BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 19 2011, pages 2641–2647
doi:10.1093/bioinformatics/btr459

Sequence analysis

Advance Access publication August 5, 2011

Tree-structured algorithm for long weak motif discovery

He Quan Sun[1,*], Malcolm Yoke Hean Low[1], Wen Jing Hsu[1], Ching Wai Tan[1] and Jagath C. Rajapakse[1,2,3]

[1]Department of Computer Science, School of Computer Engineering, Nanyang Technological University, Singapore 639798, [2]Singapore-MIT Alliance, Nanyang Technological University, Singapore 637460 and [3]Department of Biological Engineering, Massachusetts Institute of Technology, MA 02142, USA

Associate Editor: Alex Bateman

ABSTRACT

Motivation: Motifs in DNA sequences often appear in degenerate form, so there has been an increased interest in computational algorithms for weak motif discovery. Probabilistic algorithms are unable to detect weak motifs while exact methods have been able to detect only short weak motifs. This article proposes an exact tree-based motif detection (TreeMotif) algorithm capable of discovering longer and weaker motifs than by the existing methods.

Results: TreeMotif converts the graphical representation of motifs into a tree-structured representation in which a tree that branches with nodes from every sequence represents motif instances. The method of tree construction is novel to motif discovery based on graphical representation. TreeMotif is more efficient and scalable in handling longer and weaker motifs than the existing algorithms in terms of accuracy and execution time. The performances of TreeMotif were demonstrated on synthetic data as well as on real biological data.

Availability: https://sites.google.com/site/shqssw/treemotif

Contact: sunh0013@e.ntu.edu.sg

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on March 30, 2011; revised on July 31, 2011; accepted on August 1, 2011

1 INTRODUCTION

Regulatory regions such as promoters, enhancers, locus control regions, etc. contain *motifs* or regulatory elements that control biological processes such as gene expression (Laurent and Philipp, 1997). Generally, activation or inhibition of gene expression is regulated by proteins referred to as transcription factors (TFs) that bind to particular sites in regulatory regions. For example, TATA box with a core sequence of 5′-TATAAA-3′ located in the promoter region of genes is bound by TFs such as TFIID, TFIIA and TFIIB to initiate transcription. Such TF binding sites (TFBS) are useful DNA motifs and their locations and detection are important to decipher and control sophisticated regulatory mechanism of gene expression.

In practice, sequence motifs occur with mutations or degeneration at sites, which are then referred to as motif instances. Motifs are denoted as (l,d)-motifs where l denotes the motif length and d denotes the maximum number of mutations allowed in the motif. Many techniques have been proposed for detecting motifs: enumeration algorithms (Sinha and Tompa, 2003), probabilistic algorithms and (Bailey and Elkan, 1994; Boucher *et al.*, 2007; Hertz and Stormo, 1999; Lawrence *et al.*, 1993; Roth *et al.*, 1998; Yao *et al.*, 2006) deterministic algorithms (Ho *et al.*, 2009; Pavesi *et al.*, 2004; Sagot, 1998; Sun *et al.*, 2010a, b). In addition, ensemble techniques have been attempted (Wijaya *et al.*, 2008).

An algorithmic challenge for motif discovery was posed in Pevzner and Sze (2000): find 20 planted motif instances of (l,d)-motifs in 20 DNA sequences of length 600. It was observed that most existing techniques were unable to detect a $(15,4)$-motif. This problem, referred to as motif challenge problem, planted motif discovery problem or subtle/weak motif discovery problem, has recently received increased attention. Many deterministic graph-based algorithms have been proposed for weak motif discovery, which begin by representing l-mer substrings sampled from sequences in the dataset as nodes in a multipartite graph. Thereafter, the nodes with $\leq 2d$ Hamming distance are connected by edges as two instances of the same motif must not differ by more than in $2d$ sites. The cliques of a sufficient size of this graph represent instances of the motifs in the dataset.

A graph-based algorithm WINNOWER was proposed to solve the motif challenge problem (Pevzner and Sze, 2000). cWINNOWER imposes an additional filtering constraint to WINNOWER to decide whether edges should be deleted or retained (Liang *et al.*, 2004). The modification substantially improves the execution time compared with WINNOWER. We have previously proposed an exact graph-based algorithm DPCFG, which efficiently detects weak motifs by constructing lists (cliques) of motif instances in graphs (Yang and Rajapakse, 2004). DPCFG had better accuracy and execution time than those of WINNOWER for sequences as long as 2000 for the (15, 4)-motif problem. However, for longer and weaker motifs such as (24, 8), DPCFG demands high memory requirements. This work attempts to solve problems with longer and weaker motifs with less memory and time requirements.

Probabilistic graph-based algorithms, such as MotifCut (Fratkin *et al.*, 2006) and MCL-WMR (Boucher *et al.*, 2007), have also been proposed for weak motif discovery. MotifCut formulates the problem into finding maximum density subgraphs in a weighted graph constructed using all l-mer substrings from the dataset. MCL-WMR is a heuristic that uses the Markov cluster to search for cliques of motif instances in weighted graphs. MCL-WMR has shown competitive execution time and accuracy.

