Received July 30, 2019, accepted August 18, 2019, date of publication September 9, 2019, date of current version September 23, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2940115*

# A New Efficient Algorithm for Quorum Planted Motif Search on Large DNA Datasets

**QIANG YU AND XIAO ZHANG**
School of Computer Science and Technology, Xidian University, Xi'an 710071, China
Corresponding author: Qiang Yu (qyu@mail.xidian.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61502366 and Grant 81772011, and in part by the National Key Research and Development Program of China under Grant 2016YFC0102000.

**ABSTRACT** Quorum planted $(l, d)$ motif search (qPMS) is a challenging computational problem in bioinformatics, mainly for the identification of regulatory elements such as transcription factor binding sites in DNA sequences. Large DNA datasets play an important role in identifying high-quality $(l, d)$ motifs, while most existing qPMS algorithms are too time-consuming to complete the calculation of qPMS in a reasonable time. We propose an approximate qPMS algorithm called APMS to deal with large DNA datasets mainly by accelerating neighboring substring search and filtering redundant substrings. Experimental results on them show that APMS can not only identify the implanted $(l, d)$ motifs, but also run orders of magnitude faster than the state-of-the-art qPMS algorithms. The source code of APMS and the python wrapper for the code are freely available at https://github.com/qyu071/apms.

**INDEX TERMS** Quorum planted $(l, d)$ motif search, large DNA datasets, transcription factor binding sites.

## I. INTRODUCTION

Transcription factors bind with specific sites in DNA sequences to initiate gene transcription and to control the transcription efficiency of genes. These sites, typically from 5 to 20 base pairs (bps) in length, are called transcription factor binding sites (TFBSs). Locating TFBSs is of great significance for the research on gene expression regulation [1] and disease-causing variants detection [2].

Quorum planted $(l, d)$ motif search (qPMS) [3], [4] is one of the well-known problem descriptions of locating TFBSs in DNA sequences. For a particular transcription factor (TF), there may be multiple TFBSs in DNA sequences. These TFBSs are usually similar to each other and share the same sequence pattern called a DNA motif. Each of these TFBSs can be regarded as a motif instance, that is, a conservation occurrence of the motif in DNA sequences. Based on the information above, the qPMS problem is defined as follows. Given a set of $tn$-length DNA sequences $D = \{s_1, s_2, \ldots, s_t\}$ and three parameters $l$, $d$ and $q$ satisfying $0 < l < n$, $0 \leq d < l$ and $0 < q \leq 1$, the task is to find a $(l, d)$ motif $m$, an $l$-length string that occurs in at least $qt$ input sequences with up to $d$ mismatches.

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Huei Cheng.

qPMS is a challenging computational problem. First, finding all the $(l, d)$ motifs present in $D$, i.e., solving qPMS exactly, is NP-complete [5]. Second, the input of the current motif discovery is large DNA datasets generated by next-generation sequencing [6], which increases the computational challenge. For example, the ChIP-seq technique [7], a combination of chromatin immunoprecipitation with high-throughput sequencing, allows us to locate TFBSs at genome level, but it generates thousands of or even more sequences containing the binding sites of a certain ChIP-ed motif.

Over the past 10+ years, many qPMS algorithms have been proposed [8]–[10]. Exact qPMS algorithms are always capable of finding the optimum $(l, d)$ motif through brute-force search, but they are time-consuming especially when dealing with large datasets. Sample-pattern-driven exact algorithms, such as PMSprune [4], StemFinder [11], qPMS7 [12], TravStrR [13], PMS8 [14] and qPMS9 [15], usually contain sample-driven phase and pattern-driven phase. In the sample-driven phase, these algorithms adopt some selected reference sequences as constraints to generate as few candidate motifs as possible. In the pattern-driven phase, they verify whether each candidate motif is a $(l, d)$ motif. Such exact algorithms outperform other exact algorithms when processing small datasets, but they cannot process large datasets due to the generation of too many candidate motifs. The suffix tree-based exact algorithms, such as Weeder [16], RISOTTO [17]

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

and FMotif [18], search all candidate motifs on a pattern tree and speed up the verification of candidate motifs by using the suffix tree indexes of the input sequences. Algorithms of this kind can efficiently solve the problem instances of small $l$ and $d$ on large datasets, but they fail to efficiently solve the problem instances of large $l$ and $d$, even for small datasets.

