International Journal of Computational Intelligence Systems, Vol. 4, No. 5 (September, 2011), 1032-1041

A Frequent Pattern Mining Method for Finding Planted Motifs of Unknown Length in DNA Sequences*

Caiyan Jia [1] , Ruqian Lu [2,3] , Lusheng Chen [3]

[1] *Department of Computer Science, Beijing Jiaotong University, Beijing 100044, China*
E-mail: cyjia@bjtu.edu.cn

[2] *Institute of Mathematics, Chinese Academy of Sciences, Beijing 100080, China*
E-mail: rqlu@math.ac.cn

[3] *Shanghai Key Lab of Intelligent Information Processing & Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China*
E-mail: lschen@fudan.edu.cn

Abstract

Identification and characterization of gene regulatory binding motifs is one of the fundamental tasks toward systematically understanding the molecular mechanisms of transcriptional regulation. Recently, the problem has been abstracted as the challenge planted (l,d)-motif problem. Previous studies have developed numerous methods to solve the problem. But most of them need to specify the length l of a planted motif in advance and use depth first search strategy. In this study, we present an exact and efficient algorithm, called Apriori-Motif, without given the length l of a planted motif a priori. And a breadth first search strategy is used to prune search space quickly by the downward closure property utilized in Apriori, which is a classical algorithm for frequent pattern mining. Empirical study shows that Apriori-Motif is better than some existing methods.

Keywords: Motif, frequent pattern, Apriori, downward closure property.

1. Introduction

In the post-genomic era, a major challenge is represented by deciphering expression regulation of thousands of annotated genes in genomes. However, experimental studies of transcriptional regulation are time consuming. Computer aided researches of large scale expression data and functional information on genes are greatly helpful. For understanding gene transcription and regulation, one of the basic steps is to find signals in DNA sequences[1,2,3].

In statistical sense, a signal (often called motif) in DNA sequences is not exactly identical but presents mutations. Usually, this signal is a short subsequence, typically about 10 bp (base pairs) long[1], in the midst of a great amount of statistical noise and is too complicated to be discriminated by computational methods. It makes that the existing methods still have low performance on identifying weak conserved motifs (also called subtle motifs) in DNA sequences[1,2].

*The preliminary form was appeared in the Proceedings of RSKT 2010.

Published by Atlantis Press

In the literature, Prevzner & Sze formalize the problem as the planted (l,d)-motif problem (PMP for short)[4]. Given a set of strings S=$\{s_1, s_2, \ldots, s_N\}$ over a symbol set Σ=$\{$A,C,G,T$\}$ such that $|s_i| \leqslant L$, $1 \leqslant i \leqslant N$ and positive integers l and d such that $1 \leqslant l \leqslant L$ and $0 \leqslant d < l$, the planted (l,d)-motif problem is to find a string $t \in \Sigma^l$ such that for every string s_i in S, there exists a substring (consecutive symbol string in a sequence) t_i in s_i such that $d(t,t_i) = d$, where $d(t,t_i)$ means the Hamming distance between t and t_i.

In other words, a conserved motif at length l with just d mutations is planted into each at most L bp background sequence. The target of PMP is to find the planted motif and all its variants on a set of N sequences only given l and d. It's believed that when the ratio d to l is larger than 0.25, the planted motif is subtle and hard to be discriminate from the background sequences[5].

The most classical (l,d)-planted motif problem is (15, 4), where a planted motif also its variants need to be found on a sample of 20 sequences, each 600 nucleotides long and containing an unknown variant of the planted motif at length 15 with 4 mismatches[4]. Besides proposing a challenge problem to researchers, PMP model also enables to test the performance of any motif finding algorithm.

Previous studies have developed numerous methods in order to find planted motifs, including SPELLER[6], WINNOWER[4], SP-STAR[4], MITRA[7], PROJECTION[8], MULTI-PROFILE[5], PatternBranching[9], CENSUS[10], WEEDER[11], PMS[12], Voting[13,14], RISOTTO[15], etc.

By large, two kinds of strategies are used. One is local optimal strategy, e.g. Gibbs Sampling method[8,16]. The kind of algorithms are based on PWM (Position-Specific Weighted Matrix, also called profile) model, can find motifs of any specified length very efficiently. But they are inevitably to slump into a local optimal. The other is heuristic enumeration strategy based on consensus model (more accurately, mismatch model)[6,10,11]. The kind of algorithms can find all exact solutions by searching the whole pattern space, but they are limited by the time or the space complexity of algorithms[17]. That is to say, the length l of a planted motif and the number d of mutations that the algorithms can execute are still relative small so far. Thus, people are still struggling for giving an efficient algorithm for PMP with much longer l and much larger d.