Non-graph-based algorithms were also proposed. For example, PROJECTION enhanced discovery performances through finding good starting points for expectation maximization (Buhler and Tompa, 2002). PROJECTION produced higher performance than WINNOWER for weaker (l,d)-motif detection problems. However,

*To whom correspondence should be addressed.

its performance begins to drop significantly above sequence length of 1400 for the (15, 4)-motif problem (Buhler and Tompa, 2002).

A rule-based algorithm iTriplet was recently proposed for weak motif discovery (Ho *et al.*, 2009), which uses substrings of sample sequences to construct triplets. Based on several proposed rules, iTriplet generates candidate motifs from the intersection of three circles centered at each member of a triplet. The candidate motifs are then associated to the sequences. If a generated candidate motif is associated to an expected number of sequences in the dataset, it will be selected as the target consensus motif. iTriplet has been shown to be capable of handling highly degenerated long motifs, such as (18, 6)- and (24, 8)-motifs. However, it suffers from substantial computational requirements.

In this article, a new deterministic tree-structured graph-based algorithm is introduced for weak motif discovery. Using reference nodes from a pair of reference sequences, TreeMotif constructs one node trees with nodes from one specific sequence by dynamically appending new nodes from the other sequences as the leaf nodes of the trees. The TreeMotif is able to detect long weak motifs such as (40, 14) efficiently in time with acceptable memory requirements.

2 TREEMOTIF

TreeMotif is inspired by graph-based approaches to weak motif discovery. The novelty of TreeMotif is that it uses a tree-structured algorithm to find cliques from the graphical representation of motif instances. In addition, as a clique may not represent a deterministic motif consensus [for some sites, more than one bases are likely to appear (Price *et al.*, 2003)], TreeMotif, driven by samples instead of specific patterns, is capable of detecting motifs with highly degenerated positions. The algorithm consists of three main steps: graph representation, tree construction and motif refinement. In what follows in this section, we describe these steps in detail.

2.1 Graph representation

Given a set $s=\{s_i\}_{i=0}^{m-1}$ of m DNA sequences where $s_i=(s_{i,j})_{j=0}^{n-1}$ denotes the i-th sequence in the dataset and $s_{i,j}\in\Omega=\{A, C, G, T\}$, we are interested in detecting (l,d)-motifs. For simplicity, all the sequences are assumed to be of the same length n ($\gg l$) and one-occurrence-of-motif-per-sequence (OOPS) model is assumed. Let $v_i=\{v_{i,j}\}_{j=0}^{n-l}$ be the set of all l-mer subsequences in sequence s_i where $v_{i,j}=(s_{i,j'})_{j'=j}^{j+l-1}$ represents the subsequence starting at site j of the sequence. These subsequences are represented as nodes in a m-partite graph.

Let function D compute the Hamming distance between two nodes and $D_{\max}$ compute the maximum pairwise Hamming distance among the nodes in its arguments. In order to build the graph representing motif instances in the dataset, a pair of reference sequences are chosen: without loosing generality, let us choose s_0 and s_1 as reference sequences. The reference node pairs, one from each reference sequence, are selected as the node pairs that are of $2d$ distance apart. A graph is constructed by drawing edges between the reference node pair and each node in the other sequences $\{v_i\}_{i=2}^{m-1}$ if the maximum Hamming distance of the node to the reference node pair is $\leq 2d$. Algorithm 1 describes the selection of nodes for construction of trees. From each reference node pair, the set $\{P_i\}_{i=2}^{m}$ denotes the nodes in the graph for tree construction where P_i stands for the nodes selected from sequence s_i.

Algorithm 1 Node selection

```
Given $\{\{v_{i,j}\}_{j=0}^{n-l}\}_{i=0}^{m-1}$ representing $l$-mer subsequences on all sequence $\{s_i\}_{i=0}^{m-1}$
Select sequences $s_0$ and $s_1$ as reference sequences
Initialize $\{P_i=\emptyset\}_{i=2}^{m-1}$
for $\{j',j''\}_{j,j'=0}^{n-l}$ do
  if $D(v_{0,j'},v_{1,j''})\leq 2d$ then
    for $\{i\}_2^{m-1}$ do
      for $\{j\}_{j=0}^{n-l}$ do
        if $D_{\max}(v_{0,j'},v_{1,j''},v_{i,j})\leq 2d$ then
          $P_i=P_i\cup\{v_{i,j}\}$
return $\{P_i\}_{i=2}^{m-1}$
```

2.2 Tree construction

Another reference sequence (other than the reference pair for node selection) is selected for tree construction. Let us select s_2 as the reference, and then the nodes in P_2 are initialized as the individual roots for building trees. Let $T=\{\text{root}(T)\}\bigcup_{k=1}^{K}T_k$ be a K-ary tree with a finite set of nodes where root(T) denotes the root of the tree. Except the root node of the tree, other nodes are partitioned into k disjoint subsets $\{T_k\}_{k=1}^{K}$ where each T_k is a K-ary tree. Except for the root, each node in T has one unique parent node and can have up to K child nodes. Let parent() denote the parent node of the argument node, child() denote the set of child nodes of the argument node and depth() denote the depth of the argument node in the tree. Note that depth(root(T)) = 1.

