2000; GuhaThakurta and Stormo, 2001; Fogel et al., 2004; Wei, and Jensen, 2006; Sandve and Drabløs, 2006; Das and Dai, 2007; Ausiello et al., 2008; Klepper et al., 2008 ].

In order to obtain a list of candidate motifs, one can apply the classical pattern finding algorithms based on approaches such as the Pattern Driven Approach (PDA) or Sample Driven Approach (SDA) [Eskin and Pevzner, 2002; Pavesi et al., 2001]]. Probably the best method for finding short $\ell$-mer motifs is the PDA [Brazma et al., 1998] that tests all $4^\ell$ $\ell$-mer patterns of fixed length $\ell$ in lexical order, compares each pattern to every $\ell$-mer in the sample, and scores each pattern by the number of approximate occurrences in the sample (or by a more involved function) and finds the high scoring patterns and returns all $(\ell, d)$-$k$ patterns [Eskin and Pevzner, 2002; Staden, 1989; Wolferstetter et al., 1996; van Helden et al., 1998]. However, an exhaustive search through all $4^\ell$ $\ell$-mer patterns

becomes impractical for $\ell > 10$. To bypass the problem of excessive time requirements in PDA, Waterman et al. [1984] suggested an algorithm that significantly reduces the time requirements of the pattern driven approach, called SDA. They noticed the most of $4^\ell$ patterns examined in the PDA are not worth examining, since neither these patterns nor their neighbors appear in the sample, and a significant speed up can be achieved by eliminating these patterns from the search. This approach is used in different heuristic methods to prune the search space, either by searching only for a subset of possible patterns, for example, those that occur exactly in the sequences at least once as in [Gelfand et al., 2000] or by imposing restrictions on the location of mismatches along the pattern, as in [Brazma et al., 1998; Califano, 2000] where mismatches can occur only at fixed positions [Eskin and Pevzner, 2002].

Thus, in order to discover longer and subtler motifs, in the most widely used algorithms the PDA has been abandoned, and the motif is extracted by analyzing and comparing the patterns occurring in the sequences, i.e. the SDA is used [Lawrence et al., 1993; Baily and Elkan, 1995; Hertz and stormo, 1999; Pavesi et al., 2001; Buhler and Topma 2002; Stine et al., 2003; Gertz et al., 2007].

In this paper we present a simple and efficient algorithm MotifST (Motif finding using Suffix Trie) for pattern discovery in DNA sequences or more generally any sequences with an arbitrary alphabet. The algorithm MotifST uses a tree similar to suffix trie with a fixed depth $\ell$ (the length of the motif) and by a Depth First Search (DFS) on this tree, patterns with $d$ mismatches are detected. Previously, suffix tree is used for motif finding in [Marsan and Sagot, 2000; Pavesi et al. 2001]. In our algorithm we use suffix trie instead suffix tree for motif finding. The time complexity of our algorithm is $O(t^2n^2\ell^2)$ and the space complexity of the algorithm is $O(tn)$. We performed our algorithm on the real data sets and the obtained results are compared with the result of well-known motif finding tools Weeder [Pavesi et al., 2001], MotifSampler [Thijs et al., 2001], MEME [Bailey and Elkan, 1995], YMF [Sinha and Tompa, 2003], oligodyad-analysis [van Helden et al., 1998], ANN-Spec [Workman and Stormo, 2000], Consensus [Hertz and Stormo, 1999], MITRA [Eskin and Pevzner, 2002], Improbizer [Ao et al., 2004], SeSiMCMC [Favorov et al., 2005], GLAM [Frith et al., 2004], QuickScore [Egnier and Denise 2004], and AlignACE [Hughes et al., 2000] to demonstrate the effectiveness of our proposed method. Note that the reason for choosing these tools for comparison is due to their availability and the accuracy of their results in comparison with the other existing tools so far.

# 2 Pattern Finding Algorithm

Giving a set of $t$ sequences $S = \{s_1, s_2, \ldots, s_t\}$ on the alphabet $\Sigma = \{A, C, G, T\}$ such that each $s_i = s_i[1], \ldots, s_i[n]$ for $1 \leq i \leq t$, and $n$ denotes the length of sequence $s_i$, it is desired to find all $(\ell, d)$-$k$ patterns (or motifs), that are patterns of length $\ell \in [\ell_{min}, \ell_{max}]$ with at most $d$ mismatches in at least $k$ sequences of the set $S$, where $d = \varepsilon \times \ell$ and $\varepsilon \in [0, 1]$ is a mismatch rate.

As mentioned, in PDA for finding motif in the set $S$ we should search all the search

space which is equal to $|\Sigma|^{\ell}$ ($\ell \in [\ell_{min}, \ell_{max}]$) in the set $S$ which is an exponential space. For this reason we should reduce the search space, and therefore we assume that motif is exactly occurred in at least one sequence of $S$ (without mismatch), i.e. we use SDA and the search space $|\Sigma|^{\ell}$ for $\ell \in [\ell_{min}, \ell_{max}]$ is reduced to all the subsequences of length $\ell$ where at least one of them is exactly appeared in one sequence.