Approximate qPMS algorithms aim to identify the optimum or near optimum motif. They usually adopt an optimization method, such as expectation maximization [19], Gibbs sampling [20] and genetic algorithm [21], [22], to refine a group of initial motifs. In these algorithms, MEME-ChIP [19], which is based on expectation maximization, emerges as one of the most famous motif discovery algorithms. Such algorithms have good time performance in processing small datasets. However, as both the number of initial motifs and the computational complexity of refining one initial motif increase with the dataset size, the total running time of such algorithms grows rapidly with the increment.

The recent years have witnessed the proposal of some motif discovery algorithms based on new strategies aimed at efficiently processing large datasets. PairMotifChIP [23] discovers motifs by mining and merging pairs of similar substrings in the input sequences. It spends a large portion of running time on the former operation, which shows quadratic growth as the dataset size increases. Some algorithms, such as the web version of MEME-ChIP [19] and MICSA [24], randomly select a portion of the entire dataset (e.g., 600 input sequences) to discover motifs, which may result in the loss of infrequent motifs. qPMS10 [25] and SamSelect [26] are specialized algorithms for selecting sample sequences. They select some sample sequences and then perform motif discovery on the selected sample sequences by running the existing qPMS algorithms. However, the running time required for processing challenging problem instances is still relatively long.

In summary, there is still significant room for improvement in the time performance of solving qPMS on large DNA datasets. For the traditional pattern-driven qPMS algorithms, they utilize a portion of sequences (i.e., reference sequences) to generate candidate motifs that cover the $(l, d)$ motifs, but the inefficient utilization of the entire dataset results in too many redundant candidate motifs, leading to a considerable amount of unnecessary candidate motif verifications. In this paper, we employ the entire dataset to generate some high-quality initial motifs (hereinafter referred to as seeds) and design new pattern-driven methods for efficient refinement of the seeds. In addition, considering that multiple seeds may correspond to the same motif, we design a method to filter redundant seeds with an aim to reduce unnecessary calculations. Based on these preconditions, we propose an approximate qPMS algorithm called AMPS. APMS can not only identify the implanted $(l, d)$ motifs in large DNA datasets, but also run orders of magnitude faster than the compared algorithms.

## II. METHODS

### A. ALGORITHM OVERVIEW

Notations used in this paper are shown below. $D = \{s_1, s_2, \ldots, s_t\}$ denotes a set of $t$ input DNA sequences; each input sequence $s_i$ is an $n$-length string over the DNA alphabet $\Sigma = \{\mathrm{A, C, G, T}\}$. $s[j]$ denotes the $j$th character in a string $s$. $s[j..j']$ denotes a substring of a string $s$ from the $j$th position to the $j'$th position. An $l$-mer denotes an $l$-length string. $x \in_l y$ denotes that a string $x$ is an $l$-length substring of a string $y$. In other words, $x$ is an $l$-mer in a string $y$. $d_H(x, y)$ denotes the Hamming distance between two strings $x$ and $y$ of equal length. $|x|$ denotes the length of a string $x$, the size of a set $x$ or the number of columns in an alignment $x$ of a set of strings. For two strings $y$ and $s$ satisfying $|y| < |s|$, $dis(y, s)$ denotes the distance between $y$ and $s$, which is the minimum Hamming distance between the strings $y$ and $z$ for $z \in_{|y|} s$. The $d$-neighbors of an $l$-mer $x$ is the $l$-mers with Hamming distance less than or equal to $d$ from $x$, represented as $\{y : y \in \Sigma^{|x|}, d_H(x, y) \leq d\}$.