Usually, current heuristic enumeration algorithms have the following properties.

1. The depth first search strategy is used to enumerate all potential signals.

2. Most of algorithms need to know the length of a target motif in advance.

3. Most of algorithms suppose there exists an occurrence or a variant of a motif in each sequence.

4. Half of methods only deal with the case that the allowed mismatches are just d.

In fact, researchers may not know the size l of a planted motif a priori. And they are more likely to be interested in motifs that are d or fewer mutations away form each rather than exact d mutations. Moreover, as experimental data are commonly rife with noise, it is likely that some sequences may contain no motifs at all[18,19]. Thus, in this study, we intend to solve PMP under the conditions that the length l of a planted motif is unknown a priori, there are at least q ($q \leqslant N$) sequences where each contains a planted signal, and there are at most d mismatches between a planted signal in a sequence and the planted motif.

Based on the previous study[17], at the inspiration of frequent pattern mining techniques, we give a new algorithm, Apriori-Motif, for finding motifs exactly and efficiently under the conditions specified above in this paper. Similar with Apriori[20], Apriori-Motif uses breadth first search to scan the whole patten space indexed by the consensus tree structure[17]. And it can find motifs with length from $d+1$ to l iteratively by the downward closure property of motifs. The empirical studies have shown that Apriori-Motif can solve PMP without given l, with up to d mismatches and with quorum constraint q at reasonable time and low main memory.

The rest of paper is organized as follows. Section 2 introduces some definitions used in the paper. Section 3 presents the algorithm, Aprior-Motif. Section 4 analyzes its complexity. Section 5 gives some experimental results and compares the algorithm with other methods. Section 6 concludes the paper.

2. Preliminaries

Definition 1: Let $a = a_1a_2\cdots a_n$ and $b = b_1b_2\cdots b_n$ be two strings from Σ^+. The Hamming distance $d(a,b)$ between a and b is defined as

$$d(a,b) = \sum_{i=1}^{n} \varepsilon(a_i,b_i),\ \ \varepsilon(a_i,b_i) = \begin{cases} 1, & a_i \neq b_i, \\ 0, & otherwise. \end{cases}$$

It can be interpreted as the number of symbol mutations needed to turn one string into another.

Definition 2: Given $d \geqslant 0$ as the number of maximally allowed mutations (or mismatches), any string b with $d(a,b) = x \leqslant d$ is called an x-mutated copy (or simply mutated copy) of a and vice versa. A zero mutated copy is also called an exact copy. All x-mutated copies of a, where $x \leqslant d$, form the d-neighborhood of a. a is called the center of this neighborhood.

Given a center a, the size of d-neighborhood of a is bounded by $v(d,l)$, where

$$v(d,l) = \sum_{i=0}^{d} \binom{l}{i} (|\Sigma|-1)^i.$$

And the number of all d-mutated copies of a is at most

$$v'(d,l) = \binom{l}{d} (|\Sigma|-1)^d.$$

Thus, PMP with at most d mutations is much harder than that with exact d mutations since the former has much larger search space than the latter.

Definition 3: A set with property P is said to satisfy downward closure property if all nonempty subsets of the set also have the property P.

It's easy to know that motifs satisfy the downward closure property. For an example, if TCTGAC satisfies the quorum constraint q that their x-mutated copies ($x \leqslant d$) appears in at least q ($q \leqslant N$) sequences of a sample, then its any substring, e.g. TCTGA or CTGAC, satisfies this constraint. Thus, for finding a planted motif, we can search the pattern space from short strings to long strings since if one of TCTGA and CTGAC does not satisfy the quorum constraint, TCTGAC does not satisfy the constraint definitely and can be pruned from the search space. The idea is just the essence of Apriori[20], will be used in Apriori-Motif.

Definition 4: A tree-like structure is called consensus tree if it is used to storage all possible motifs in the searching process (see Lu and Jia[17]). And a consensus tree has the following properties.

1. There is one and only one path corresponding to a potential motif. The length of the path is just that of the motif.

2. A full consensus tree is a complete $|\Sigma|$ branches tree. But in real applications, only nodes (corresponding motifs) with x-mutated copies in at least q inputting sequences are allowed to grow in the tree, where $x \leqslant d$ and $q \leqslant N$.

3. A candidate motif spelled by the path ending at the node might not occur in sequences at all although its x-mutated copies ($x \leqslant d$) occur more than q times in at least q different sequences.

4. All nodes in a tree can be divided into two classes. One is real nodes representing real signals contained in sequences. They are allowed to grow in the next level of the tree. The other is virtual nodes representing candidate but false signals. They are prohibited to grow in the tree.