We use the term *motif instance* as a subsequence of length $\ell$ in each sequence of the set $S$ with at most $d$ mismatches with original motif. The number of matches between two sequences $x$ and $y$ of length $\ell$ is defined as:

$$H(x, y) = \sum_{i=1}^{\ell} h(x[i], y[i]),$$

where

$$h(a, b) = \begin{cases} 1 & \text{if } a = b, \\ 0 & \text{otherwise.} \end{cases}$$

For example, for the given original motif $e = AGATT$ of length $\ell = 5$, if we have the following sequences

$$\begin{aligned} s_1 &= \text{gggagtcccgtAGCTTaggcctcgg} \\ s_2 &= \text{aACAATaccaacgcaactctagggc} \\ s_3 &= \text{aaaccccaaACATAaaacgccgcta} \\ s_4 &= \text{cttaccatcGCATAcgaggacagaa} \end{aligned}$$

then the *motif instance* for $e$ in $s_1$ with 1 mismatch is $AGCTT$, in $s_2$ with 2 mismatches is $ACAAT$, in $s_3$ with 2 mismatches is $ACATA$, and in $s_4$ with 3 mismatches is $GCATA$.

The set $U = \{u_1, \ldots, u_t\}$ is called *motif instance set* where each $u_i$ is a *motif instance* in $s_i$ for the original motif. The pattern shared by the subsequences $u_i$ ($1 \leq i \leq t$) of the *motif instance set* $U = \{u_1, \ldots, u_t\}$ is dubbed a *consensus* or a *consensus motif* and shown by $\delta$. Obviously, for the sequences and *motif instances* of the mentioned example the *motif instance set* is equal to $U = \{u_1, u_2, u_3, u_4\}$ where $u_1 = AGCTT$, $u_2 = ACAAT$, $u_3 = ACATA$, and $u_4 = GCATA$ and their corresponding *consensus* is equal to $\delta = ACATA$. For a given *motif instance set* easily we can obtain the Position Frequency Matrix (PFM) representation. As mentioned, PFM representation contains the probability for each nucleotid at each position *i.e.* the value of $j$-th column in each row gives us the probability of occupying the $j$-th position of the motif by symbols A, C, G or T. For example, the PFM corresponding to the above *motif instance set* $U = \{u_1, u_2, u_3, u_4\}$ is given below.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| A | 3/4 | 0 | 3/4 | 1/4 | 2/4 |
| C | 0 | 3/4 | 1/4 | 0 | 0 |
| G | 1/4 | 1/4 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 3/4 | 2/4 |

Note that in motif finding problem the original motif is unknown and we desire to find. For this reason, we use potential motif. A potential motif is a subsequence of length $\ell$ in a sequence $s_i$ of $S$ such that at least $k$ subsequence in $s_j$ ($1 \leq j \leq t$, $i \neq j$) have at most $d$ mismatches with it. In fact, these subsequences are *motif instances* of this potential motif. For these *motif instances* we can obtain *motif instance set*. Therefore for finding motifs, first we find all potential motifs in $S$ and all *motif instance sets* of these potential motifs are created. Later between all *motif instance sets* the best *motif instance set* is selected. The selection criteria can be different in various motif finding algorithms. Eventually the *consensus motif* of the selected *motif instance set* is considered as an original motif.

In order to get a simple and efficient way of sequence comparison, our algorithm explores these type of patterns by employing suffix trie. This algorithm is composed of four steps:

1. Constructing the suffix trie.
2. Searching for the pattern.
3. Enumeration of pattern.
4. Scoring and motif selection.

The detail descriptions of these steps are given below.

## 2.1 Constructing the suffix trie

In the first step, a tree similar to the suffix trie for all the given subsequences of $S$ is constructed.

A suffix trie is a data structure that exposes the interval structure of a string in a very deep meaningful way. A suffix trie $\tau$ for a string $x = x_1, \ldots, x_n$ is a rooted directed tree with exactly $n$ leaves numbered from 1 to $n$. This tree is a special case of a suffix tree [Ukkonen, 1995, Pavesi et al., 2001]. In suffix trie, each edge is labeled with non empty character of $x$. Two edges leaving the same node can not have labels beginning with the same character. For any leaf $i$, the concatenation of the edge labels on the path from the root to leaf $i$ exactly spells the suffix of $x$ starting at position $i$, that is, it spells out $x_i, \ldots, x_n$. The same structure can be built also for the set of $t$ sequences. The easiest way is to append an array of length $t$, to each leaf, so to distinguish which sequence of a suffix belong to, and to add the sequences to the tree one by one. In other word, it is possible to annotate each leaf $j$ of tree with an array of length $t$, where $i$-th entry of this array is set to $r$ if the word spelled by the path from root ending at the leave $j$ occurs in position $r$ in $i$-th sequence. Clearly, if more than one entry is set, the subsequence is occurred in more than one sequence. Also, if the subsequence is occurred more than one position, the corresponding entry should be an array of positions. The construction of the structure requires $O(N)$ time and $O(N)$ space [Ukkonen, 1995], where $N$ is the overall length of the sequences, while annotating it with the array takes additional $O(tN)$ time and $O(tN)$ space.

Conventionally the depth of a suffix trie corresponding to a sequence is equal to the length of the corresponding sequence but here the depth of constructed suffix trie is equal to the maximum length of the pattern ($\ell_{max}$) which we intend to find. We call this, pseudo suffix trie. In this pseudo suffix trie each node is labeled with a character of input sequences, and each node is connected to its children and shows a subsequence of the input sequences, by concatenation the characters in the path beginning from the root through to the corresponding node. Also the position of each subsequence in given sequence is denoted in each node. The maximum number of leaves in this tree is equal to $t(n - \ell_{max} + 1)$. For example a suffix trie for the sequences ACCA and CCAAG with length $\ell_{max} = 3$ is shown in Figure 1; the subsequence AAG is occurred in position 3 of the second, therefore the second entry of the array corresponding to the first leave is set to 3.