Specifically, for each reference node pair, all the trees initialized are constructed by appending nodes in $\{P_i\}_{i=3}^{m-1}$ incrementally. Any branch of a tree is a connected path of nodes. A branch is said to be *extendable* by a node if the maximum Hamming distance between any one of the nodes on the branch and the node is $\leq 2d$. The extendable() function is designed by filtering tree T in a breadth-first manner by using a first-in-first-out queue Q, as given in Algorithm 2. The leaf nodes are retained in Q as filtered nodes. Initially, Q is set as empty. For the root node R of T, if $D(v_{i,j},R)\leq 2d$, R is inserted into Q; otherwise, filtering is stopped and Q is returned as empty. The process of possible filtering starts at the check on the front node in Q. If the node is a leaf node of T, filtering is stopped and Q is returned; otherwise, the node is removed from Q and its children (whose Hamming distances from $v_{i,j}$ are not > $2d$) are inserted into the end of Q. While Q is not empty, the above process is repeated. Note that a tree node $v_{i,j}$ corresponds to a node from S_i starting at position j.

As discussed, if queue Q is non-empty, all filtered nodes correspond to branches that are extendable by node $v_{i,j}$. Thus, we directly append $v_{i,j}$ onto these leaf nodes. If Q is empty, it means that no branches in T can be extendable by $v_{i,j}$. After check on appending of $v_{i,j}$, the same filtering and appending processes are carried out for all the other nodes in P_i. If not any node in P_i is appended, the current tree can be stopped for construction and cleared. The tree-construction algorithm is given in Algorithm 3.

2.3 Motif refinement

There may be spurious branches that cannot lead to motifs subjected to OOPS constraint and redundant branches that correspond to the

Algorithm 2 extendable($v_{i,j}, T$)

```
Given tree $T$ and a node $v_{i,j}$
$Q \leftarrow \emptyset$ // set the queue as empty
if $D(v_{i,j}, \text{root}(T)) \leq 2d$ then
    $Q.back() \leftarrow \text{root}(T)$; // insert the root to the end of $Q$
    while $Q \neq \emptyset$ and depth($Q.front()$) $< i-2$ do
        $F \leftarrow Q.front()$ // get the front node and delete it from $Q$
        for each node $c \in \text{child}(F)$ do
            if $D(c, v_{i,j}) \leq 2d$ then
                $Q.back() \leftarrow c$
return $Q$
```

Algorithm 3 Tree construction

```
Given a reference node pair $(v_{0,j'}, v_{1,j''})$ and $\{P_i\}_{i=2}^{m-1}$
for each node $v_{2,r} \in P_2$ do
    $v_{2,r} = \text{root}(T_r)$
    for $\{i\}_3^{m-1}$ do
        Flag = FALSE
        for each node $v_{i,j} \in P_i$ do
            if extendable($v_{i,j}, T_r$) then
                Add $v_{i,j}$ to $T_r$ according to $Q$
                Flag = TRUE
        if Flag is FALSE then
            clear $T_r$ and break;
return $\{T_r\}_{v_{2,r} \in P_2}$ with $(v_{0,j'}, v_{1,j''})$
```

same motif. The refinement of the trees as follows is necessary to include more target motif instances.

- *Elimination of spurious branches in tree construction:* if a branch cannot be extendable by any node in a specific sequence, pruning of this branch is performed by backtracking from the leaf node to a node whose parent has more than one child; and just prune away the subbranch from this node to the leaf node.
- *Merging redundant tree branches after tree construction:* once a tree is obtained, each branch together with related reference nodes corresponds to a clique of motif instances. Therefore, such cliques are merged if all the nodes are within $2d$ (making the clique size larger than m). This can guarantee more target instances included in the same clique.

After refinement, trees that have nodes from every sequence represent motifs. Consensuses of the motifs are then found using all the motif instances (that is, the nodes in the trees). An example on tree construction and motif refinement is given in Section A of Supplementary Material.

2.4 Time complexity

Suppose p is the probability of two random strings of length l having a Hamming distance $\leq 2d$. Then

$$p = \sum_{i=0}^{2d} \binom{l}{i} (3/4)^i (1/4)^{l-i}$$

where p is a measure about the weakness of the (l, d)-motifs. The larger the value p, the weaker the motif becomes (Buhler and Tompa, 2002; Ho *et al.*, 2009; Yang and Rajapakse, 2004).

Table 1. Comparisons of time complexities of different motif discovery algorithms

WINNOWER	iTriplet	DPCFG	TreeMotif
$m^4n^4p^4$	$mn^3pl^3d^2$	$mn^3p^2 + n^mp^{3m-6}$	mn^4p^2

Table 1 shows the time complexity of TreeMotif and related motif discovery algorithms. The time complexity of TreeMotif is approximately $O(mn^4p^2)$ when $p < 0.32$ and $n \leq 800$. When sequences are long or motifs become weaker, TreeMotif is able to reduce the execution time by generating new branches more accurately compared with DPCFG.

In addition, TreeMotif does not have to duplicate the previous tree for new nodes. It only needs to append new nodes onto the previous tree after filtering. Necessary pruning operations only take place when all nodes (selected from a specific sequence) have been checked for extendability. This makes information from the previous sequences intact without duplications, reducing execution time and memory requirements.