The process and basic ideas of APMS are described below:

a) Extract high-frequency substrings of some length $k$ from the dataset and store them in a set $A$. This step is the preparation for the generation of seeds. For a large DNA dataset $D$, there may be a common $k$-length substring $x$ of some instances of a specific motif, so that the frequency of $x$ in $D$ is usually higher than that of a random $k$-mer in $D$. Thus, a set of high-frequency $k$-mers may contain some substrings of motif instances with high probability. We need to elaborately determine the value of $k$ and the high-frequency threshold.

b) Generate a seed $m'$ by the $k$-mer $x$ with the highest frequency in $A$. We define the instances of $x$ as the substrings of length $2l - k$ in $D$ derived by extending the occurrences of $x$ in $D$ to both the left and to the right for $l - k$ characters separately. Suppose that $x$ is a common substring of some instances of a motif $m$. Then some instances of $x$ will cover the motif instances of $m$. Therefore, the basic idea for seed generation is to select the $l$-length fragment with the largest information content from the alignment of the instances of $x$, and take the consensus sequence of this fragment as the seed $m'$ generated by $x$.

c) Refine the seed $m'$ generated by $x$. $m'$ is expected to be an instance of some $(l, d)$ motif $m$. The refinement of $m'$ is to find $m$ by searching the $d$-neighbors of $m'$. Since the number of $d$-neighbors of $m'$ grows dramatically with the increase of $l$ and $d$, the key to refinement is to design efficient methods for searching $d$-neighbors and verifying whether each $d$-neighbor is a $(l, d)$ motif.

d) If a $(l, d)$ motif $m$ is obtained through the refinement of $m'$, then filter out a portion of redundant $k$-mers in $A$ using $m$. For a large DNA dataset $D$, the instances of $m$ in $D$ can render multiple high-frequency $k$-mers. Since $m$ can be obtained by refining the seed generated by each of these $k$-mers, most of the $k$-mers are redundant. Filtering out redundant $k$-mers helps to reduce the seeds for refinement, thereby effectively improving the time performance of the entire algorithm.

e) If A is not empty, return to step b), otherwise output the obtained $(l, d)$ motifs in a descending order according to their consensus sequence score.

Steps of extracting high-frequency substrings, generating seeds, refining seeds and filtering redundant substrings are described in detail below.

### B. EXTRACTING HIGH-FREQUENCY SUBSTRINGS

First, probability analysis is employed to determine a suitable value of $k$, so that we can better distinguish the $k$-mers in background sequences from that in motif instances. Let $f_r(k)$ denote the expectation of the frequency of an arbitrary background $k$-mer occurring in $D$. Let $f_m(k)$ denote the expectation of the frequency of an arbitrary $k$-mer in an arbitrary motif instance occurring in $D$. The greater the ratio of $f_m(k)$ to $f_r(k)$, the more distinguishable the $k$-mers in background sequences from that in motif instances. Therefore, we use (1) to determine the value of $k$, where $k_{\min}$ represents the minimum value of $k$ and $\varepsilon$ is a factor to deal with the situation where $f_r(k)$ is less than 1. According to empirical studies, $k_{\min}$ and $\varepsilon$ are set to 5 and 1, respectively.

$$k = \arg\max_{k_{\min} \le i \le l} \frac{f_m(i)}{f_r(i) + \varepsilon} \tag{1}$$

The way of how to calculate $f_r(k)$ and $f_m(k)$ is introduced. $f_r(k)$ can be calculated by (2). For a $k$-mer $x_1$ at an arbitrary initial position in an arbitrary motif instance $m_1$ and a $k$-mer $x_2$ in an arbitrary motif instance $m_2$ at the same initial position as $x_1$, let $p_k$ denotes the probability that $x_1$ is equal to $x_2$. With $p_k$, $f_m(k)$ can be calculated by (3), which indicates the total frequency of an arbitrary $k$-mer in an arbitrary motif instance occurring in the background sequences and the motif instances.