5. For a non-root node, there is a link (pointer) between the son of the node and the brother of the node, where the son and the brother have the same node content (a symbol in Σ). Thus, the branches of a node are the same as the active branches (corresponding nodes are real node, can grow up towards the next layer) of its uncle. We call it heredity property of a consensus tree. The property just comes from the downward closure property of a motif. Thus, only real nodes need the link structure.

A typical example of a consensus tree is shown in Figure 1. It is a four-branch tree. And the content of a node is one of the four nucleotides A, C, G, T. The path from the root to a node represents a candidate motif. All candidate motifs make up the entire search space. And each candidate motif has one and only one path in the tree. This is a compressed compact structure for representing a search space. In general, the longest signal is just the planted motif.

In Figure 1, the red nodes represent for all real nodes. The green nodes stand for all virtual nodes. L_i denotes the real signals in the i-th level. C_i de-

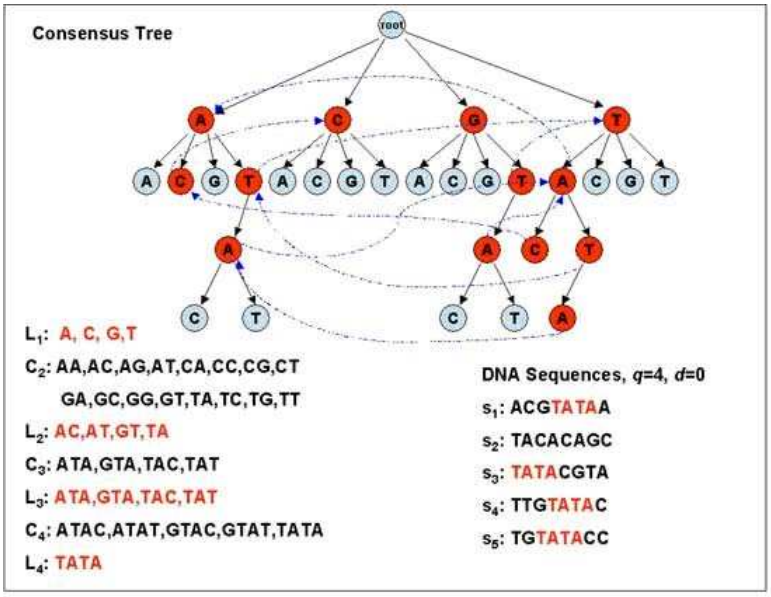

Figure 1: Consensus tree of the sample S={s1, s2, ⋯, s5}

notes all candidate signals in the i-th level. $L_i \subseteq C_i$. The longest signal contained in the sequences is TATA. This structure is used in Apriori-Motif for storing and scanning the pattern space of a sample. And according to the downward closure property, the branch TA can grow up two branches TAC and TAT since its uncle A has two active branches AC and AT. The branch TAT can grow up one branch TATA since it uncle AT has an active branch ATA. They are just examples of heredity property of a consensus tree.

3. Apriori-Motif

We know that any string of length L can be degenerated into L suffixes, and these suffixes can be stored in a suffix tree [21]. When the suffix tree of a string is constructed, searching for a substring of length m in the string just requires time $O(m)$. For the problem of motif finding, we generally need to repeatedly search DNA substrings in a set of sequences multiple times. A brute force string search is going to be terrible and inefficient. Thus, the index structure, suffix tree, is used in our algorithm. In fact, many algorithms including SPELLER and WEEDER use the structure to store DNA sequences for searching substrings quickly.

However, we observed that only the first $l+1$ (l is the maximal length of real signals contained in a sample of sequences) levels of a suffix tree are useful for detecting whether a candidate signal is real or not. Building a full suffix tree for sequences is not necessary for the problem. Therefore, in Apriori-Motif, we only build the first $l+1$ levels of the suffix tree for a sample. And we attach the (j, k) tuples of all exact copies of a candidate motif and an N-bit string to the corresponding node of the tree. Where (j, k) stands for a substring starting at the k-th position of the j-th sequence. As for an N-bit string attached in a node, the i-th bit is set to 1 if the candidate motif spelled by the path ending at the node occurs in the i-th sequence, otherwise it is set to 0. The (j, k) tuples show the position that a signal appeared in a sample and the N-bit string can be used to count the number of occurrences of a signal conveniently.