3 RESULTS

The performance of TreeMotif was evaluated on synthetic data as well as on real biological data. Synthetic datasets provide quantitative measures that can compare its performance with the existing techniques (Sandve *et al.*, 2007; Tompa *et al.*, 2005). Experiment results of MEME, GibbsDNA, PROJECTION, iTriplet, DPCFG and TreeMotif were obtained from runs on a workstation with 2.66 GHz CPU and 3 GB RAM. Detections with MEME, GibbsDNA, PROJECTION and DPCFG were based on exactly one motif occurrence per sequence (OOPS) while with iTriplet and TreeMotif, detections were made with at least one motif instance per sequence.

3.1 Synthetic data

Following Pevzner and Sze (2000), synthetic datasets are generated using the model of fixed number of mutations. First, a specific number of i.i.d. sequences were generated. Second, an instance of a target motif of length l is planted at a random position of every sequence. Third, for each motif instance, equal to or less than d positions were mutated by randomly selecting from l positions.

3.1.1 Motif challenge problem for one dataset, 20 instances of a $(15, 4)$-motif were planted in random positions of 20 i.i.d. sequences of length 600. A total of 50 synthetic datasets were created to obtain unbiased estimates of performances. Following Pevzner and Sze (2000), nucleotide performance coefficient *nPC*[1] and sensitivity *nSn*[2] were evaluated using true positives (TPs), false positives (FPs), false negatives (FNs) and true negatives (TNs), to compare the performances of different methods. Table 2 shows *nPC*, *nSn* and execution time (with ± SD) averaged over the results of all 50 datasets. Three motifs ranked as the best ones were used to compare the highest *nPC*. For TreeMotif, different cliques of motif instances are ranked based on information content (Schneider and Stephens,

[1]*nPC*=TP/(TP + FN + FP).
[2]*nSn*=TP/(TP + FN).

1990). Note that *nPC* is <1.0 because random substrings could exist in the sequences, which are as conserved as the target motif instances.

TreeMotif, DPCFG and iTriplet show comparable *nPC* and *nSn*, higher than those of MEME, GibbsDNA and PROJECTION. Thereinto, DPCFG shows the highest *nPC* but with lower *nSn* while iTriplet shows perfect *nSn* at the cost of lower *nPC*. With OOPS model, TreeMotif can achieve the same results as DPCFG; if the target is to find an exact consensus motif, TreeMotif can achieve the same results as iTriplet. On performance, TreeMotif can be considered as an eclectic method of iTriplet and DPCFG.

Note that TreeMotif and DPCFG did not detect the correct motif (at low *nSn*), because the clique of the correct motif instances was (found but) not reported as the top ones. Although pattern-driven methods such as iTriplet can show better *nSn* due to the alignment using candidate consensus patterns, when there is no exact consensus motif in a dataset, they usually fails to find the motif instances. This is why the clique of motif instances are kept as the target for TreeMotif. As for time efficiency, TreeMotif and DPCFG required comparable execution times that are less than the other methods for (15, 4)-motif problem.

3.1.2 Weakness of embedded motif The *p* value reflects the weakness of an (l, d)-motif (Section 2.4). With $n=600$ and $m=20$, the average *nPC* and average execution time of TreeMotif, DPCFG and iTriplet with increasing *p* are depicted in Table 3. For relatively stronger motifs (showing smaller *p* value), such as (13, 3) and (40, 12), TreeMotif shows comparable performance to DPCFG. For weaker motifs such as (14, 4) and (19, 6), TreeMotif shows relatively higher *nPC* and consumes less execution time. In addition, TreeMotif can successfully produce results for much weaker motifs such as (16, 5), (24, 8) and (40, 14) while DPCFG cannot produce results due to memory shortage.

As seen, when $l<20$, TreeMotif shows comparable *nPCs* to iTriplet while consuming less execution times. TreeMotif performs poor on the (16, 5)-motif, because many random cliques occur in such data. In this case, more informative target function on ranking the cliques are needed. Meanwhile, when $l>20$, TreeMotif can find the target motif instances with slightly higher *nPCs* while consuming much less execution time than iTriplet. For instance, for the (24, 8)-motif problem which other methods such as MEME and GibbsDNA all have difficulty in dealing with (Ho *et al.*, 2009), TreeMotif requires about one-seventh the execution time of iTriplet.

iTriplet yielded low values of *nPC* because it failed to find the target motif instances for a part of the datasets tested (due to out of memory). As seen from Table 3, performance of iTriplet is influenced by both the weakness *p* of the motif and the motif length while that of TreeMotif is influenced only by weakness. Generally, pattern-driven algorithms cannot scale efficiently with the length of motif as the number of candidate motifs to test grows exponentially with l. TreeMotif is a sample-driven algorithm, which is more scalable for dealing with long weak motifs. For example, with settings (13, 3) and (40, 12) showing approximately the same *p*, iTriplet requires much more time on the latter while TreeMotif performs consistently.

3.1.3 Length of embedded sequences We tested TreeMotif on synthetic data with sequence length increased from 600 to 2000. For each dataset, one instance of a (15, 4)-motif is planted in each of the 20 sequences. Performances of the algorithms are shown in Figures 1 and 2.

