Reference Sequence Selection for Motif Searches

Qiang Yu[1], Hongwei Huo[1,*], Ruixing Zhao[1], Dazheng Feng[2], Jeffrey Scott Vitter[3] and Jun Huan[3]
[1] School of Computer Science and Technology, Xidian University, Xi'an, 710071, China
[2] School of Electronic Engineering, Xidian University, Xi'an, 710071, China
[3] Department of Electrical Engineering & Computer Science, The University of Kansas, Lawrence, 66047, USA
Email: {qyu, hwhuo}@mail.xidian.edu.cn, rxzhao@stu.xidian.edu.cn, dzfeng@xidian.edu.cn, {jsv, jhuan}@ittc.ku.edu.

***Abstract*—The planted (*l*, *d*) motif search (PMS) is an important yet challenging problem in computational biology. Pattern-driven PMS algorithms usually use *k* out of *t* input sequences as reference sequences to generate candidate motifs, and they can find all the (*l*, *d*) motifs in the input sequences. However, most of them simply take the first *k* sequences in the input as reference sequences without elaborate selection processes, and thus they may exhibit sharp fluctuations in running time, especially for large alphabets.**

In this paper, we build the reference sequence selection problem and propose a method named RefSelect to quickly solve it by evaluating the number of candidate motifs for the reference sequences. RefSelect can bring a practical time improvement of the state-of-the-art pattern-driven PMS algorithms. Experimental results show that RefSelect (1) makes the tested algorithms solve the PMS problem steadily in an efficient way, (2) particularly, makes them achieve a speedup of up to about 100× on the protein data, and (3) is also suitable for large data sets which contain hundreds or more sequences.

Keywords—*Planted (l, d) motif search; pattern-driven; reference sequences*

I. INTRODUCTION

Motif discovery is a fundamental problem in computational biology. It is mainly used to locate significant sequence fragments in biological sequences, such as transcription factor binding sites in DNA sequences [1] and short linear motifs in protein sequences [2].

The planted (l, d) motif search (PMS) [3] is a famous formulation for motif discovery: given a data set $D = \{s_1, s_2, \ldots, s_t\}$ with t n-length sequences over an alphabet Σ, q satisfying $0 < q \leq t$, and l and d satisfying $0 \leq d < l < n$, the goal is to find one or more l-length strings m such that m occurs in at least q sequences in D with up to d mismatches. The string m is called a (l, d) motif, and each occurrence of m is called a motif instance.

There have been numerous motif discovery algorithms [4]. They are either approximate or exact, based on whether the algorithm can always find all motifs or the optimum motif. In this paper, we mainly focus on exact motif discovery algorithms, which can find all (l, d) motifs by traversing the whole search space. The main indicator to assess exact algorithms is time performance. For the exact algorithms proposed in earlier years, such as WINNOWER [3], their search space is composed of $(n - l + 1)^t$ possible alignments of motif instances. In recent years, the exact algorithms verify all the l-length patterns in the $O(|\sum|^l)$ search space, and they are called the pattern-driven PMS algorithms [5-13].

The pattern-driven PMS algorithms have better time performance than other exact algorithms so far in identifying both short motifs and long motifs with weak signal. Their basic idea is to generate candidate motifs by using several reference sequences in the input, and then verify each candidate motif one by one. Specifically, they generate candidate motifs by using all possible h-tuple $T = (x_1, x_2 \ldots x_h)$ composed of h l-length strings coming from h distinct reference sequences. In existing pattern-driven PMS algorithms, h is 1 for PMSP [5] and PMSPrune [6]; h is 2 for PairMotif [7], qPMS7 [8] and TravStrR [9]; h is 3 for iTriplet [10] and PMS5 [11]; for PMS8 [12] and qPMS9 [13], h is greater than or equal to 3 and self-adaptive in dealing with different PMS problem instances. Moreover, these algorithms use $k = t - q + h$ reference sequences to generate candidate motifs, ensuring that there exists at least one h-tuple T so that each l-length string in T is a motif instance.

Although pattern-driven PMS algorithms outperform other exact algorithms, most of them use the first k sequences in the input as reference sequences, without considering the effect of different reference sequences on time performance. In this study we found that given a data set, different reference sequences may lead to quite different number of candidate motifs, especially for large alphabets. So, in dealing with different inputs with the same scale, the pattern-driven PMS algorithms may exhibit sharp fluctuations in running time. For instance, we randomly generate multiple groups of data sets with $|\sum| = 20$, $t = 20$, $q = 20$ and $n = 600$ following the method described in Section IV.B. When solving the (19, 9) problem instance, qPMS7 sometimes consumes 6.1 minutes, but sometimes over 48 hours. Some other pattern-driven PMS algorithms like TravStrR and PMS8, suffer from the same problem (see Supplement 1 for more examples).