The first four levels of the suffix tree are shown in Figure 2 for the DNA sample in Figure 1. For space limitation, we only drew the (j, k) tuples and the N-bit string for the branch 'TATA'. The other

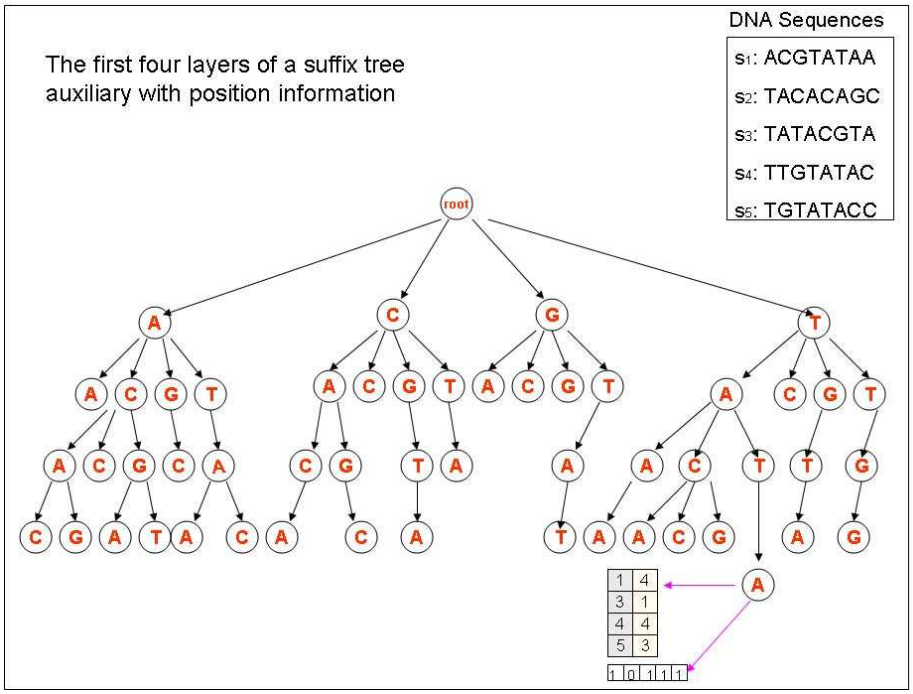

Figure 2: The first four levels of the suffix tree for the sample S={s1, s2, ⋯, s5}

nodes are the same. Where the tuple (1, 4) means that 'TATA' appears in the first sequence and starts from the fourth position ((3, 1), (4, 4), (5, 3) are similar), and the 5-bit string '10111' denotes 'TATA' appears in the first, the third, the fourth and the fifth sequences. In Apriori-Motif, the tree is built by the idea of Bpriori1 algorithm[17] with the growth of the consensus tree. The suffix tree structure makes us search all x-mutated ($x \leqslant d$) copies of a candidate quickly almost without space waste.

Consensus tree structure characterized in the definition 4 is used to store all potential motifs including real signals and candidate but false signals. Since the first d levels of a consensus tree is a full $|\Sigma|$ branches tree, we build a tree from the $(d+1)$-th level. The process of motif finding is just that of constructing the consensus tree for a sample. And the algorithm will end at outputting all real signals (e.g. the signals spelled from the root node to the red nodes of the tree in Figure 2). The details of the algorithm, Apriori-Motif, are shown as follows.

Algorithm: Apriori-Motif

1. Let $i = d+1$.

2. Build the i-th level of the suffix tree for a DNA sample.

3. According to heredity property of a consensus tree, get a node of the tree by breadth first manner in the i-th level.

4. Count the number (denoted by t) of occurrences of x-mutated copies ($x \leqslant d$) of the node from the constructed suffix tree.

4.1 If $t \geqslant q$, the node is permitted to grow in the next level. It is a real node of the tree.

4.2 if $t < q$, the node is forbidden to grow in the next level. It is a virtual node of the tree.

5. $i = i+1$, go to the step 2 until there is no real or virtual node in the current level of the consensus tree.

Since a candidate motif has the downward closure property, it induces the heredity property of a consensus tree characterized in the definition 4. According to the property, a node will grow out a new branch (forming a new and longer candidate) in the next level only if it is a real node and its link node (uncle of the node) has the active branch in the current level. And with the growth of the suffix tree,

the consensus tree is built up layer by layer, synchronously. What we should do just search the partial suffix tree of a sample and count the number of occurrences t of a candidate motif spelled from the consensus tree.

4. Complexity Analysis

In this subsection, we will give a theoretical analysis on the time and the space complexities of the Apriori-Motif algorithm.

The space complexity of the algorithm is composed of two parts. One is the space occupied by the partial suffix tree of a sample. The other is the space expense of the consensus tree for storing all the candidate signals of a sample.