Figure 2 shows that PROJECTION can produce slightly better results than MEME and GibbsDNA, but their *nPCs* are all <0.5. Approximately, *nPC* of DPCFG is the highest when sequence length is <1500. This is because it outputs exactly *m* instances and most of them are the target instances. However, when the sequence length is >1500, *nPC* of DPCFG decreases quickly. This is because too many target instances have been substituted by spurious sequence instances. *nPC* of TreeMotif and iTriplet is >0.70 as sequence length is increased to 2000. Their *nPC* also decreases when the sequences become longer since random substrings that are as conserved as the target motif instances were collected in the resulted cliques. That is, the main reason why low *nPC* is resulted by DPCFG is that it misses true signals, whereas TreeMotif and iTriplet report false signals. Overall, iTriplet has similar *nPCs* as TreeMotif. However,

Table 2. Comparisons on performances on motif challenge problem

Algorithms	*nPC*	*nSn*	Time (s)
GibbsDNA	0.01 ± 0.01	0.02 ± 0.02	3.46 ± 0.37
MEME	0.22 ± 0.20	0.32 ± 0.25	3.04 ± 0.08
PROJECTION	0.42 ± 0.27	0.54 ± 0.30	75.43 ± 3.12
iTriplet	0.94 ± 0.06	1.00 ± 0.00	185.56 ± 5.30
DPCFG	0.97 ± 0.05	0.98 ± 0.03	0.70 ± 0.03
TreeMotif	0.95 ± 0.05	1.00 ± 0.01	0.74 ± 0.04

Table 3. Comparisons of *nPC* and execution time on (l, d)-motifs with increasing value of *p*

Alg.	(l, d) *p*	(13, 3) 0.0243	(40, 12) 0.0262	(12, 3) 0.0544	(14, 4) 0.1117	(19, 6) 0.1749	(16, 5) 0.1897	(30, 10) 0.1966	(24, 8) 0.2338	(40, 14) 0.2850
a	nPC	0.94,0.06	0.46,0.50	0.81,0.07	0.84,0.06	0.93,0.14	0.86,0.08	–	0.94,0.22	–
	Time	0.55,0.02 m	0.24,0.15 h	0.05,0.00 h	0.29,0.01 h	2.00,0.07 h	1.91,0.06 h	–	15.50,1.00 h	–
b	nPC	0.98,0.05	1.00,0.00	0.87,0.09	0.76,0.23	0.87,0.11	–	1.00,0.00	–	–
	Time	0.16,0.01 s	0.18,0.02 s	0.83,0.05 s	1.10,0.12 m	0.61,0.05 h	–	0.53,0.05 h	–	–
c	nPC	0.94,0.05	1.00,0.00	0.82,0.07	0.80,0.19	0.94,0.06	0.18,0.18	1.00,0.00	0.97,0.08	1.00,0.00
	Time	0.19,0.01 s	0.33,0.01 s	0.75,0.04 s	0.42,0.03 m	0.20,0.02 h	1.16,0.16 h	0.08,0.01 h	2.09,0.30 h	6.09,1.10 h

Alg.: a, iTriplet; b, DPCFG; c, TreeMotif; each mean value in the table followed by the standard error; –, not available as the related algorithm failed to yield the motifs.

for certain datasets with sequences of length 1000, iTriplet runs out of memory and thus fails to produce correct results. This results in its lower average *nPC*.

Figure 1 compares the execution times of TreeMotif and DPCFG. Part of the execution time of iTriplet is also included. As seen, iTriplet consumes much more execution time than TreeMotif and DPCFG. As n is increased from 600 to 2000, TreeMotif requires less time to discover motifs than DPCFG. For example, when the sequence length is <1300, two algorithms require comparable time. But when the sequence length is 2000, TreeMotif consumes about one-third the execution time of DPCFG.

3.1.4 Similarity of embedded sequences For synthetic data, all bases were generated with no bias on their contents, that is, their occurrences are equally likely. Any bias on the bases introduces spurious signals into sequences (Buhler and Tompa, 2002). This may cause TreeMotif to take a longer time to find motifs (see Section B of the Supplementary Material). For example, with $G + C = 0.35$, TreeMotif requires >100 s to find out a (15, 4)-motif while it requires <1 s when $G + C = 0.5$. Execution time of probabilistic algorithms such as MEME is not affected by such settings. However, they cannot guarantee the discovery of the target instances. For pattern-driven algorithms, they are also not affected. However, as discussed in Section 3.1.1, such algorithms take effect only for data containing deterministic consensus motifs. Therefore, although sample-driven algorithms such as TreeMotif are time consuming for the discussed settings, one may be inclined to use them to produce results if not enough prior information is known about the data.

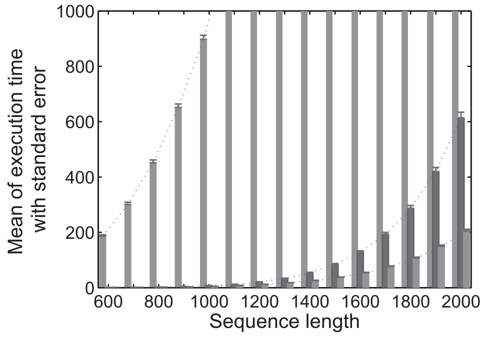

Fig. 1. Execution times of algorithms with increased sequence length on the (15, 4)-motif problem. Note: algorithms on each point (from left to right) are iTriplet, DPCFG and TreeMotif. Time unit: second.