To solve this problem, we propose a method named RefSelect to quickly select reference sequences that generate a small number of candidate motifs. RefSelect can bring a practical time improvement of the state-of-the-art pattern-driven PMS algorithms, without doing any modifications to them.

II. PROBLEM DESCRIPTION AND NOTATIONS

Reference sequence selection problem: Given a data set $D = \{s_1, s_2, \ldots, s_t\}$ over an alphabet Σ that contains t sequences of length n, the (l, d) problem instance ($0 \leq d < l < n$) and the number of reference sequences k ($0 < k < t$) required by the pattern-driven PMS algorithms, the task is to select k reference sequences from D to form the reference sequence set D', such that when using D' the pattern-driven PMS algorithms can efficiently solve the (l, d) problem instance without sharp fluctuations in running time.

Table 1 summarizes the notations used in this paper. Notice that, a sequence specially refers to an n-length string in a data set, and an l-mer refers to a short string of length l ($l < n$).

* Corresponding author. The executable program of RefSelect and all supplements are available at https://sites.google.com/site/feqond/refselect.

Table 1. Notations used in this paper

Notation	Explanation
$\|x\|$	The length of a string, the size of a set, or the number of elements in a matrix.
D, D'	D is the set of input sequences. D' is the set of reference sequences. $D = \{s_1, s_2, \ldots, s_t\}$ and $D' = \{s_{r1}, s_{r2}, \ldots, s_{rk}\}$, satisfying $D' \subset D$.
t	The number of sequences in the input sequence set D, namely $\|D\| = t$.
k	The number of required reference sequences, namely $\|D'\| = k$.
n	The length of each input sequence.
$x \in_l s$	The string x is an l-length substring of the sequence s. In other words, x is an l-mer in the sequence s.
$s[i]$	The ith character in the string s.
$s[i..j]$	A substring of the string s starting from the ith position to the jth position.
$d_H(x, x')$	The Hamming distance between two strings x and x' of the same length.
$M_d(x, x')$	The common candidate motifs of two l-mers x and x'. $M_d(x, x') = \{y: \|y\| = \|x\| = \|x'\|, d_H(y, x) \le d, d_H(y, x') \le d\}$.
$N_r(D')$	The number of candidate motifs generated from the reference sequences set D', calculated by (1).
$N_r(s_i, s_j)$	The number of candidate motifs generated from two sequences s_i and s_j, calculated by (2).
$\min(i, j)$	The minimum value between two integers i and j. $\min(i, j) = i$ if $i \le j$, j otherwise.
$sim(s_i, s_j)$	The similarity of two sequences s_i and s_j.

III. METHODS

A. Overview

We introduce why and how to select reference sequences for the pattern-driven PMS algorithms. Let us consider the following two observations, which indicate how the Hamming distance between pairs of l-mers affects the number of candidate motifs. Examples and detailed discussion for the two observations are given in Section IV.A.

Observation 1. For two l-mers x and x', the smaller their Hamming distance $d_H(x, x')$, the larger the number of their common candidate motifs $|M_d(x, x')|$.

Observation 2. For a tuple T of h l-mers, when it contains pairs of l-mers with a relatively small Hamming distance, it generates a relatively large number of candidate motifs.

Based on the two observations, different reference sequences may lead to different number of candidate motifs. The pattern-driven PMS algorithms utilize all tuples of h l-mers in k ($0 < h \le k$) reference sequences to generate candidate motifs. Once there are relatively more pairs of l-mers with small Hamming distance in these h-tuples, more candidate motifs will be generated.

Since the time performance of pattern-driven PMS algorithms mainly depends on the number of generated candidate motifs, we should select the reference sequence set generating a small number of candidate motifs. For example, as shown in Fig. 1, assume the input sequence set D is $\{s_1, s_2, s_3, s_4\}$ where each sequence has two l-mers and we select $k = 3$ reference sequences from D. In the figure, the thicker the dotted line, the more candidate motifs are generated by the associated two l-mers. Obviously, $\{s_2, s_3, s_4\}$ is the optimal reference sequence set.

Naturally, we select reference sequences by evaluating the number of generated candidate motifs, ensuring the selected

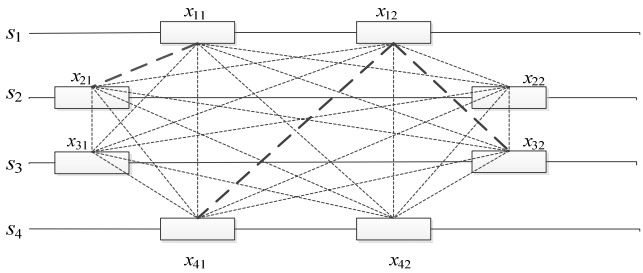

Fig.1. An example of selecting reference sequences

reference sequences generate a small number of candidate motifs. When we evaluate the number of candidate motifs generated from a tuple T of h l-mers, it is difficult to directly compute the number of common candidate motifs shared by all the h l-mers in T, denoted by N_1. An alternative way is to compute the sum of the number of common candidate motifs shared by each pair of l-mers in T, denoted by $N_2 = \Sigma|M_d(x_i, x_j)|$ for $1 \le i < j \le h$. As shown in Fig. 2 (Section IV.A), N_1 and N_2 have the consistent tendency with the variation of the Hamming distance between the pairs of l-mers in T.