It's easy to known that the number of (j, k) tuples in a partial suffix tree is bound by $NL(l+1)$ because the mumble of i-mer is $N(L-i+1)$ and only the first $l+1$ levels of a suffix tree are useful for searching candidate motifs, where $i \leqslant l$ and l is the maximal length of motifs contained in a sample of sequences. In addition, the space for storing an N-bit string at each node is $O(N/w) = O(1)$ in all cases of our experiments, where w is the length of computer word. Thus, the space complexity is just $O(NLl)$ in worst case for storing the partial suffix tree of a sample. But in real applications, the mumble of the nodes at level i of a suffix tree is far less than $N(L-i+1)$ since all the same i-mers are compressed into one path of the tree.

As for a consensus tree, we only need to keep the three consecutive levels of the tree in main memory in terms of heredity property of the tree. At each level i ($i > d$), since the number of developed i-mer in the suffix tree in worst case is $N(L-i+1)$ and each i-mer produces at most $v(d,i)$ variants (each variant represents a candidate signal), the number of all possible nodes in the i-th level of a consensus tree is no more than $N(L-i+1) \times v(d,i)$. And at each level i, there are at most 4^i nodes in a consensus tree. This makes the space complexity of Apriori-Motif be bound by

$$\begin{aligned} O_{s-Apriori-Motif} &= O(min\{4^l, N \times L \times v(d,l)\}) \\ &\leqslant O(N \times L \times v(d,l)). \end{aligned}$$

But it should be pointed out that although the number of all variants of the nodes in all levels of a suffix tree is bounded by $O(N \times L \times v(d,l))$, only if the variants satisfying the quorum constraint q will become the real nodes of the corresponding consensus tree, otherwise, will be pruned from the tree. Thus, in real applications, the number of nodes in a consensus tree including real nodes and virtual nodes is far less than the complexity bound.

The time complexity of Apriori-Motif is also composed of two parts. One is the time cost for building a partial suffix tree. The other is the time expense at constructing the corresponding consensus tree.

At each level i, we read the nodes of a suffix tree by breadth first manner and partition all attached (j, k) tuples of each node to its son nodes. Thus, the time for building a suffix tree mainly spends at partitioning (j, k) tuples of all nodes. Since there are at most $N(L-i+1)$ tuples at level i, the time complexity for constructing a suffix tree is $O(NLl)$.

Similarly, the time for building a consensus tree mainly costs at detecting all nodes in the consensus tree by breadth first manner to determine whether they are real nodes or virtual nodes. According to the above analysis, in worst case, there are less than $N \times L \times v(d,l)$ nodes in each level i of a consensus tree ($i \leqslant l$). To make a decision for a node in the tree, the corresponding candidate signal will be compared with all nodes in the same level of the suffix tree to count the number of its occurrences. In worst case, there are at most $N(L-i+1)$ comparisons in total for each node of a consensus tree in the level i. Thus, the time complexity of Apriori-Motif is

$$O_{t-Apriori-Motif} = O(N^2 \times L^2 \times l \times v(d,l)).$$

Although the time or the space complexity based on worst case analysis is still high, we should notice the following facts.

1. The number of nodes in a consensus tree is far less than $N \times L \times v(d,l)$ in the level i, $i \leqslant l$.

2. When the tree is grown up to a specified level, the number of branches in the incoming levels will dramatically reduce. The result is against with the theoretical estimation in worst case scenario.

3. The larger q, the less scale the tree is.

4. The heredity property of a consensus tree allows us to prune the tree much quickly.

Thus, the time and the space complexities at worst case analysis are heavily over-estimated in real world. For examples, the main memory used by (15, 4) is beyond 2GB for Bprori2 while it is just no more than 150MB for Apriori-Motif although the two algorithms have the same space complexity in theory. And the time used by Apriori-Motif is almost linear to the maximal length L of background sequences in all cases of the experiments in the next section while it is polynomial (quadratic) time with L in theory.

5. Results and Discussions

We test the performance of Apriori-Motif on some benchmark synthetic samples for PMP under the conditions which we are concerned with. And we compare the algorithm with some other algorithms including brute force algorithm, classical PWM based algorithm PROJECTION[8], Gemoda algorithm [18,19] (the algorithm aims to identify a planted motif also without the restriction of the length of a planted motif, but it needs to specify the window size l' of allowed mutations, $l' \leqslant l$). All experiments are performed on an Intel computer with 2 GHz processor and 2GB main memory. The operating system is Windows XP.

5.1. Benchmark datasets

Similar with the previous work[8,17], the testing samples are generated synthetically in the following steps.

1. A parent motif of length l is chosen by picking l bases from nucleotides A, C, G, T at random.

2. N i.i.d. background sequences of length L are constructed at random.

3. q ($q \leqslant N$) sequences are selected from these N background sequences randomly.