$$f_r(k) = t \times (n - k + 1) \times \frac{1}{4^k} \tag{2}$$

$$f_m(k) = t \times (n - k + 1) \times \frac{1}{4^k} + t \times q \times p_k - t \times q \times \frac{1}{4^k} \tag{3}$$

The way of how to calculate $p_k$ is further elaborated. According to the Theorem of Total Probability, $p_k$ can be derived from (4). Suppose that the motif is $m$. $\Pr_i$ and $\Pr_j$ represent the probability of $d_H(m, m_1) = i (0 \le i \le d)$ and $d_H(m, m_2) = j (0 \le j \le d)$, respectively. $\Pr_i$ and $\Pr_j$ can be calculated by (5) [23], where $g (0 \le g \le 1)$ denotes the conservation parameter. $p_{ij}$ represents the probability that $x_1$ is equal to $x_2$ under the condition that $d_H(m, m_1) = i$ and $d_H(m, m_2) = j$. As shown in (6), $p_{ij}$ accumulates the product multiplied by three factors when $a$ takes value ranging from 0 to $\min\{i, j\}$. The first factor represents the probability that there are $a$ mutations in an arbitrary $k$-mer $x_1$ in $m_1$. Let $x_2$ denote the $k$-mer in $m_2$ at the same initial position as $x_1$. The second factor represents the probability that the mutated positions in $x_2$ are the same with that in $x_1$. The third factor represents the probability that the base at each mutated position is identical for $x_1$ and $x_2$ under the condition that the mutated positions in $x_2$ are the same with that in $x_1$.

$$p_k = \sum_{0 \le i, j \le d} \Pr_i \times \Pr_j \times p_{ij} \tag{4}$$

$$\Pr_i = \binom{d}{i} \times g^i \times (1 - g)^{d-i} \tag{5}$$

$$p_{ij} = \sum_{0 \le a \le \min\{i,j\}} \frac{\binom{k}{a} \times \binom{l-k}{i-a}}{\binom{l}{i}} \times \frac{\binom{l-k}{j-a}}{\binom{l}{j}} \times \frac{1}{3^a} \tag{6}$$

When the value of $k$ is determined, the following method is adopted to store the occurrence frequency of all $k$-mers in $D$ in an array $F$ of size $4^k$. First, we initialize each element in $F$ to 0. Then, we traverse all the $k$-mers in $D$. For each $k$-mer $x$ in $D$, we increase $F[stn(x)]$ by 1, where $stn(x)$ denotes an integer ranging from 0 to $4^k - 1$ converted from a $k$-mer $x$ by encoding the characters A, C, G, and T in $x$ as the binary numbers 00, 01, 10, and 11, respectively.

Finally, we take the $k$-mers with frequency greater than or equal to a threshold $\varphi$ as high-frequency $k$-mers and store them in the set $A$. With the following aspects taken into consideration, we employ (7) to derive the value of $\varphi$. As mentioned above, $f_m(k)$ represents the expectation of the frequency of an arbitrary $k$-mer in an arbitrary motif instance occurring in $D$. If $\varphi$ is directly set to $f_m(k)$, then we will obtain multiple high-frequency $k$-mers corresponding to a specific motif. In fact, only one high-frequency $k$-mer corresponding to a specific motif is required for seed generation. Therefore, we introduce a variable proportional to $t$ to avoid obtaining too many redundant high-frequency $k$-mers.

$$\varphi = f_m(k) + t \times 0.1\% \tag{7}$$

We describe the overall process of extracting high-frequency substrings in Algorithm 1.

**Algorithm 1** ExtractHighFrequencySubstring($D, l, d, q$)

```
Input: the input dataset $D$ and the parameters $l$, $d$ and $q$
Output: a set $A$ of high-frequency substrings
1: determine the substring length $k$ by (1)
2: determine the high-frequency threshold $\varphi$ by (7)
3: $F \leftarrow$ an array of $4^k$ zero
4: $A \leftarrow$ empty
5: for each $k$-mer $x$ in $D$ do
6:    $F[stn(x)] \leftarrow F[stn(x)] + 1$
7:    if $F[stn(x)] = \varphi$ then
8:       $A \leftarrow A + x$
9: return $A$
```

### C. GENERATING SEEDS

Given a $k$-mer $x$ in the set $A$, Algorithm 2 describes the process of generating seeds, including the following three steps.