Furthermore, we use (1) and (2) to evaluate $N_r(D')$, the number of candidate motifs generated from the reference sequence set D'. It is defined as the sum of the number of common candidate motifs shared by each pair of l-mers in D' (i.e., in every two sequences in D').

$$N_r(D') = \sum_{1 \le i < j \le |D'|} N_r(s_i, s_j) \quad (1)$$

$$N_r(s_i, s_j) = \sum_{x \in_l s_i, x' \in_l s_j} |M_d(x, x')| \quad (2)$$

Our method of selecting reference sequences includes two steps. The first step is to compute the number of candidate motifs generated from every two sequences in D, in order to quickly evaluate the number of candidate motifs generated from the specified k reference sequences in the next step. The second step is to select k sequences from D to form the reference sequence set D' such that the number of candidate motifs generated from D' is as small as possible. In the following, we describe the two steps in detail.

B. Step 1: Computing the Number of Candidate Motifs

We compute the number of candidate motifs generated from every two sequences s_i and s_j in D according to (2). For an l-mer x in s_i and an l-mer x' in s_j, the number of their common candidate motifs $|M_d(x, x')|$ depends on their Hamming distance $d_H(x, x')$ [7]. The details about how to compute $|M_d(x, x')|$ are described in [7]. In implementing RefSelect, we store the values of $|M_d(x, x')|$ under different $d_H(x, x')$ in a table in advance. Once we know $d_H(x, x')$, we can immediately get $|M_d(x, x')|$ by looking up the table in $O(1)$ time.

Thus, the core operation of (2) is to compute the Hamming distance between every two l-mers $x \in_l s_i$ and $x' \in_l s_j$. For any two sequences of length n, we have $O(n^2)$ pairs of l-mers. A simple method is to traverse all these pairs of l-mers; for each pair of l-mers x and x', the Hamming distance can be computed in $O(l)$ time by comparing the characters $x[i]$ and $x'[i]$ for $1 \le i \le l$. The time complexity of this method is $O(ln^2)$.

We introduce a more efficient method to compute the Hamming distance between every pair of l-mers in s_i and s_j.

We fill an $n \times n$ matrix M, where the element in row a ($1 \le a \le n$) and column b ($1 \le b \le n$) is denoted by $M[a, b]$. Let $l_{min} = min(a, b)$, $str_1 = s_i[a - l_{min} + 1 \ldots a]$, $str_2 = s_j[b - l_{min} + 1 \ldots b]$; then, $M[a, b]$ is the number of such position i that $str_1[i] = str_2[i]$ for $1 \le i \le l_{min}$, namely $M[a, b] = l_{min} - d_H(str_1, str_2)$.

In filling the matrix M, we initialize $M[a, b]$ with 0 for the case of $min(a, b) = 0$, and obtain $M[a + 1, b + 1]$ based on $M[a, b]$:

$$M[a+1,b+1] = \begin{cases} M[a,b]+1, & \text{if } s_i[a+1] = s_j[b+1] \\ M[a,b], & \text{otherwise} \end{cases}, \quad (3)$$

where both a and b range from 0 to n.

With the matrix M, we use (4) to compute the Hamming distance between a pair of l-mers str_1' and str_2', where $str_1' = s_i[a - l + 1 \ldots a]$ and $str_2' = s_j[b - l + 1 \ldots b]$ are the l-mers at the position a ($a \ge l$) of s_i and the position b ($b \ge l$) of s_j, respectively.

$$d_H(str_1', str_2') = l - (M[a,b] - M[a-l,b-l]) \quad (4)$$

Our method is mainly to fill the matrix M. That is, we need to compute n^2 elements one by one for any two n-length sequences s_i and s_j. In computing each element $M[a, b]$ by (3) in $O(1)$ time, we simultaneously compute the Hamming distance between the l-mer at position a ($a \ge l$) of s_i and that at position b ($b \ge l$) of s_j by (4) in $O(1)$ time. Therefore, the time complexity of computing the Hamming distance for all pairs of l-mers in two sequences is reduced to $O(n^2)$.

C. Step 2: Selecting Reference Sequences

After getting the number of candidate motifs generated from every two sequences in D, we can evaluate the number of candidate motifs generated from a set of reference sequences according to (1). In this section, we introduce how to select a set of reference sequences that generates a small number of candidate motifs. The exhaustive search strategy is to traverse every possible reference sequence set D' and then report the set generating the smallest number of candidate motifs. However, its computational time increases dramatically with the increase of the number of input sequences t and the number of selected reference sequences k.