4. Do the following steps for all selected q background sequences.

4.1) Create a mutated copy of the parent motif by randomly choosing d ($d < l$) positions of the motif and mutating these d bases to one of the four nucleotides $\{$A, C, G, T$\}$ at random.

4.2) Select from each background sequence a consecutive substring of length l at random.

4.3) Replace it with the just generated mutated copy of the motif.

In our experiments, the above method is used to generate all the testing (l,d) samples . If not specified, the number N of sequences in a sample is always set to 20 and the length L of a sequence is always set to 600. According to the model of the sequence generation, the real mumble of mismatches is at most d since it is possible that a nucleotide of these chosen bases will be mutated to itself. When we specify the length l of a planted motif and let $q = N$, the problem is just the classical PMP with up to d mismatches. Otherwise, it's a more general and harder problem than PMP.

5.2. Comparison with some other algorithms

Firstly, we compared Apriori-Motif with brute force algorithm, classical PWM based algorithm PROJECTION and Gemoda algorithm on the samples shown in Table 1. Where *s* means second, *min* means minute, *h* denotes hour, and *mon* denotes month. All of them are time units for counting the running time of the algorithms.

Following the previous studies, the accuracy of the algorithms (i.e. *acc* in Table 1) is measure by the *performance coefficient* $\frac{\|K \cap P\|}{\|K \cup P\|}$ defined by Pevzner & Sze[4], where K is the set of known signal positions in a sample and P is the set of positions predicted by the algorithms.

Among the four algorithms, the length l of the planted motif for a sample is given to brute force algorithm and PROJECTION a priori. And Gemoda needs to known the mutated window size l' ($l' = l$, $q = N$ for all cases in Table 1). But we only know the number of maximal mismatches d and $q = N$ in Apriori-Motif.

According to the results in Table 1, the speed of PROJECTION is very fast, but the algorithm can only find the approximate answers. It is just the pros and the cons of a local optimal method that we have discussed in Section 1. And it is well known that PMP is NP-hard. The brute force algorithm,

Table 1: The Performance Comparison on a Range of (l,d)[†]

(l,d)	Brute force		PROJECTION		Gemoda		Apriori-Motif	
	acc	time	*acc*	time	*acc*	time	*acc*	time
(10, 2)	1.00	72*min*	0.82	161.1*s*	1.00	8*min*	1.00	60.109*s*
(11, 2)	-	-	0.95	12.5*s*	1.00	$<$1*min*	1.00	60.235*s*
(12, 3)	-	-	0.71	8.7*min*	1.00	10.5*h*	1.00	15.9*min*
(13, 3)	-	-	0.94	46.0*s*	1.00	10*min*	1.00	15.6*min*
(14, 4)	-	-	0.65	15.4*min*	1.00	$>$3*mon*	1.00	3.368*h*
(15, 4)	-	-	0.90	129.0*s*	1.00	6*h*	1.00	3.134 *h*

which enumerates all potential patterns without using any pruning strategy, will work only when l and d are both small ($l \leqslant 10$ and $d \leqslant 2$). Compared with Gemoda which is designed for solving PMP with a specified mutation window of a planted motif, Apriori-Motif is much more efficient while it is designed for solving PMP without given any information both on the length of a planted motif and the window size of mutations.

Moreover, in all of experiments of Table 1, the longest predicted motifs are just the planted motifs except for (14, 4) sample. For (14, 4) sample, Apriori-Motif reports two predicted motifs with maximal length 14. One of the two predicted motifs is false positive, the other is just the planted one. But the false positive and the real signal have long-range overlaps.

What's more, the larger the ratio d/l, the harder the problem is when the length of the planted motif is specified a priori to an algorithm. Taking (14,4) and (15, 4) as instances, (14, 4) is much harder than (15, 4) since the former has higher noise ratio. But when the length of a signal is not given to an algorithm, (15, 4) might be harder than (14, 4) since we should detect (14, 4) firstly for getting the solution of (15, 4).

Then, we test the influence of the length L of background sequences on Apriori-Motif. The testing results are shown in Figure 3. Where the X-coordinate denotes the length L of background sequences, the Y-coordinate stands for the execution time (the time unit is second for (10, 2), and it is hour for (15,4)). In these group of experiments, the length of sequences is set to 600 bp, 700 bp, 800 bp, 900 bp and 1000 bp, respectively, for all the (l,d) samples tested in Table 1. Since the results of all cases are very similar and the space is limited, we only show the results of (10, 2) and (15, 4).