The first step (lines 2 to 4) is to obtain the instances of $x$ in $D$ and store them in a set $I(x)$. As we do not know the initial

**Algorithm 2** GenerateSeed($x$, $D$)

```
Input: a $k$-mer $x$, the input dataset $D$
Output: a seed
1: $I(x) \leftarrow$ empty, $MinPQ \leftarrow$ empty
2: for each $k$-mer $s_i[j..j+k-1]$ in $D$ do
3:   if $x = s_i[j..j+k-1]$ and $j-l+k \geq 1$ and $j+l-1$
       $\leq |s_i|$ then
4:     $I(x) \leftarrow I(x) + s_i[j-l+k..j+l-1]$
5: for each instance $y$ in $I(x)$ do
6:   $MinPQ \leftarrow MinPQ + y$
7:   if $|MinPQ| > |I(x)| - f_r(k)$ then
8:     dequeue the instance with the minimum score from
       $MinPQ$
9:   $align \leftarrow$ the alignment of the instances in $MinPQ$
10: while $|align| > l$ do
11:   if $r(align[1]) < r(align[|align|])$ then
12:     update $align$ by deleting the first column
13:   else
14:     update $align$ by deleting the last column
15: return the consensus obtained from $align$
```

position of $x$ in the motif, we extend each occurrence of $x$ in $D$ to both the left and right for $l-k$ characters separately and refer the obtained substring of length $2l-k$ as an instance of $x$ in $D$. For example, suppose that $s_i[j..j+k-1]$ is an occurrence of $x$ in $D$, and then the associated instance of $x$ in $D$ is $s_i[j-l+k..j+l-1]$. By doing so, we expect that an instance of $x$ in $D$ can cover a motif instance. For the convenience of the alignment operation in the third step, we ignore such occurrences of $x$ in $D$ that cannot be extended to the instances of length $2l-k$ because the number of characters on the left or right is less than $l$–$k$.

The second step (lines 5 to 8) is to filter out randomly overexpressed instances in $I(x)$. If an instance $y$ of $x$ in $D$ does not contain a motif instance, that is, $y$ is composed entirely of background bases, we call $y$ a randomly overexpressed instance. The $k$-mers in a randomly overexpressed instance except for $x$ normally do not have a relatively high frequency. The randomly overexpressed instances affect the quality of seeds by reducing the information content of the columns in the seeds but not in $x$. We employ (8) to evaluate the score $score_{\mathrm{i}}(y)$ of an instance $y$, representing the maximum frequency of the $k$-mer in $y$ with up to $k-3$ overlapped bases with $x$. We do not consider the $k$-mers adjacent to $x$ in $y$ because their frequency may be affected by $x$. The smaller the score of an instance $y$, the more likely $y$ is to be randomly overexpressed. As $f_r(k)$ indicates the expectation of the frequency of an arbitrary background $k$-mer occurring in $D$, we filter out $f_r(k)$ instances of the lowest score from $I(x)$.

$$score_{\mathrm{i}}(y) = \max_{z \in_k y[1..l-3] \text{ or } z \in_k y[l-k+4..2l-k]} f(z) \tag{8}$$

The third step (lines 9 to 15) is to generate a seed using the remaining instances in $I(x)$. These instances can form an alignment of length $2l-k$. In this alignment, the part corresponding to the motif is $l$ continuous columns and the information content of these columns is generally higher than that of other columns. Let $r(align[i])$ represent the information content of the $i$th column in an alignment $align$. The method for computing $r(align[i])$ can be found in literature [27]. We repeatedly remove columns in the alignment with small information content on both sides until we obtain an $l$-length alignment, and then take the consensus sequence of this $l$-length alignment as the generated seed.