In order to quickly select reference sequences, we convert the problem to graph clustering. The t sequences in D are taken as t nodes in a graph. The similarity between two nodes s_i and s_j is related to the number of candidate motifs generated from s_i and s_j (i.e., $N_r(s_i, s_j)$). We hope that the nodes corresponding to the reference sequences with small number of candidate motifs form a dense subgraph, and they belong to the same cluster after graph clustering. We use the MCL algorithm [14] to complete clustering. We describe the details involved in clustering as follows.

1) Similarity Measure

We design the similarity of two nodes (sequences) s_i and s_j based on $N_r(s_i, s_j)$. Simultaneously, we consider the following two factors.

First, we further increase the effect of the pairs of l-mers in s_i and s_j with small Hamming distance on the total number of candidate motifs generated from s_i and s_j. By doing this, it is helpful for the clustering process to distinguish the different reference sequence sets that lead to different number of candidate motifs. Specifically, we use (5) instead of (2) to evaluate the number of candidate motifs generated from two sequences s_i and s_j.

$$N_r'(s_i,s_j) = \sum_{x \in_l s_i, x' \in_l s_j} \frac{|M_d(x,x')|}{d_H(x,x')+1} \quad (5)$$

Second, we aim to put a set of sequences D' to the same cluster such that every two sequences s_i and s_j in D' generate a small number of candidate motifs. So, we should ensure that s_i and s_j have a larger similarity when they generate a smaller number of candidate motifs. Finally, we compute the similarity of s_i and s_j as follows:

$$sim(s_i,s_j) = \frac{1}{N_r'(s_i,s_j)} \times \max_{1 \le i < j \le t} N_r'(s_i,s_j). \quad (6)$$

2) Cluster Refinement

The clustering process may produce more than one cluster, and there may not be exact k nodes in each cluster. We refine each obtained cluster C in order to get a set of k reference sequences. Then, we sort the sets of reference sequences and output the set with the highest score.

For the cluster C with only one node, we take it as an invalid cluster, since the node in C has a low similarity with other nodes. For the cluster C with two or more nodes, it corresponds to three cases: (a) there are exact k nodes in C; (b) there are more than k nodes in C; (c) there are less than k nodes in C. For Case (a), we can get the reference sequence set D' directly by using the k sequences in C. Next, we introduce how to refine C under Cases (b) and (c).

For Case (b), we use greedy strategy to select k sequences from C ($|C| > k$) to form D'. First, we initialize D' with $\{s_a, s_b\}$ such that $sim(s_a, s_b) = \max\{sim(s_i, s_j)\}$ for all $s_i, s_j \in C$ and $s_i \ne s_j$. Then, we repeatedly choose a node s_r satisfying (7) from $C - D'$ and add it to D' until $|D'| = k$.

$$s_r = \arg\max_{s_i \in C - D'} \sum_{s_j \in D'} sim(s_i, s_j) \quad (7)$$

For Case (c), we use the similar method to choose $k - |C|$ nodes from $D - C$, and add them to C to form D'. First, D' is initialized with C. Then, we repeatedly choose a node s_r satisfying (8) from $D - D'$ and add it to D' until $|D'| = k$.

$$s_r = \arg\max_{s_i \in D - D'} \sum_{s_j \in D'} sim(s_i, s_j) \quad (8)$$

We describe how to refine a cluster C in Algorithm 1. Because the process that we select reference sequences by using greedy strategy is similar to the Prim algorithm for computing minimum spanning tree, the time complexity of Algorithm 1 is $O(|C|^2 \lg|C|)$ and $O((t - |C|)^2 \lg(t - |C|))$ under Case (b) and Case (c), respectively.

Algorithm 1 Cluster Refinement
Input: C, D, k
Output: D'
1: **if** $|C| = 1$ **then**
2: $D' \leftarrow \Phi$ // Φ is an empty set
3: **if** $|C| = k$ **then**
4: $D' \leftarrow C$
5: **if** $|C| > k$ **then**
6: initialize D' with $\{s_a, s_b\}$ such that $sim(s_a, s_b) = \max\{sim(s_i, s_j)\}$ for $s_i, s_j \in C$ and $s_i \ne s_j$
7: **while** $|D'| < k$ **do**
8: select a node s_r from $C - D'$ according to (7)
9: add s_r to D'

10: **else**
11: initialize D' with C
12: **while** $|D'| < k$ **do**
13: select a node s_r from $D - D'$ according to (8)
14: add s_r to D'
15: **return** D'

After cluster refinement, if we obtain more than one reference sequence set D', we score each D' by (9), and then output the D' with the highest score.

$$score(D') = \sum_{s_i \in D', s_j \in D', i \neq j} sim(s_i, s_j) \quad (9)$$

D. Whole Algorithm

This section gives the whole algorithm of RefSelect.