Given the annotated pseudo suffix trie of depth $\ell_{max}$, our algorithm searches for the potential motif $p$ of length $\ell \in [\ell_{min}, \ell_{max}]$ using DFS on this tree. The *motif instances* of potential motif $p$ is considered in two cases: exact *motif instances* and approximate *motif instances*. In the first case, $u$ is a *motif instance* of $p$ such that $H(p, u) = 0$ and $u$ is a subsequence in some position $j$ in some sequence $s_i$, and in the second case, $H(p, u) \leq 0$. For this reason two different lists called *self-postvector* and *mis-postvector* are considered for each node; *self-postvector* keeps the positions of exact potential motifs and *mis-postvector* keeps the positions of both approximate and exact potential motifs. Since we find subsequences of length $\ell$, therefore for all the nodes these two arrays do not need to exist, and just for nodes in depth $\ell \in [\ell_{min}, \ell_{max}]$ should be considered *i.e.,* the nodes in depth less than $\ell_{min}$ do not need such arrays. The algorithm given in Algorithm 1 illustrates the construction of this tree. It should be noted that initially we should create a dummy node as a *root* of tree and the algorithm is called with parameters *root*, $S$, $\ell_{min}$, and $\ell_{max}$. In this step, simultaneously with construction of pseudo suffix trie, the position of exact potential motifs are also calculated and added to the arrays *self-postvector* and

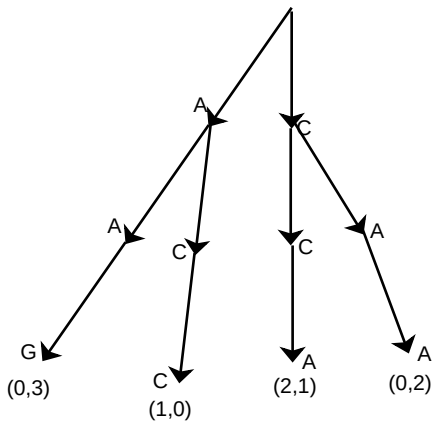


Figure 1: Suffix trie for sequences ACCA and CCAAG with length 3.

**Algorithm 1** Constructing the pseudo suffix trie algorithm.

```
1: Procedure ConsTree (root, S, ℓ_min, ℓ_max)
2: Var ℓ, i, j, r : Integer ; b : Character ; v : TreeNode ;
3: Begin
4:     For ℓ := ℓ_min To ℓ_max Do Begin
5:       For i := 1 To t Do Begin
6:         For j := 1 To |s_i| − ℓ Do Begin
7:           v = root ;
8:           For r := j To j + ℓ − 1 Do Begin
9:             c = s_i[r] ;
10:            If One of children of v has label b Then
11:              v = child v with label b ;
12:            Else Begin
13:              create a child for v and label it with b ;
14:              v = child v with label b;
15:            End ;
16:          End ;
17:          self-postvector[v][i] =self-postvector[v][i] ∪ j ;
18:          mis-postvector[v][i] =mis-postvector[v][i] ∪ j ;
19:        End ;
20:      End ;
21:    End ;
22: End ;
```

*mis-postvector*. In the next step (Step 2), the position of approximate potential motifs are set.

## 2.2 Searching for the pattern

As mentioned, in this step the position of approximate potential motifs with desirable mismatches are investigated and assigned to *mis-posvector*. The process is done by the DFS traversal on the pseudo suffix trie. By getting to a node $v$ in the depth $\ell \in [\ell_{min}, \ell_{max}]$ in DFS traversal, the subsequence in the path from root to $v$ is considered and checked as a candidate for a potential motifs. This step is shown in Algorithm 2. At first call of this algorithm, parameter $v$ is equal to the root of tree (a global pointer to the root of tree) and *depth* is equal to zero and $\ell$ is equal to $\ell_{max}$. In this algorithm, the algorithm *CheckPattern* is called for finding *motif instances* of each candidate potential motif $p$. This algorithm is given in Algorithm 3. The algorithm *CheckPattern* finds the position of all subsequences of length $\ell$ that can be constructed from root to each node in level $\ell$ with $d$ mismatches from $p$, and theses positions are interested into the *mis-posvector*. In this algorithm a potential motif $p$ is matched along with the different subsequences on the tree, starting from the root, and keeps track of the number of mismatches between $p$ and

**Algorithm 2** Potential motif searching algorithm.

```
1: Procedure FindPotentialMotif(v, depth, ℓ)
2: Var  p : String ; y : TreeNode ;
3: Begin
4:       if (depth < ℓ) Then Begin
5:          For each child y of node v Do
6:             FindPotentialMotif(y, depth + 1, ℓ) ;
7:       End ;
8:       p = A subsequence from root to v ;
9:       CheckPattern(v, root, 0, ε, 0, ℓ, p);
10:      CheckPattern(v, root, 0, ε, 0, ℓ, p^rc);
11: End ;
```

**Algorithm 3** The checking pattern algorithm.

```
1: Procedure CheckPattern(v, u, d, ε, depth, ℓ, p)
2: Var  i, j, r : Integer ; b : Character ; y : TreeNode ;
3: Begin
4:       If (u ≠ v) Then Begin
5:          If (d < ε × depth) or (d = 0) Then Begin
6:             If depth < ℓ Then Begin
7:                b = p[depth] ;
8:                For each child y of node u Do Begin
9:                   If (label(y) = b) Then
10:                     CheckPattern(v, y, d, ε, depth + 1, ℓ, p); ;
11:                  Else
12:                     CheckPattern(v, y, d + 1, ε, depth + 1, ℓ, p); ;
13:               End ;
14:            End
15:            Else Begin
16:               mis-postvector[v] = mis-postvector[v] ∪ self-postvector[u] ;
17:               mis-postvector[u] = mis-postvector[u] ∪ self-postvector[v] ;
18:            End ;
19:         End ;
20:      End ;
21: End ;
```

each subsequences encountered at each path. Whenever the number of mismatches on a path is greater than $d$, the subsequence is discarded. If our search completes the whole sequence $p$, the surviving paths represent all the *motif instances* of $p$ in the sequence with at most $d$ mismatches. The subsequence $p$ appears in this method are given by the union of the entry corresponding to different paths. As shown in Algorithm 2, in order to find

the reverse complement of $p$ in tree, *CheckPattern* is recalled for reverse complement of $p$ ($p^{rc}$).