## D. REFINING SEEDS

Given a seed $m$', it is too time-consuming for the refinement of $m$' by exhaustively searching the $d$-neighbors of $m$'. In order to deal with this problem, we design a heuristic search method.

As shown in Fig. 1, the $d$-neighbors of a seed $m$' can be represented as a binomial tree of depth $d$. The root node of the tree represents $m$'. Any internal node or leaf node $v$ corresponds to a group of $d$-neighbors of $m$' that differ from $m$' in the marked positions. Thus, the nodes of depth $h$ $(0 \leq h \leq d)$ indicate all the $d$-neighbors of $m$' with Hamming distance $h$ from $m$'. The binomial tree in Fig.1 is a simplified version, which shows the positions differing from $m$' without any indication of the expansion of the altered bases. The tree mentioned later in this paper is fully expanded, that is, each node corresponds to one $d$-neighbor.

Suppose that the Hamming distance between the target $(l, d)$ motif $m$ and $m$' is $d$, then $m$ is a node of depth $d$. For each layer $h(0 \leq h < d)$ of the tree, we define the intermediate motifs as such $d$-neighbors $m$'' under the condition that for any position $j(1 \leq j \leq l)$, if $m''[j] \neq m'[j]$, then $m''[j] = m[j]$.Therefore, the ancestor node of $m$ in the $h$th $(0 \leq h < d)$ layer must be an intermediate motif in that layer. Furthermore, the child nodes of the intermediate motifs in the $h$th layer must contain the intermediate motifs of the $h+1$th layer. We adopt (9), the score of consensus sequence under the qPMS model, to evaluate the score of each node $y$ in the tree. As described in (10), $D'(y)$ is a set of $qt$ sequences selected from $D$ to calculate the score of $y$. Generally, the score of a node corresponding to an intermediate motif is higher than that of other nodes; moreover, the closer it is to $m$, the higher the score of the intermediate motif.

$$score_{\mathrm{n}}(y) = \sum_{s \in D'(y)} l - dis(y, s) \tag{9}$$

$$\begin{cases} D'(y) \subset D, \\ |D'(y)| = qt, \\ \min_{s \in D'(y)} l - dis(y, s) \geq \max_{s' \in D - D'(y)} l - dis(y, s'). \end{cases} \tag{10}$$

Based on these considerations, we search the tree layer by layer from the root node. First, we determine whether the root node is a $(l, d)$ motif and take the child nodes of the root node as the extended nodes in the first layer. Then,