Algorithm 2 RefSelect

Input: D, l, d, k
Output: D'
1: $D' \leftarrow \Phi$
2: **for** $i \leftarrow 1$ to $|D|$ **do**
3: **for** $j \leftarrow i + 1$ to $|D|$ **do**
4: compute $sim(s_i, s_j)$ according to (6)
5: $sim(s_j, s_i) \leftarrow sim(s_i, s_j)$
6: cluster the sequences in D by using the MCL algorithm
7: **for** each obtained cluster C **do**
8: refine C by using Algorithm 1 and obtain a set of reference sequences D''
9: **if** $score(D'') > score(D')$ **do**
10: $D' \leftarrow D''$
11: **return** D'

In line 1 of the pseudocode, we initialize D' with an empty set. Lines 2 to 5, corresponding to the first step of RefSelect, compute the similarity of any two nodes (sequences). As described in III.B, the core operation of this step is to compute the Hamming distance from all l-mers in s_i to all l-mers in s_j in $O(n^2)$ time, for any two sequences s_i and s_j in D. Therefore, the time complexity of this step is:

$$O\left(\binom{t}{2} \times n^2\right).$$

Lines 6 to 10, corresponding to the second step of RefSelect, cluster the t sequences in D by using the MCL algorithm and refine each obtained cluster. The time complexity of clustering is $O(t^3)$. The time complexity of refining clusters, which is described in Section III.C.2, is negligible with respect to the time complexity of the first step. So, the time complexity of RefSelect is:

$$O\left(\binom{t}{2} \times n^2 + t^3\right).$$

In executing RefSelect, we need $O(tn)$ space to store the input sequence set D, $O(n^2)$ space to store the matrix M for computing Hamming distance, and $O(t^2)$ space to store the similarity matrix of t input sequences. So, the space complexity of RefSelect is $O(tn + n^2 + t^2)$.

IV. Results and Discussion

A. Effect of Hamming Distance on Candidate Motif Number

First, we consider the case of two l-mers x and x', and analyze the effect of their Hamming distance $d_H(x, x')$ on the number of their common candidate motifs $|M_d(x, x')|$. Table 2 gives the values of $|M_d(x, x')|$ with $d_H(x, x')$ varying from 0 to $2d$ for both the DNA and protein data. In the table, the values are obtained by using challenging PMS problem instances [8]. We can find that $|M_d(x, x')|$ increases with the decrease of $d_H(x, x')$ for both the DNA and protein data.

Table 2. The effect of Hamming distance on the candidate motif number for a pair of l-mers

$i = d_H(x, x')$	$\lvert M_d(x, x')\rvert$	
	***DNA data*, (*l*, *d*) = (17, 6)**	***protein data*, (*l*, *d*) = (17, 8)**
16	-	1.29×10^4
15	-	9.40×10^5
14	-	2.83×10^7
13	-	4.61×10^8
12	9.24×10^2	4.51×10^9
11	6.47×10^3	2.85×10^{10}
10	2.36×10^4	1.27×10^{11}
9	6.35×10^4	4.38×10^{11}
8	1.41×10^5	1.25×10^{12}
7	2.79×10^5	3.14×10^{12}
6	5.04×10^5	7.09×10^{12}
5	8.52×10^5	1.48×10^{13}
4	1.39×10^6	2.95×10^{13}
3	2.14×10^6	5.68×10^{13}
2	3.44×10^6	1.09×10^{14}
1	4.90×10^6	2.12×10^{14}
0	1.07×10^7	4.31×10^{14}

Second, we consider the case of h ($h > 2$) l-mers containing pairs of l-mers with different Hamming distance, and analyze the effect of Hamming distance on the number of common candidate motifs shared by the h l-mers. In our example, we set h as 3 and the three l-mers x_1, x_2 and x_3 can form three pairs of l-mers; then, we fix $d_H(x_1, x_2) = 2d - 2$ and vary $d_H' = (d_H(x_1, x_3) + d_H(x_2, x_3)) / 2$ from $2d - 2$ to $2d - 7$. Fig. 2(a) and 2(b) give the tendency of the number of common candidate motifs $|M_d(x_1, x_2, x_3)|$ in contact with the decrease of d_H' on (19, 7) problem instance for the DNA data and (19, 9) problem instance for the protein data, respectively. The y-axis is in log-scale. We can see that no matter for the DNA or protein data, $|M_d(x_1, x_2, x_3)|$ increases with the decrease of d_H'. In other words, when h ($h > 2$) l-mers contain some pairs of l-mers with a relatively small Hamming distance, they generate a relatively large number of candidate motifs. Also, the tendency of $|M_d'(x_1, x_2, x_3)| = |M_d(x_1, x_2)| + |M_d(x_1, x_3)| + |M_d(x_2, x_3)|$ is given in Fig. 2. Both $|M_d'(x_1, x_2, x_3)|$ and $|M_d(x_1, x_2, x_3)|$ increase with the decrease of d_H', namely they have the consistent tendency.