It's easy to know from Figure 3, the longer the background sequences, the more the running time will be used by Apriori-Motif. And the time complexity is almost linear with the maximal length L of background sequences in the real world. It proofs that the real space used by Apriori-Motif is far less than the theoretical bound which is quadratic time with L.

Also, the parameter q has strong influence on the hardness of the problem. We do not show it since the result is similar with that in Lu & Jia[17]. So does the parameter N.

5.3. *Discussions*

Firstly, Apriori-Motif is a breadth first search method. It can find planted motifs exactly without given the length l. It allows some sequences that may not contain any occurrence of a motif at all and the number of mismatches between an occurrence and the motif can range from 1 to d. Moreover, Apriori-Motif uses the downward closure property to prune the search space. For an example, the signal ACCTA can be extended to a longer candidate ACCTAT only if its substring ACCTA and CCTAT are both real signals. The strategy can speed up the algorithm. Based on our knowledge, all of the existing enumerative algorithms use depth first search strategy, e.g. SPELLER [6] and WEEDER[11]. Although SPELLER and WEEDER can be extended to find planted motifs also under the conditions which

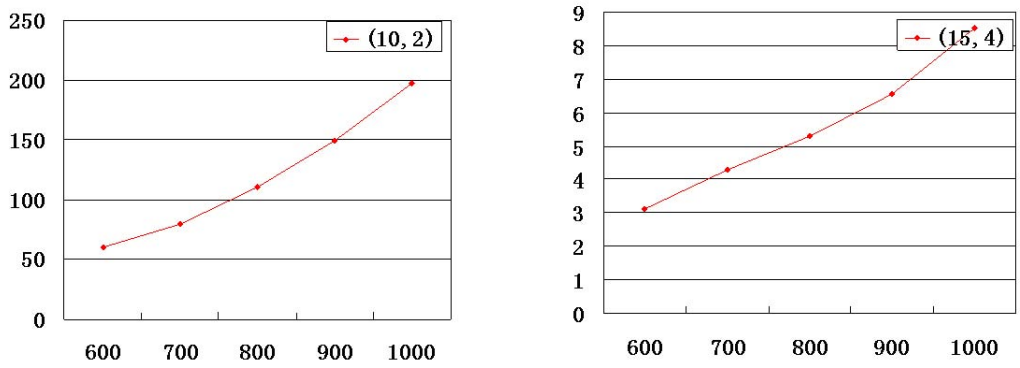

Figure 3: The influence of parameter L on Apriori-Motif

we are concerned with, SPELLER does not use any pruning strategy and WEEDER can only get the approximate answers since WEEDER only allows a mismatch occurred in a block with window size $l\varepsilon$ $(0 < \varepsilon < 1)$ for narrowing down the search space. Although the depth first strategy can save the main memory, Apriori-Motif is a complementary to depth first search methods and can make a trade-off between the time and the space complexities.

Secondly, WEEDER introduces the parameter ε for narrowing down the search space, and lowers down the parameter q for approximating real answers since the lower the quorum constraint q, the more chances the algorithm will find real answers. Similar with WEEDER, we can introduce a parameter e to Apriori-Motif for narrowing down the pattern space, where e is the maximal number of allowed consecutive mismatches between a planted occurrence and the planted motif, since the larger e, the smaller probability the planted occurrence will be appeared in the real world. It's easy to know that the smaller e, the faster Apriori-Motif will be, but the less accurate the result will be. The testing results can be found in Table 2, where we range the parameter e from 1 to d to test its influence on the algorithm.

Thirdly, Apriori-Motif is different from Bpriori series algorithms[17]. Bpriori series algorithms can not efficiently solve the problem like (15, 4). Bpriori2 needs too much space while Bpriori3 needs too much running time for (15, 4). In fact, Apriori-Motif is an improved version of Bprori algorithms. As mentioned above, it makes a trade-off between the space complexity and the time complexity.

In the future, we intend to design an efficient algorithm for solving the dyad planted motif problem since in higher organism transcription factors seldom function in isolation, but act in concert with nearby bound factors in a combinatorial manner to induce specific regulatory behaviors[22].

6. Conclusions

In the study, we presented an algorithm, Apriori-Motif, based on the techniques in frequent pattern mining. It is a breadth first search method, and can prune the search space quickly by the downward closure property of motifs. Similar with Apriori, a classical algorithm for mining frequent pattern, Apriori-Motif can find motifs from length $d+1$ to l iteratively and terminate at the length l of a planted motif. So it can find any planted motif without given the length l a priori. Moreover, it allows some sequences that may not contain any occurrence of a planted motif with up to d mismatches. Both theoretical analysis and experimental tests have shown the good performance of the algorithm.