3.2 Real data

To test the practicality, TreeMotif was tested on datasets of *pre-proinsulin*, *DHFR* and *c-fos* (Keich and Pevzner, 2002), *LexA* (Hertz *et al.*, 1990), and *E.coli CRP* (Stormo and Hartzell, 1989). Discovered and published motifs are shown in Table 4. For all the datasets, each sequence was assumed to contain one instance and experiments were carried out with l starting from 6, and pair (l,d) were set empirically to check if the target motif consensus is discovered. Data *c-fos*, *DHFR* and *prerpoinsulin* consist of sequences in the upstream regions of eukaryotic genes. Motifs they contained had only a few mutations, which were discovered efficiently and accurately by TreeMotif.

3.2.1 E.coli CRP data For *E.coli CRP* data from GenBank Release 55, each sequence contains at least one *CRP*-binding site. Although each sequence consists of only 105 bases, the motif tgtgaxxxxgxtcaca is rather weak. For this data, TreeMotif can produce the motif as *tgtgaxxxagxtcaca*tt with $l = 18$, $d = 6$. With the same l and d, PROJECTION cannot find the motif although it can produce the consensus as t***tgtgatggagttcaca***ttt with $l = 20$, $d = 4$. Also, as not all the bases on each position are unique, pattern-driven methods such as iTriplet fail to find the motif. This demonstrates the better capability of TreeMotif to discover weaker motifs than PROJECTION [note that (18, 6) is a weaker setting compared with (20, 4)] and the pattern-driven methods.

3.2.2 LexA data This data contain 16 sequences but only 14 of them contain more than one motif instances (sequences *himA* and *uvrC* contain no instances). By setting $l = 24$ and $d = 8$, one motif is discovered as *tactgtatataxaxxcagtx*xaat (with 80% of the bases of the published motif). In addition, after excluding *himA* and *uvrC* (to obtain the OOPS data), *tactgtatataxatacagta* is discovered by setting $l = 20$ and $d = 5$.

3.2.3 Other benchmark data Benchmark datasets for the evaluation of motif discovery algorithms have been constructed by

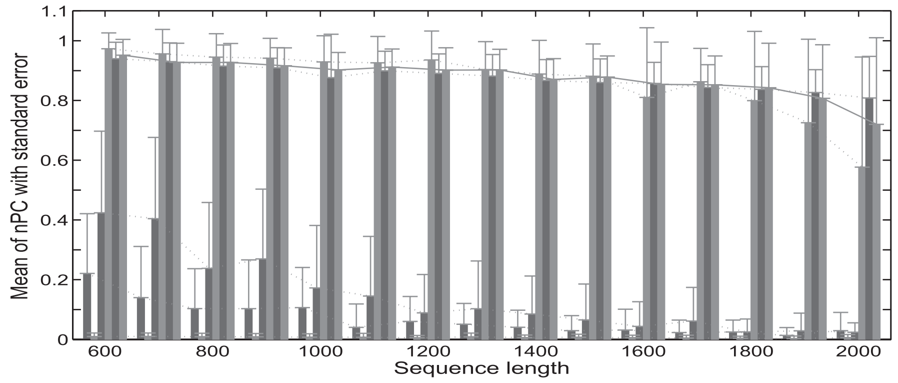

Fig. 2. *nPCs* of different algorithms with increasing sequence length on the (15, 4)-motif discovery. Algorithms on each point (from left to right) are MEME, GibbsDNA, PROJECTION, DPCFG, iTriplet and TreeMotif.

Table 4. Results on real data with (l, d)-motifs

Data	Discovered	(l,d)	Published
1	***ccatattaggacatct***	(16, 3)	ccatattaggacatct
2	***ttcgcgccaaact***	(13, 2)	ttcgcgccaaact
	cctcagcccc	(10, 2)	cctcagcccc,
3	***agacccagca***	(10, 2)	agacccagca,
	ccctaatgggcca	(13, 2)	ccctaatgggcca
4	***tgtgaxxxagxtcaca***tt	(18, 6)	tgtgaxxxxgxtcaca
	tactgtatataxatacagta	(20, 5)	
5	***tactgtatataxaxxcagtx***xaat	(24, 8)	tactgtatatatatacagta

x, none of the bases appear with opportunity > 50%; 1, *c-fos*; 2, *DHFR*; 3, *Preproinsulin*; 4, *E.coli CRP.*; 5, *LexA*.

Sandve *et al.* (2007) using binding sites extracted from TRANSFAC database version 9.4. The real data version of the benchmarks contains 50 datasets formed using genomic sequences. Each dataset contains five or more sequences of the maximum length of sequences as 2000 bp. For each motif, gaps are not allowed and each sequence is expected to have one or more occurrences of the motif instances. Compared with the benchmark designed by Tompa *et al.* (2005), these datasets are more appropriate for evaluating motif discovery algorithms (Sandve *et al.*, 2007).