B. Results on Practical Time Improvement of PMS Algorithms

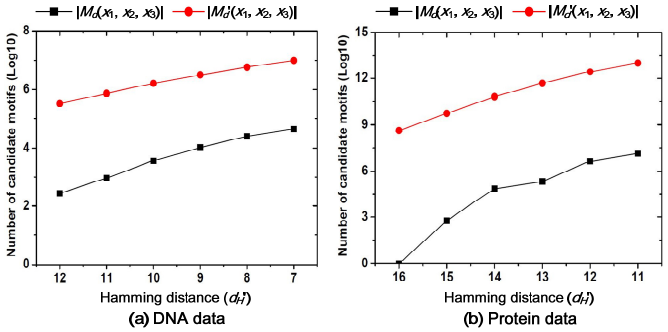

Fig.2. The effect of Hamming distance on the candidate motif number for three l-mers.

In this section we check the validity of RefSelect as follows. First, we use RefSelect to select k reference sequences from the given t input sequences, and adjust the order of the t input sequences by preposing the k sequences; RefSelect is implemented in C++ and its running time is denoted by T_{rs}. Second, we test pattern-driven PMS algorithms on the input sequences of original order and that of new order, obtaining the running time T_1 and T_2, respectively. Finally, we compare T_1 with $T_{rs} + T_2$.

Three pattern-driven PMS algorithms qPMS7 [8], TravStrR [9] and PMS8 [12] are chosen to participate in the test. They are all newly proposed algorithms and outperform the previous exact algorithms on challenging instances. Notice that qPMS9 [13] is also a newly proposed PMS algorithm with good time performance; we do not choose it as a tested algorithm, because it does not use fixed k reference sequences to obtain h-tuples. All the tested algorithms are executed on a 2.67 GHz single core and a 4 Gbyte Memory, except for PMS8, which is executed on a 16-core platform in solving the (21, 10) and (23, 11) instances of the protein data.

In the experiments, we generate data sets following [3]. First, we randomly generate t sequences of length n and a motif m of length l, and randomly choose q ($0 < q \leq t$) out of the t sequences; then, for each of the q sequences, we generate a random motif instance m' that differs from m in at most d positions, and implant m' into a random position of the sequence. For each specific test instance, we generate five data sets to get an average result.

First, we fix $t = 20$, $n = 600$ and $q = 20$, and give in Tables 3 and 4 the results on challenging instances of the protein and DNA data, respectively. For qPMS7 and TravStrR, k is set as 2, while for PMS8 k is set dynamically under different (l, d) instances according to [12]. We find that:

(1) RefSelect can make the tested algorithms solve the PMS problem steadily in an efficient way. For example, for the (19, 9) problem instance in Table 3, the minimum and maximum running time of qPMS7 are reduced to 140.00s and 287.00s from 820.00s and 37121.00s after using the reference sequences selected by RefSelect.

(2) The speedup on the protein data is significantly larger than that on the DNA data. We give the explanation by using Table 2. The fact that Pattern-driven PMS algorithms sometimes show poor performance is mainly caused by the pairs of l-mers with relatively small Hamming distances; these l-mers can generate more candidate motifs. The larger the difference between the number of candidate motifs for the pairs of l-mers with relatively small Hamming distance and that for the pairs of l-mers with relatively large Hamming

Table 3. Running time and speedup for the protein data

(l, d)		qPMS7			TravStrR			PMS8		
		T_1	$T_{rs} + T_2$	*speedup*	T_1	$T_{rs} + T_2$	*speedup*	T_1	$T_{rs} + T_2$	*speedup*
(15, 7)	min	93.00s	85.00s	6.58	5.80s	5.63s	1.00	2.21s	3.64s	1.89
	average	722.40s	109.80s		7.51s	7.49s		288.33s	178.43s	
	max	2596.00s	122.00s		8.89s	8.74s		1126.80s	544.00s	
(17, 8)	min	131.00s	120.00s	7.68	11.88s	8.98s	4.03	390.13s	8.52s	5.85
	average	1204.40s	156.80s		76.98s	19.10s		1647.89s	281.83s	
	max	3742.00s	214.00s		207.32s	54.01s		3092.82s	719.78s	
(19, 9)	min	820.00s	140.00s	86.22	11.58s	11.25s	76.38	189.07s	17.11s	2.21
	average	18502.20s	214.60s		1591.67s	20.84s		14619.67s	6629.83s	
	max	37121.00s	287.00s		4355.44s	30.23s		53376.12s	17235.40s	
(21, 10)	min	191.00s	161.00s	> 118.94	22.47s	22.28s	61.63	18268.00s	3022.00s	> 9.65
	average	-o	671.40s		1745.09s	24.25s		-o	17905.40s	
	max	-o	1783.00s		6453.27s	53.06s		-o	31054.00s	
(23, 11)	min	53978.00s	242.00s	> 34.77	44.30s	12.19s	15.56	6327.12s	1608.23s	> 12.89
	average	-o	4970.00s		3033.99s	195.04s		-o	13409.80s	
	max	-o	17141.00s		10909.72s	378.31s		-o	50421.00s	

s: seconds; -o:over 48 hours; T_1 and T_2: running time of a PMS algorithm on the input sequences of original order and new order; T_{rs}: running time of RefSelect; min, average and max: the minimum, average and maximum running time on five data sets; speedup: average T_1 / average $T_{rs} + T_2$.