## 2.3 Enumeration of pattern

At the end of Step 2, *mis-postvector* keeps the position of the exact and approximate *motif instances*. In fact the array *mis-postvector* corresponding to each leaf of a pseudo suffix trie shows the start position of *motif instances* and the sequence in which *motif instances* is occurred. Now, the number of sequences including the occurred subsequences are enumerated. If this number was equal or greater than $k$, a *motif instance set* is created from these subsequences (*motif instances*) and is added to the set of candidate *motif instance sets* $M$. Finally, $M = \{U_1, \ldots, U_m\}$ contains $m$ *motif instance sets*, such that each $U_i$ is a *motif instance set* and contains at least $k$ *motif instances*, i.e. $U_i = \{u_{i,1}, \ldots, u_{i,q}\}$ where each $u_{i,j}$ is a *motif instance* with length $\ell \in [\ell_{min}, \ell_{max}]$, and $k \leq q \leq t$.

## 2.4 Scoring and motif selection

All the patterns obtained in the previous step are scored in this step. Assume the set $M = \{U_1, U_2, \ldots, U_m\}$ includes all *motif instance sets* and is obtained in Step 2. The scoring schema for scoring these *motif instance sets* are based on information content (IC), this means that each *motif instance set* is scored separately with regard to the background (sequence set $S$). Thus, for the *motif instance set* $U_i = \{u_{i,1}, \ldots, u_{i,q}\}$, the corresponding position frequency matrix $W_i$ is constructed. Later the IC value corresponding to $W_i$ is computed as follows:

$$IC(W_i) = \sum_{\alpha \in \{A,C,G,T\}} \sum_{j=1}^{\ell} W_i[\alpha, j] \log(\frac{W_i[\alpha, j]}{\omega[\alpha]}).$$

where $W_i[\alpha, j]$ is equal to the occurrence probability of character $\alpha \in \{A, C, G, T\}$ in $j$-th position of the *motif instance set* $U_i$, and $\omega[\alpha]$ is the occurrence probability of character $\alpha$ in the set $S$.

All the obtained *motif instance sets* are sorted based on this score. Finally the top 10 *consensus motifs* between the obtained *motif instance sets* are investigated for reporting as motifs.

## 2.5 Final Algorithm

Regarding the above steps, the final algorithm MotifST is shown in Algorithm 4. In this algorithm the variables *mis-postvector* and *self-postvector* are global variables, as mentioned, and keep the position of potential motifs.

The procedure *ConsTree* is performed in $O(tn\ell)$ and the procedure *FindPattern* using a DFS traversal on suffix trie for finding pattern is performed in $O(tn\ell)$ and since this process is performed for each word of length $\ell$, therefore the total time complexity of

**Algorithm 4** The final MotifST algorithm.

1: **Procedure** *MotifST(S, $\ell_{min}$, $\ell_{max}$)*
2: **Var** $\ell$ : **Integer** ; *mis-postvector, self-postvector*: Array of Integer ;
3: *root* : **TreeNode** ;
4: **Begin**
5: Create a dummy node *root* ;
6: ConsTree(*root*, $S$, $\ell_{min}$, $\ell_{max}$) ;
7: **For** $\ell := \ell_{min}$ **To** $\ell_{max}$ **Do**
8: FindPattern(*root*, 0, $\ell$) ;
9: Create a set of *motif instance sets* $M = \{U_1, \ldots, U_m\}$ from *mis-postvector* ;
10: Sort the set $M = \{U_1, \ldots, U_m\}$ based on IC ;
11: Select top 10 *consensus motif* of *motif instance set* $M$ as motifs ;
12: **End** ;

the algorithm is $O(t^2n^2\ell^2)$. Finally, the total time complexity of Algorithm MotifST is $O(t^2n^2\ell^2)$. For space complexity, the tree is constructed in $O(tn)$ space, and integer array corresponding to each leave consumes $O(tn)$ space, therefore the total space of this algorithm is $O(tn)$.

# 3 Experimental results

Our algorithm is tested in similar method used in [Tompa et al., 2005], we employ the data sets addressed in [Tompa et al., 2005] to test our algorithm. These data sets contain a well-known promoter database of 10 yeast, 8 fly, 12 mouse, and 12 human and their corresponding their motifs. From these data sets, 10 of yeast and 6 of human data sets are selected for test. This selection is done because of the reported results in [Tompa et al., 2005] on these data sets, show that the most algorithms get better results on these data sets. The obtained results by MotifST are compared with the well-known programs Weeder [Pavesi et al., 2001], MotifSampler [Thijs et al., 2001], MEME [Bailey and Elkan, 1995], YMF [Sinha and Tompa, 2003b], oligodyad-analysis [van Helden et al., 1998], ANN-Spec [Workman and Stormo, 2000], Consensus [Hertz and Stormo, 1999], MITRA [Eskin and Pevzner, 2002], Improbizer [Ao et al., 2004], SeSiMCMC [Favorov et al., 2005], GLAM [Frith et al., 2004], QuickScore [Egnier and Denise 2004], and AlignACE [Hughes et al., 2000]. As mentioned, the reason for choosing these tools for comparison is because of their availability and the accuracy of their results in comparison with the other existing tools so far.