Acknowledgments

This work was supported in part by NSFC (Grant No. 60905029, 60875031 and 90820013),

Table 2: The Influence of parameter *e* on Apriori-Motif

(l,d)	*e*=1		*e*=2		*e*=3		*e*=4	
	acc	time	*acc*	time	*acc*	time	*acc*	time
(10, 2)	0.7	47.140s	1.0	60.109s	-	-	-	-
(11, 2)	1.0	50.157s	1.0	60.235s	-	-	-	-
(12, 3)	0.66667	5.169min	0.58333	12.765min	1.00	15.908min	-	-
(13, 3)	0.76923	4.954min	1.0	14.320min	1.00	15.612min	-	-
(14, 4)	0.64285	12.948min	0.92857	1.820h	1.00	2.840	1.00	3.368h
(15, 4)	0.60	10.643min	0.80	1.945h	1.00	2.914h	1.00	3.134 h

973 Project (Grant No. 2007CB311002 and 2009CB320701).

References

1. M. Tompa, "Assessing computational tools for the discovery of transcription factor binding sites," *Nature Biotechnology*, **23**, 137–144 (2005)
2. J. Hu, B. Li, D. Kihara, "Limitations and potentials of current motif discovery algorithms," *Nucleic Acids Research* **33(15)**, 4899–4913 (2005)
3. M. K. Das, H. K. Dai, "A survey of DNA motif finding algorithms," *BMC Bioinformatics*, **8(suppl 7)** (2007)
4. P. Pevzner, S. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, California, USA, 269–278 (2000)
5. U. Keich, P. Pevzner, "Subtle motif: defining the limits of finding algorithms," *Bioinformatics*, **18(10)**, 1382–1390 (2002)
6. M. F. Sagot, "Spelling approximate repeated or common motifs using a suffix tree," *Proceedings of LATIN'98: Theoretical Informatics*, LNCS 1380, 111–127 (1998)
7. E. Eskin, P. Pevzner, "Finding composite regulatory patterns in DNA sequences," *Bioinformatics*, 354–363 (2002)
8. J. Buhler, M. Tompa, "Finding motifs using random projections," *Proceedings of The Fifth Annual International Conference Computational Molecular Biology*, Canada, ACM Press (2001)
9. A. Price, S. Ramabhadran, P. Pevzner, "Finding subtle motifs by branching from sample string," *Bioinformatics*, **2**, 1–7 (2003)
10. P. A. Evans, A. D. Smith, "Toward optimal motif enumeration," *Proceedings of Algorithms and Data Structures, 8th International Workshop*, 47–58 (2003)
11. G. Pavesi, G. Mauri, G. Pesole, "An algorithm for finding signals of unknown length in DNA sequences," *Bioinformatics*, **17**, 207–214 (2001)
12. J. Davila, S. Balla, S. Rajasekaran, "Fast and practical algorithms for planted (*l*, *d*) motif search," *IEEE/ACM Trans. On Computational Biology and Bioinformatics*, **4**, 544–552 (2007)
13. Y. L. Chin, C. M. Leung, "Voting algorithms for discovering long motifs," *Proceedings of the Third Asia-Pacific Bioinformatics Conference*, Singapore, 261–271 (2005)
14. C. M. Leung, Y. L. Chin, "An efficient algorithm for the extended (*l*, *d*)-motif problem with unknown number of binding sites," *Proceedings of the Fifth IEEE Symposium on Bioinformatics and Bioengineering*, 11–18 (2005)
15. N. Pisanti, A. M. Carvalho et al, "RISOTTO : fast extraction of motifs with mismatches," *Proceedings of the Seventh Latin Am. Theoretical Informatics Symp.*, 757–768 (2006)
16. C. E. Lawrence, S. F. Altschul et al, "Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment," *Science* **262**, 208–214 (1993)
17. R. Q. Lu, C. Y. Jia et al, "An exact data mining method for finding center strings and all their instances," *IEEE Trans. On Knowledge and Data Engineering*, **19(4)**, 509–522 (2007)
18. M. P. Styczynski, K. L. Jensen, "An extension and novel solution to the (*l*, *d*)-motif challenge problem," *Genome Informatics*, **15**, 63–71 (2004)
19. K. L. Jensen, M. P. Styczynski et al, "A generic motif discovery algorithm for sequential data," *Bioinformatics*, **22**, 21–28 (2006)
20. R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago de Chile, Chile, 487–499 (1994)
21. E. Ukkonen, "Constructing suffix trees on-line in linear time," *Proceedings of the Information Processing*, 1992:484–492.
22. K. Klepper, G. Sandve, "Assessment of composite motif discovery methods," *BMC Bioinformatics*, **9(123)** (2008).