Table 4. Running time and speedup for the DNA data

(l, d)		qPMS7			TravStrR			PMS8		
		T_1	$T_{rs} + T_2$	*speedup*	T_1	$T_{rs} + T_2$	*speedup*	T_1	$T_{rs} + T_2$	*speedup*
(15, 5)	min	149.00s	128.00s	1.10	68.34s	64.46s	1.03	38.51s	37.73s	1.04
	average	173.00s	157.67s		73.84s	71.61s		65.43s	63.21s	
	max	220.00s	196.00s		91.96s	85.52s		113.16s	106.54s	
(17, 6)	min	557.00s	504.00s	1.08	230.41s	185.54s	1.03	244.95s	216.00s	1.04
	average	660.40s	611.80s		263.39s	255.38s		387.80s	371.42s	
	max	884.00s	756.00s		322.78s	319.74s		601.29s	601.20s	
(19, 7)	min	2520.00s	2357.00s	1.06	1009.06s	970.67s	1.05	1105.88s	1092.14s	1.04
	average	2911.00s	2737.20s		1116.18s	1060.56s		1834.32s	1753.06s	
	max	3846.00s	3553.00s		1281.81s	1264.79s		3064.09s	2811.09s	
(21, 8)	min	12144.00s	11343.00s	1.03	3403.84s	3168.97s	1.00	7997.65	7896.77s	1.13
	average	12791.40s	12444.00s		4377.85s	4357.19s		12494.79s	11102.56s	
	max	13701.00s	14029.00s		5306.58s	5301.84s		14243.2s	11395.31s	
(23, 9)	min	61512.00s	61561.00s	1.07	14456.36s	14184.14s	1.00	23984.70s	22081.70s	1.11
	average	68741.60s	64022.60s		18234.85s	18157.41s		38705.98s	34847.10s	
	max	77427.00s	68709.00s		21315.48s	22094.73s		61160.00s	61278.10s	

s: seconds; -o:over 48 hours; T_1 and T_2: running time of a PMS algorithm on the input sequences of original order and new order; T_{rs}: running time of RefSelect; min, average and max: the minimum, average and maximum running time on five data sets; speedup: average T_1 / average $T_{rs} + T_2$.

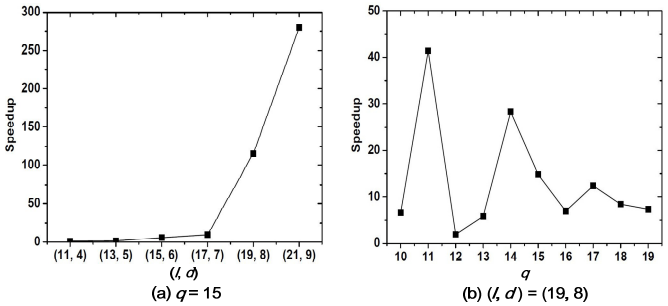

Fig.3. The speedup brought by RefSelect for qPMS7 in the case of $q < t$.

distance, the larger speedup can be achieved. As can be seen from Table 2, the difference on the protein data (large alphabet) is significantly larger than that on the DNA data (small alphabet).

Second, we discuss the case of $q < t$ by fixing $|\Sigma| = 20$ (protein data), $t = 20$ and $n = 600$. In the above three algorithms, we only choose qPMS7 as the tested algorithm because PMS8 cannot solve the PMS problem with $q < t$ and TravStrR usually quits unexpectedly in our test environment. We select $t - q + 2$ reference sequences for qPMS7, and test it by using the same (l, d) instances with [8]. On the one hand, we set q as 15, and test qPMS7 on different (l, d) instances; as shown in Fig. 3(a), RefSelect makes qPMS7 perform better and the speedup increases with the increase of l and d. On the other hand, we fix (l, d) = (19, 8) and test qPMS7 by varying q from 10 to 19; as shown in Fig. 3(b), RefSelect can effectively accelerate qPMS7 under different q.

C. Assessment of RefSelect on Large Data Sets

All experiments involved in Section IV.B focus on the data sets of small scale, namely the number of input sequences t is small. In recent years, with the rapid development of high-throughput technologies, which allows genome-wide identification of motifs, the data sets such as ChIP-seq [15] contain hundreds or more sequences. Thus, it is necessary to further assess the time performance and validity of RefSelect on large data sets. In the experiments, we set the maximum value of t, k and the sequence length n as 600, 5% $\times$ t and 200, respectively.