The comparison of these algorithms are performed based on the following comparison measurements: nucleotide Performance Coefficient (nPC), Site Sensitivity (sSn), Site Positive Prediction (sPP), and Site Average Performance (sAP) [Tompa et al., 2005]. The definition of these measurements are given in Table 1. In this table, the variables $nTP$ denotes the number of nucleotide positions in both known sites and the predicted sites, $nFP$ shows the number of nucleotide positions not in known sites but in the predicted sites,

Table 1: Comparison measurement formulas.

| *Measurement* | *Formula* |
|---|---|
| nPC | $\frac{nTP}{(nTP+nFP+nFN)}$ |
| sSn | $\frac{sTP}{sTP+sFN}$ |
| sPP | $\frac{sTP}{sTP+sFP}$ |
| sAP | $\frac{sSn+sPP}{2}$ |

$nFN$ defines the number of nucleotide positions in known sites but not in the predicted sites, $nTN$ denotes the number of nucleotide positions in neither known sites nor the predicted sites, $sTP$ defines the number of known sites overlapped by the predicted sites, $sFP$ shows the number of predicted sites not overlapped by known sites, and $sFN$ denotes the number of known sites not overlapped by the predicted sites [Benitez-Bellon et al., 2002]. By regarding the $nPC$ formula in Table 1, we have $nPC \leq 1$ and the higher value of $nPC$ shows that the known sites and the predicted sites are more similar. Obviously, if the predicted sites were equal to the known sites then $nPC$ is equal to one. Also, $sAP$ shows the accuracy of the algorithm in site level, and is equal or less than 1. All of these measurements are previously defined in [Tompa et al., 2005] as suitable measurements for comparison of motif finding tools.

For comparing the obtained results by our algorithm with other well-known algorithm, we have employed the *Assessment of Computational Motif Discovery Tools* also introduced by Tompa et al. [2005] and addressed in [http://bio.cs.washington.edu/assessment]. This tool gets the results of each algorithm as input and produces the values of the above measurements for comparison. We need a way for summarizing the performance of a given motif finding program over all data sets. Recall from [Tompa et al., 2005], one method for summarizing is *Combined* method. In this method, first we calculate $nTP$, $nFP$, $nFN$, $nTN$, $sTP$, $sFP$ and $sFN$ over all data sets, then we add up these values, and finally all data sets are considered as a large data set and the nPC and sAP measures are computed from these values.

As mentioned before, the algorithm for finding $(\ell, d)$-$k$ pattern (or motif) gets the parameters $\ell_{min}$, $\ell_{max}$, $d$, and $k$ as input parameters. For comparing the algorithm MotifST with other pattern finding tools we let $\ell_{min} = 6$, $\ell_{max} = 15$, $\varepsilon = 0.25$ and $k = 3$. The algorithm achieved best results with these parameters.

Because of the repeated subsequences in the DNA sequences, motifs might not be selected correctly. For this reason, prior to the execution of the program, the repeated subsequences are deleted in the DNA sequences. The *Tandom Repeats Finder* software is used for deleting the repetitive subsequences [Benson, 1999]. This software employs masking method for deleting repeated subsequences. After deleting the repeated subsequences in 16 data sets (10 from yeast and 6 from human data sets), MotifST algorithm is performed on them, with the above mentioned parameters. All the obtained motifs by this algorithm, are sorted based on the information content. Finally, all motifs which are similar

or nearly similar to actual motifs reported in [http://bio.cs.washington.edu/assessment] are selected as a motifs. Therefore, in these 16 data sets, in 7 cases the first motif, in 4 cases the second motif, in 3 cases the third motif and in two cases the sixth motif are selected as the final motifs.

As mentioned, for comparing the obtained results by MotifST algorithm with other algorithms, the *Assessment of Computational Motif Discovery Tools* (ACMD) introduced by Tompa et al. [2005], addressed in [http://bio.cs.washington.edu/assessment] are used, i.e. the motifs obtained in the above 16 data sets by our algorithm and the other well-known algorithms are provided as input parameters for this tool. This tool computes statistical measures introduced in Table 1 for these motifs.

In Table2 the results of ACMD tool for the motifs of 6 human data sets obtained by Weeder, MotifST, MotifSampler, MEME, YMF and oligodyad-analysis are denoted. The other programs achieved not good results on this data set and their results are not shown. Based on the Combined method, the $nPC$ measurement of the obtained results and $sAP$ measurement of the obtained results are shown in Figure 2 and Figure 3 respectively.

In Table 3 the results of ACMD tool for motifs of 10 yeast data sets obtained by MotifST, Weeder, MotifSampler, MEME, YMF, AlignACE, oligodyad-analysis, ANN-Spec, Consensus MITRA, Improbizer, SeSiMCMC, GLAM, and QuickScore are denoted. Based on Combined method, the $nPC$ measurement of obtained results and $sAP$ measurement of obtained results are shown in Figure 4 and Figure 5 respectively.