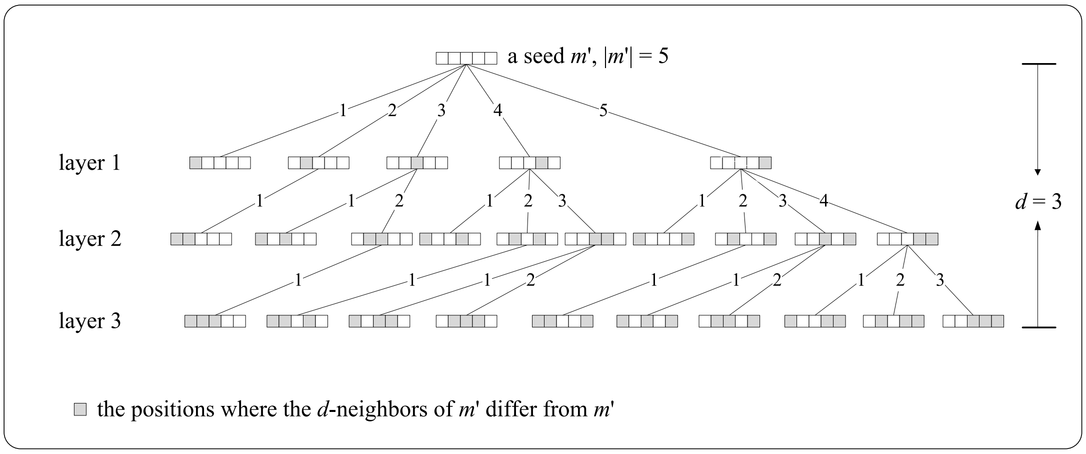


**FIGURE 1.** Binomial tree representation of the $d$-neighbors of a seed.

for the $i$th layer ($0 < i < d$), we determine whether each extended node in this layer is a ($l$, $d$) motif, select some extended nodes with high score in this layer as potential intermediate motifs, and take the child nodes of these selected nodes as the extended nodes in the $i + 1$th layer. Let $N_{mm}(i)$ denote the number of selected nodes in the $i$th ($0 < i < d$) layer. To avoid losing intermediate motifs, we utilize (11) to calculate $N_{mm}(i)$, indicating the number of intermediate motifs in the $i$th layer multiplied by a security factor $\alpha$ ($\alpha \geq 1$). In the implementation of APMS, we set $\alpha$ to 2 empirically. Finally, for the $d$th layer, we determine whether each extended node in this layer is a ($l$, $d$) motif. In this searching process, if more than one ($l$, $d$) motif is obtained, we output the one with the highest score and give its $P$-value [28], [29].

$$N_{mm}(i) = \binom{d}{i} \times \alpha \tag{11}$$

Algorithm 3 describes the process of refining a seed. We employ a minimum priority queue $MinPQ$ to store $N_{mm}(i)$ selected nodes in the $i$th layer. It is important to note that, the calculation of the score of a node (i.e., a $d$-neighbor) by (9) can be simultaneously used to determine whether this node is a ($l$, $d$) motif.

In addition, in the implementation of Algorithm 3, we speed up the calculation of the score of a $d$-neighbor $y$ of the seed $m$' (i.e., the calculation of verifying whether $y$ is a ($l$, $d$) motif). The key is how to quickly calculate $dis(y,s_i)$ for each sequence $s_i (1 \leq i \leq t)$ in $D$. We sort $l$-mers $z$ in $s_i$ in an ascending order according to $d_H(m', z)$ and store these $l$-mers in an array $C_i$. Given a position $j$ ($1 < j \leq |C_i|+ 1$), let $dis(y, C_i, j)$ denote the smallest $d_H(y, C_i[j'])$ for $1 \leq j' < j$. Let $dis(y, C_i) = dis(y, C_i, |C_i|+ 1)$, and then $dis(y, s_i) = dis(y, C_i)$. We have the following three observations.

**Observation 1.** Given a seed $m$' and $C_i$, for any position $j$ and $j$' satisfying $1 \leq j \leq j' \leq |C_i|$, $d_H(C_i[j'], m') \geq d_H(C_i[j], m')$.

**Observation 2.** Given a seed $m$', a $d$-neighbor $y$ of $m$', $C_i$ and a positon $j$ ($1 \leq j \leq |C_i|$), if $d_H(C_i[j], m') - d_H(y, m') \geq 0$, then $d_H(C_i[j], m') - d_H(y, m')$ is the minimum value of $d_H(y,C_i[j])$.

**Observation 3.** Given a seed $m$', a $d$-neighbor $y$ of $m$', $C_i$ and a positon $j$ ($1 < j \leq |C_i|$), if $d_H(C_i[j], m') - d_H(y, m') \geq dis(y, C_i, j)$, then for any $j \leq j' \leq |C_i|$, $d_H(C_i[j'], m') - d_H(y, m') \geq dis(y, C_i, j)$, and thus $dis(y, C_i) = dis(y, C_i, j)$.

Based on these observations, given a seed $m$', a $d$-neighbor $y$ of $m$' and a sequence $s_i$ in $D$, Algorithm 4 describes how $dis(y, s_i)$ is efficiently calculated.