On this dataset, using performance of MEME as the base, the performance of TreeMotif was compared with Weeder (Pavesi *et al.*, 2004) which is a suffix tree-based quasi-exact algorithm, as it shows relatively better performance than some other tools according to the analysis in Tompa *et al.* (2005). Following Tompa *et al.* (2005) and Sandve *et al.* (2007), the nucleotide level correlation coefficient *nCC* was used as the performance metric.[3]

The discovery results of TreeMotif is uploaded to the web benchmark tool provided by Sandve *et al.* (2007). We compare *nCC* of TreeMotif and Weeder by keeping FP approximately at the same level, as shown in Figures 3 and 4. The *nCC* of all datasets produced by MEME are sorted in the ascending order and used as a basis. Then the performances of Weeder and TreeMotif on each dataset are drawn accordingly. As seen, TreeMotif achieves better *nCC* than weeder for 34 out of 50 datasets. Overall, at the comparable level of FP, TreeMotif achieves an average *nCC* of 0.194 compared with that of Weeder as 0.096.

For the five datasets on which Weeder achieves better *nCC* by reporting fewer FP, it also reported less or equal TP than TreeMotif. Weeder outperforms other existing tools, because it reports only the ‘strongest’ motifs (thus fewer FP) (Tompa *et al.*, 2005). For the other 11 datasets, TreeMotif fails as too many random cliques were found while the target clique is not reported as the top one (this is similar to the discussion in paragraph 5 of Section 3.1.1). A possible reason for this may be that the contents of nucleotide bases are biased, i.e. bases are not uniformly distributed. For instance, the $G + C$ content of these files is ∼0.55 or 0.45. This increases the probability of two random strings being within $2d$ (related to the problem we addressed), especially when the background sequences are long. As a result, the number of random cliques, which may inhibit the detection of correct target, increases.

[3]
$$nCC = \frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{FP}+\mathrm{TN})(\mathrm{TN}+\mathrm{FN})(\mathrm{FN}+\mathrm{TP})}}$$

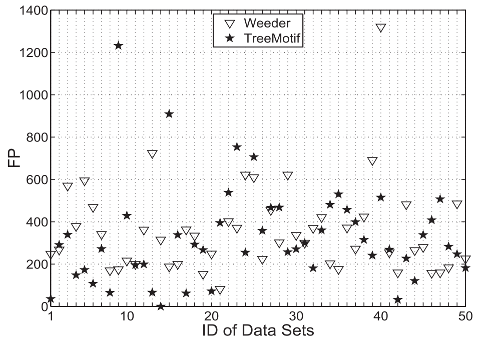

Fig. 3. False positives of TreeMotif and Weeder of real benchmark data.

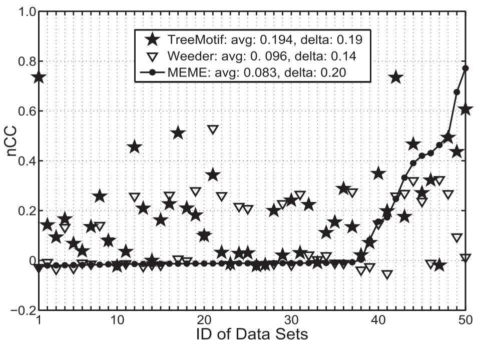

Fig. 4. TreeMotif, Weeder and MEME on real benchmark data (Sandve *et al.*, 2007).

4 DISCUSSION

Weak motif discovery is an important yet unresolved problem in computational biology. Several graph-based exact methods such as WINNOWER and DPCFG have shown improved accuracy compared with heuristics or probabilistic methods. However, clique finding to discover the motifs can be time consuming. In this article, we designed TreeMotif for discovering weak (l, d)-motifs by building up trees on nodes of graphs. After initialization, TreeMotif appends new nodes onto existing trees, where the appending operations happen only when cliques indicated by the leaf nodes are extendable by the new nodes.

The novel-constrained strategy of clique construction introduced in TreeMotif can avoid trivial operations on intermediate results of final cliques. In addition, by doing motif refinement, the sensitivity of motif discovery (defined in Section 3.1.1) can be increased compared with the list algorithm in Sun *et al.* (2010b) which produces exact OOPS motif instances by using a recursive node selection. Performance comparisons with several popular motif discovery methods have shown that TreeMotif detects weak motifs accurately, especially for relatively longer weak motifs. Moreover, it performs better on a recent benchmark constructed using real motif data.

One drawback of TreeMotif is that its memory requirement grows exponentially as the sequence length n or parameter p is increased. Because of large memory requirements, TreeMotif failed to find weaker (l, d)-motifs such as (13, 4), (15, 5), (17, 6), (18, 6) and (19, 7). This is due to the maintenance of increased number of tree branches (i.e. intermediate cliques of size $i \leq m-1$). In all such cases, if the number of sequences is small, say K, TreeMotif might be able to find K-cliques ($K < m$). Therefore, by devising K that divides a dataset of m sequences into m/K subdatasets, TreeMotif can be used on these subdatasets. After all cliques of size K are found, they could be merged as larger ones. This will be part of our future work.