Since there is not an exact algorithm that can efficiently deal with large data sets, we assess the validity of RefSelect as follows. Let $N_{original}$ denote the number of candidate motifs generated from the first k sequences in the original input sequences, and $N_{improved}$ denote the number of candidate motifs generated from the k reference sequences selected by RefSelect. Then, we compute $N_{original} / N_{improved}$.

Table 5. Assessment of RefSelect on large data sets

t	k	DNA Sequences		Protein Sequences	
		time	$N_{original} / N_{improved}$	*time*	$N_{original} / N_{improved}$
50	3	0.5s	2.37	0.4s	15.79
100	5	2.0s	2.56	1.8s	15.41
200	10	8.2s	1.85	7.3s	14.20
300	15	18.6s	2.43	16.2s	18.31
400	20	33.5s	2.52	28.7s	18.42
500	25	52.4s	2.78	44.9s	16.64
600	30	75.8s	2.56	66.8s	15.98

s: seconds; *time*: the running time of RefSelect; $N_{original}$ and $N_{improved}$: the number of candidate motifs generated from the first k original input sequences and that for the k reference sequences selected by RefSelect.

On the above basis, we get the running time of RefSelect and $N_{original} / N_{improved}$ on both the DNA and protein data sets, by varying t from 50 to 600. From the results shown in Table 5, we can find that: (1) RefSelect can quickly select reference sequences from large data sets and its running time is independent of the alphabet size; (2) RefSelect can still reduce the generated candidate motifs, especially for the protein data.

V. CONCLUSIONS

We propose a method named RefSelect to select reference sequences for the pattern-driven PMS algorithms, in order to solve the problem that many pattern-driven PMS algorithms present execution time instability. RefSelect requires a small amount of storage space and is capable of selecting reference sequences efficiently and effectively, even for large data sets.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 61173025, 61373044 and 61502366, and the Fundamental Research Funds for the Central Universities under Grant JB150306 and XJS15014.

REFERENCES

[1] P. D'haeseleer, "How does DNA Sequence Motif Discovery Work," Nature Biotechnology, 24(8):959-961, 2006.

[2] N.E. Davey, K.V. Roey, R.J. Weatheritt, et al., "Attributes of Short Linear Motifs," Molecular BioSystems, 8:268-281, 2012.

[3] P. Pevzner, S. Sze, "Combinatorial Approaches to Finding Subtle Signals in DNA Sequences," Proc. The 8th ISMB, AAAI Press, Aug. 2000, pp. 269-278.

[4] F. Zambelli, G. Pesole, G. Pavesi, "Motif Discovery and Transcription Factor Binding Sites before and after the Nextgeneration Sequencing era," Briefings in Bioinformatics, 14(2):225-237, 2013.

[5] J. Davila, S. Balla, S. Rajasekaran, "Space and Time efficient Algorithms for Planted Motif Search," Proc. The 2nd IWBRA, UK, 2006:822-829.

[6] J. Davila, S. Balla, S. Rajasekaran, "Fast and Practical Algorithms for Planted (l, d) Motif Search," IEEE/ACM Transactions on Computational Biology and Bioinformatics, 4(4):544-552, 2007.

[7] Q. Yu, H. Huo, Y. Zhang and H. Guo, "PairMotif: a New Pattern-Driven Algorithm for Planted (l, d) DNA Motif Search," PLoS ONE, 7(10):e48442, 2012.

[8] H. Dinh, S. Rajasekaran and J. Davila, "qPMS7: a Fast Algorithm for Finding (l, d)-Motifs in DNA and Protein Sequences," PLoS ONE, 7(7):e41425, 2012.

[9] S. Tanaka, "Improved Exact Enumerative Algorithms for the Planted (l, d)-Motif Search Problem," IEEE/ACM Transactions on Computational Biology and Bioinformatics, 11(2):361-374, 2014.

[10] E.S. Ho, C.D. Jakubowski, S.I. Gunderson, "iTriplet, a Rule-based Nucleic Acid Sequence Motif Finder," Algorithms for Molecular Biology, 4:14, 2009.

[11] H. Dinh, S. Rajasekaran and V. Kundeti, "PMS5: An Efficient Exact Algorithms for the (l, d)-motif Finding Problem," BMC Bioinformatics, 12:410, 2011.

[12] M. Nicolae and S. Rajasekaran, "Efficient Sequential and Parallel Algorithms for Planted Motif Search," BMC Bioinformatics, 15:34, 2014.

[13] M. Nicolae and S. Rajasekaran, "qPMS9: An Efficient Algorithm for Querum Planted Motif Search," Scientific Reports, 5:7813, 2015

[14] S. van Dongen, "Graph Clustering by Flow Simulation," In PhD thesis Centers for mathematics and computer science (CWI), University of Utrecht, 2000.

[15] E.R. Mardis, "ChIP-seq: Welcome to the New Frontier," Nature Methods, 4:613-614, 2007.