By the definition of $nPC$ value, higher value of $nPC$ shows the more number of $TP$ and less number of $FN$ and $FP$. Actually, the algorithm with higher $nPC$ value shows a better performance. Also, by the definition of $sAP$ value, higher value of $sAP$ shows more number of $TP$ and less number of $FN$ and $FP$ in site level. Therefore, the algorithm with higher $sAP$ value shows a better accuracy. By analyzing the results shown in Figures 2 and 3, we can see for Human data sets, Weeder has obtained best results and the second best algorithm is our algorithm that provides the second best results. For Yeast data set, as shown in Figures 4 and 5, MotifSampler and Weeder show better results than our algorithm but our algorithm performs better than the other tools. Therefore, experimentally we see that for yeast and human data sets, MotifST is as good as the other tools and can

Table 2: The evaluated measurement values for Human data sets.

| Tools | nPC | nCC | sSn | sPP | sAP |
|---|---|---|---|---|---|
| Weeder | 0.319809 | 0.489695 | 0.596154 | 0.659575 | 0.627864 |
| MotiST | 0.212581 | 0.346444 | 0.365385 | 0.51514 | 0.439449 |
| Oligidyad-analysis | 0.203008 | 0.365561 | 0.307692 | 0.470588 | 0.38914 |
| YMF | 0.190998 | 0.340687 | 0.307692 | 0.390244 | 0.348968 |
| MEME | 0.112646 | 0.197526 | 0.153846 | 0.3037692 | 0.230769 |
| MotifSampler | 0.012546 | -0.003912 | 0.038462 | 0.025317 | 0.031889 |

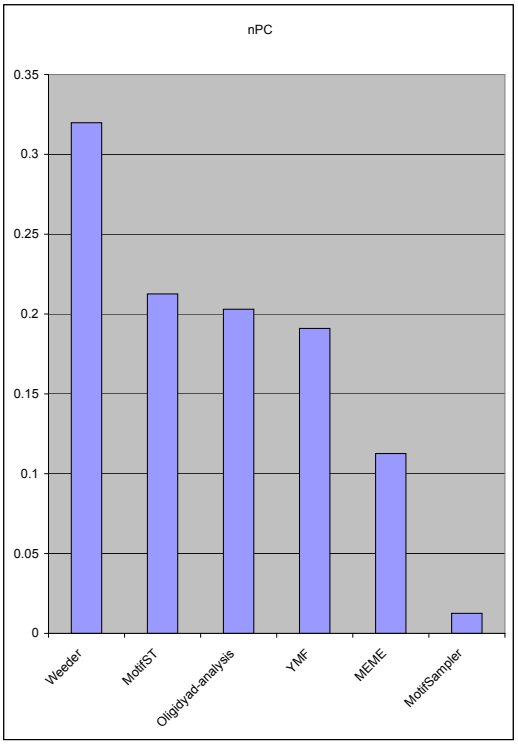


Figure 2: The *nPC* measure evaluated by combined method for the results obtained from the different algorithms on Human data sets.

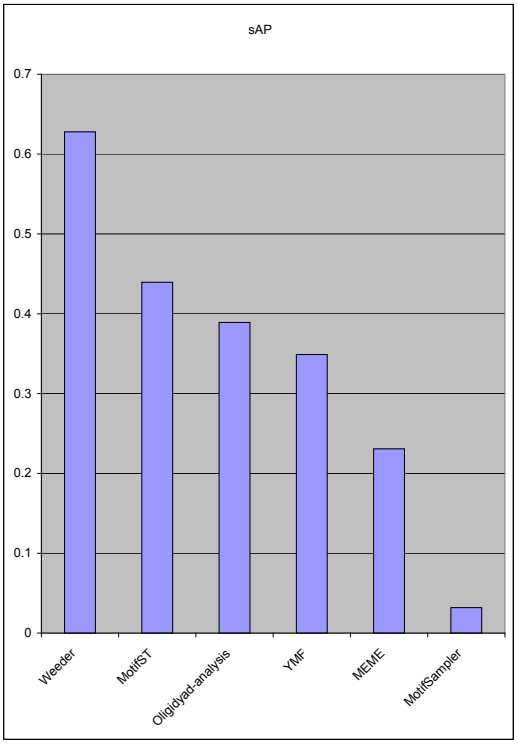


Figure 3: The *sAP* measure evaluated by combined method for the results obtained from the different algorithms on Human data sets.

Table 3: The evaluated measurement values for yeast data sets.

| Tools | nPC | nCC | sSn | sPP | sAP |
|---|---|---|---|---|---|
| Weeder | 0.2330536 | 0.3863337 | 0.52 | 0.549258 | 0.5346479 |
| MotifSampler | 0.2045307 | 0.3501753 | 0.3866667 | 0.4915254 | 0.439096 |
| UTMotif | 0.1554878 | 0.2693545 | 0.3648649 | 0.3913043 | 0.3780846 |
| MEME | 0.140914 | 0.2381559 | 0.32 | 0.3037975 | 0.3118987 |
| YMF | 0.1115288 | 0.2076962 | 0.28 | 0.3387097 | 0.3093548 |
| AlignACE | 0.1020954 | 0.1684257 | 0.28 | 0.2019231 | 0.2409615 |
| ANN-Spec | 0.0820595 | 0.1331542 | 0.3066667 | 0.1411043 | 0.2238855 |
| Oligidyad-analysis | 0.0760795 | 0.1639418 | 0.1866667 | 0.3043478 | 0.2455072 |
| MITRA | 0.0686717 | 0.1148955 | 0.16 | 0.1538462 | 0.1569231 |
| Improbizer | 0.0629461 | 0.0988463 | 0.2666667 | 0.1333333 | 0.2 |
| Consensus | 0.0602706 | 0.1149074 | 0.1466667 | 0.2391304 | 0.1928986 |
| SeSiMCMC | 0.0378673 | 0.0504601 | 0.0933333 | 0.0721649 | 0.0827491 |
| GLAM | 0.0332075 | 0.0442325 | 0.1466667 | 0.0789475 | 0.1022807 |
| QuickScore | 0.0294249 | 0.0391636 | 0.12 | 0.0434783 | 0.0817391 |