Another limitation of TreeMotif is that it can deal with only OOPS type of data at the current stage. While in practice, the data may contain sequences with no motif instances. Therefore, techniques to avoid noisy instances during tree construction have to be designed. For example, a strategy has been introduced to construct cliques of all varying sizes and has shown success in finding motifs in noisy data (Ho and Rajapakse, 2006). Correspondingly, a similar strategy might be used that appends new nodes onto not only leaf nodes but also nodes on the upper levels of a tree. In this way, cliques of all sizes will be constructed instead of those of only size i. This might involve more space; however as a trade-off, the noise in the cliques might be excluded. Generally, the strategy for tree construction still needs improvement to make TreeMotif more time and memory efficient.

Funding: This study is supported by the NTU Research Scholarship; ARC 9/10 Tier-2 grant of Ministry of Education, Singapore (to J.C.R.) in part.

Conflict of Interest: none declared.

REFERENCES

Bailey,T.L. and Elkan,C. (1994) Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Menlo Park, CA, pp. 28–36.

Boucher,C. *et al.* (2007) A graph clustering approach to weak motif recognition. In *7th Workshop on Algorithms in Bioinformatics (WABI '07)*. Vol. 4645 of *Lecture Notes in Computer Science* Springer, Berlin/Heidelberg, pp. 149–160.

Buhler,J. and Tompa,M. (2002) Finding motifs using random projections. *J. Comput. Biol.*, **9**, 225–242.

Fratkin,E. *et al.* (2006) MotifCut: finding regulatory motifs with maximum density subgraphs. *Bioinformatics*, **22**, e150–e157.

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

Hertz,G.Z. *et al.* (1990) Identification of consensus patterns in unaligned DNA sequences known to be functionally related. *Comput. Appl. Biosci.*, **6**, 81–92.

Ho,L.S. and Rajapakse,J.C. (2006) Graphical approach to weak motif recognition in noisy data sets. In *Workshop on Pattern Recognition in Bioinformatics (PRIB 2006)*. Vol. 4146 of *Lecture Notes in Computer Science*, Springer, Hong Kong, China, pp. 23–31.

Ho,E.S. *et al.* (2009) iTriplet, a rule-based nucleic acid sequence Motif Finder. *Algorithms Mol. Biol.*, **4**.

Keich,U. and Pevzner,P.A. (2002) Finding motifs in twilight zone. *Bioinformatics*, **18**, 1374–1381.

Laurent,D. and Philipp,B. (1997) Searching for regulatory elements in human noncoding sequences. *Curr. Opin. Struct. Biol.*, **7**, 399–406.

Lawrence,C.E. *et al.* (1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, **262**, 208–214.

Liang,S. *et al.* (2004) cWINNOWER algorithm for finding fuzzy DNA motifs. *J. Bioinformatics Comput. Biol.*, **1**, 47–60.

Pavesi,G. *et al.* (2004) Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Res.*, **32** (Suppl. 2), W199–W203.

Pevzner,P.A. and Sze,S.-H. (2000) Combinatorial approaches to finding subtle signals in DNA sequences. *Intell. Syst. Mol. Biol.*, **8**, 269–278.

Price,A. *et al.* (2003) Finding subtle motifs by branching from sample strings. *Bioinformatics*, **19** (Suppl. 2), ii149–ii155.

Roth,F.P. *et al.* (1998) Finding DNA regulatory motifs within unaligned non-coding sequences clustered by whole-genome mRNA qantitation. *Nat. Biotechnol.*, **16**, 939–945.

Sagot,M.-F. (1998) Spelling approximate repeated or common motifs using a suffix Tree. In *Theoretical Informatics: Third Latin American Symposium (LATIN '98). Lecture Notes in Computer Science*, Springer, Brazil, pp. 374–390.

Sandve,G.K. *et al.* (2007) Improved benchmarks for computational motif discovery. *BMC Bioinformatics*, **8**, 1–13.

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

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

Stormo,G.D. and Hartzell,G. W. (1989) Identifying protein-binding sites from unaligned DNA fragments. In *Proceedings of the National Academy of Sciences of the United States of America*, PNAS, USA, pp. 1183–1187.

Sun,H.Q. *et al.* (2010a) ListMotif: a time and memory efficient algorithm for weak motif discovery. In *Proceedings 2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2010)*. IEEE, inc., China, pp. 254–260.

Sun,H.Q. *et al.* (2010b) RecMotif: a novel fast algorithm for weak motif discovery. *BMC bioinformatics*, **11** (Suppl. 11).

Tompa,M. *et al.* (2005) Assessing computational tools for the discovery of transcription factor binding sites. *Nat. Biotechnol.*, **23**, 137–144.

Wijaya,E. *et al.* (2008) MotifVoter: a novel ensemble method for fine-grained integration of generic motif finders. *Bioinformatics*, **24**, 2288–2295.

Yang,X. and Rajapakse,J.C. (2004). Graphical approach to weak motif recognition. *Genome Informat. Ser.*, **15**, 52–62.

Yao,Z. *et al.* (2006) CMfinder-A covariance model based RNA motif finding algorithm. *Bioinformatics*, **22**, 445–452.