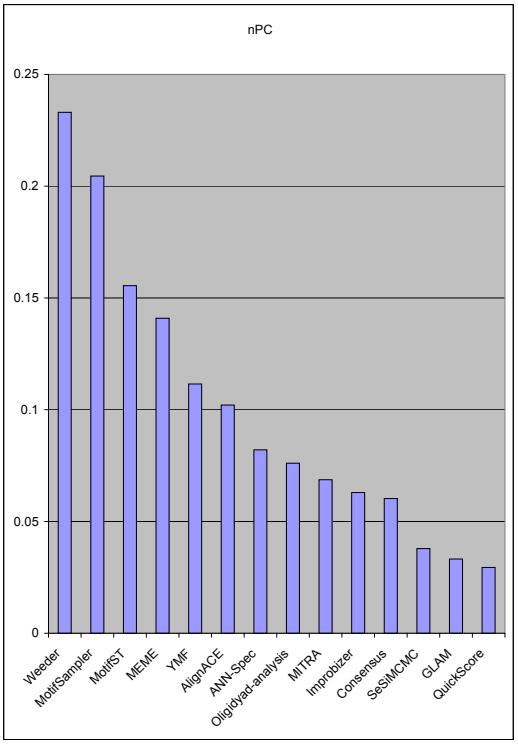


Figure 4: The *nPC* measure evaluated by combined method for the results obtained from the different algorithms on Yeast data sets.

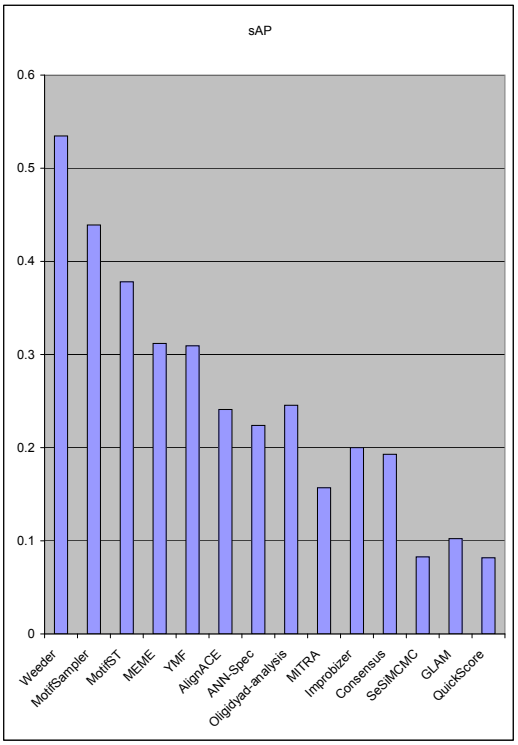


Figure 5: The *sAP* measure evaluated by combined method for the results obtained from the different algorithms on Yeast data sets.

be used as an standard tool for motif finding.

# 4 Conclusion

In this article, we have proposed a new approach for finding motifs in biological data or other data. We developed a novel motif-finding algorithm MotifST that detects motifs based on the suffix trie. The method is implemented and tested on real data sets. The obtained results are compared with other well-known motif finding algorithms. For comparison two measurements *nPC* and *sAP* are considered. In experiments on real biological data sets, Tompa's motif assessment benchmarks, we observed that MotifST performs as well as the other existing motif finding tools and therefore it can be used as a standard motif finding tool.

## Acknowledgements

This research was partially supported by University of Tehran.

## REFERENCES

**Ao, W., Gaudet, J., Kent, W.J., Muttumu, S., and Mango, S. E.** (2004) Environmentally induced foregut remodeling by PHA-4/FoxA and DAF-12/NHR. *Science,* Vol. **305**, pp.1743–1746.

**Ausiello, G., Gherardini, P.F., Marcatili, P., Tramontano, A., Via, A., and Helmer-Citterich, M.** (2008) FunClust: a web server for the identification of structural motifs in a set of non-homologous protein structures. *BMC Bioinformatics,* Vol. **9**, pp.S2.

**Bailey, T., and Elkan, C.** (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning,* Vol. **21**, pp.51–80.

**Benitez-Bellon, E., Moreno-Hagelsieb, G., and Collado-Vides J.** (2002) Evaluation of thresholds for the detection of binding sites for regulatory proteins in Escherichia coli K12 DNA. *Genome Biology,* Vol. **3**, pp.1–16.

**Benson, G.** (1999) Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Res.,* Vol. **27**, pp.573–580.

**Brazma, A., Jonassen, I., Vilo, J., and Esko Ukkonen, E.** (1998) Predicting gene regulatory elements in Silico on a genomic scale. *Genome Research,* Vol. **8**, pp.1202–1215.

**Buhler, J., and Tompa, M.** (2003) Finding motifs using random projections. *J. Comput. Biol.,* Vol. **9**, pp.225–242.

**Califano, A.** (2000) Splash: structural pattern localization analysis by sequential histogram. *Bioinformatics,* Vol. **16**, pp.341–357.

**Das, M.K., and Dai, H.K.** (2007) A survey of DNA motif finding algorithms. *BMC Bioinformatics,* Vol. **8**, pp.S21.

**Egnier, M.R., and Denise, A.** (2004) Rare events and conditional events on random strings. *Discret Math. Theor. Comput. Sci.,* Vol. **6**, pp.191–214.

**Eskin, E., and Pevzner, P.** (2002) Finding composite regulatory patterns in DNA sequences. *Bioinformatics,* Vol. **18**, pp.354–363.

**Favorov, A.V., Gelfand, M.S., Gerasimova, A.V., Ravcheev, D.A., Mironov, A.A., and Makeev, V.J.** (2005) A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length. *Bioinformatics,* Vol. **21**, pp.2240–2245.

**Fogel, G., Weekes, D., Varga, G., Dow, E., Harlow, H., Onyia, J., and Su, C.** (2004) Discovery of sequence motifs related to co-expression of genes using evolutionary computation. *Nucleic Acids Res.,* Vol. **32**, pp.3826–3835.

**Frith, M.C., Hansen, U., Spouge, J.L., and Weng, Z.** (2004) Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res.,* Vol. **32**, pp.189–200.

**Gelfand, M., Koonin, E., and Mironov, A.** (2000) Prediction of transcription regulatory sites in archaea by a comparative genomic approach. *Nucleic Acids Res.,* Vol. **28**, pp.695–705.

**Gertz, J., Riles, L., Turnbaugh, P., Ho, S., and Cohen, B.** (2007) Discovery, validation, and genetic dissection of transcription factor binding sites by comparative and functional genomics. *Genome Res.,* Vol. **15**, pp.1145–1152.

**GuhaThakurta, D., and Stormo, G.** (2001) Identifying target sites for cooperatively binding factors. *Bioinformatics,* Vol. **17**, pp.608–621.

**Hertz, G., and Stormo, G.** (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics,* Vol. **15**, pp.563–577.

**Hughes, J., Estep, P., Tavazoie, S., and Church, G.** (2000) Computational identification of cis-regulatory elements associated with functionally coherent groups of genes in Saccharomyces cerevisiae. *J. Mol. Biology,* Vol. **296**, pp.1205–1214.

**Klepper, K., Sandve, G.K., Abul, O., Johansen, J., and Drablos, F.** (2008) Assessment of composite motif discovery methods. *BMC Bioinformatics,* Vol. **9**, pp.S123.

**Lawrence, C., Altschul, S., Bogusky, M., Liu, J., Neuwald, A., and Wootton, J.** (1993) Detecting subtle sequence signals, Gibbs sampling strategy for multiple alignment. *Science,* Vol. **262**, pp.208–214.

**Marsan, L., and Sagot, M.** (2000) Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *J. Comput. Biol.,* Vol. **7**, pp.345–360.

**Pavesi, G., Mauri, G., and Pesole, G.** (2001) An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics,* Vol. **17**, pp.207–214.

**Pevzner, P.** (2000) *Computational Molecular Biology: An Algorithmic Approach.* The MIT Press, Massachusetts.

**Rigoutsos, I., and Floratos, A.** (1998) Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics,* Vol. **14**, pp.55–67.

**Sandve, G.K., and Drabløs, F.** (2006) A survey of motif discovery methods in an integrated framework. *Biol. Direct,* Vol. **1**, pp.11.

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

**Staden, R.** (1998) Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in Biosciences,* Vol. **5**, pp.293–289.

**Stine, M., Dasgupta, D., and Mukatira, S.** (2003) Motif discovery in upstream sequences of coordinately expressed genes. *Evol. Comput.,* Vol. **3**, pp.1596–1603.

**Stormo, G.** (2000) DNA binding sites: representation and discovery. *Bioinformatics,* Vol. **16**, pp.16–23.

**Thijs, G., Lescot, M., Marchal, K., Rombauts, S., De Moore, B., Rouze, P., and Moreau, Y.** (2001) A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. *Bioinformatics,* Vol. **17**, pp.1113–1122.

**Tompa, M., Li, N., Bailey, T., Church, G., De Moor, B., Eskin, E., Favorov, A., Frith, M., Fu, Y., Kent, W., et al.** (2005) Assessing computational tools for the discovery of transcription factor binding sites. *Nat. Biotechnol.,* Vol. **23**, pp.137–144.

**van Helden, J., Andre, B., and Collado-Vides, J.** (1998) Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J. Mol. Biol.,* Vol. **281**, pp.827–842.

**van Helden, J., Rios, A., and Collado-Vides, J.** (2000) Discovery regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Res.,* Vol. **28**, pp.1808–1818.

**Ukkonen, E.** (1995) On line construction of suffix trees. *Algorithmica,* Vol. **14**, pp.249–260.

**Waterman, M.S., Arratia, R. and Galas, D.J.** (1984) Pattern recognition in several sequences; consensus and alignment. *Bull. Math. Biol.,* Vol. **46**, pp.515–527.

**Wei, Z., and Jensen, S.** (2006) GAME: detecting cis-regulatory elements using a genetic algorithm. *Bioinformatics,* Vol. **22**, pp.1577–1584.

**Wijaya, E., Rajaraman, K., Yiu, S., and Sung, W.** (2007) Detection of generic spaced motifs using submotif pattern mining. *Bioinformatics,* Vol. **23**, pp.1476–1485.

**Wolferstetter, F., Frech, K., Herrmann, G., and Werner, T.** (1996) Identification of functional elements in unaligned nucleic acids sequences by a novel tuple search algorithm. *Computer Applications in Biosciences,* Vol. **12**, pp.71–80.

**Workman, C.T., and Stormo, G.D.** (2000) ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. *Pac. Symp. Biocomput.,* Vol. **5**, pp.464–475.