**Algorithm 3** RefineSeed($m$', $D$)

```
Input: a seed $m'$, the input dataset $D$
Output: a $(l, d)$ motif $m$
1: $MinPQ \leftarrow$ empty, $score \leftarrow 0$, $m \leftarrow$ empty string
2: if $m'$ is a $(l, d)$ motif and $score_n(m') > score$ then
3:   $m \leftarrow m'$, $score \leftarrow score_n(m')$
4: $MinPQ \leftarrow MinPQ + m'$
5: for $i \leftarrow 0$ to $d - 1$ do
6:   $MinPQ' \leftarrow$ empty
7:   while $|MinPQ| \neq 0$ do
8:     dequeue a $node$ from $MinPQ$
9:     for each child node $node'$ of $node$ do
10:      if $node'$ is a $(l, d)$ motif and
         $score_n(node') > score$ then
11:        $m \leftarrow node'$, $score \leftarrow score_n(node')$
12:      $MinPQ' \leftarrow MinPQ' + node'$
13:      if $|MinPQ'| > N_{mm}(i+1)$ then
14:        dequeue a node from $MinPQ'$
15:  $MinPQ \leftarrow MinPQ'$
16: return $m$
```

## E. FILTERING REDUNDANT SUBSTRINGS

For each obtained motif $m$, we employ $m$ to determine whether each $k$-mer $x$ in $A$ is a redundant $k$-mer. A redundant $k$-mer $x$ is a common substring of some instances of $m$ with the same initial position or there is a $k$'-length

**Algorithm 4** ComputeMinDis($m$', $y$, $s_i$)

```
Input: a seed $m$', a neighbor $y$ of $m$' and a sequence $s_i$ in $D$
Output: $dis(y, s_i)$
1: $minDis \leftarrow l$
2: for $j \leftarrow 1$ to $|C_i|$ do //$C_i$ is precomputed according to $m$' and $s_i$
3:   if $d_H(C_i[j], m') - d_H(y, m') \geq minDis$ then
4:     return $minDis$
5:   else
6:     if $d_H(y, C_i[j]) < minDis$ then
7:       $minDis \leftarrow d_H(y, C_i[j])$
8: return $minDis$
```

**Algorithm 5** FilterRedundancy($m$, $A$)

```
Input: a motif $m$, a set $A$ of high-frequency $k$-mers
Output: a set $A$ of high-frequency $k$-mers after filtering
1: for each$k$-mer $x$ in $A$ do
2:   if there exists $z \in_k m$ such that $d_H(z, x) \leq e(k)$ then
3:     $A \leftarrow A - m'$
4:   else
5:     for $k' \leftarrow k_{\min}$ to $k - 1$ do
6:       if $d_H(pf(m, k'), sf(x, k')) \leq e(k')$ or $d_H(sf(m, k'), pf(x, k')) \leq e(k')$ then
7:         $A \leftarrow A - m'$
8:         break
9: return $A$
```

($k_{\min} \leq k' < k$) overlap between $x$ and some instances of $m$. $k_{\min}$ is set to 5 according to Section II.B.

Here is the formalized definition of redundant $k$-mers. Let $e(k)$ represent the expectation of the Hamming distance between a $k$-mer $x_1$ at an arbitrary initial position in an arbitrary motif instance and a $k$-mer $x_2$ in the motif at the same initial position as $x_1$. Let $pf(x, k')$ and $sf(x, k')$ represent the $k'$-length prefix and $k'$-length suffix of a string $x$, respectively. For a given motif $m$, a $k$-mer $x$ in $A$ that meets the following requirements is regarded as a redundant $k$-mer: there is a $k$-mer $z$ in $m$ such that $d_H(z, x) \leq e(k)$, or there is a value of $k'$ within the range $k_{\min} \leq k' < k$ such that $d_H(pf(m, k'), sf(x, k')) \leq e(k')$ or $d_H(sf(m, k'), pf(x, k')) \leq e(k')$.

$e(k)$ is calculated by (12). First, $e(l)$ can be calculated based on the Theorem of Total Probability. Second, for any mismatch between a motif instance and the associated motif, suppose that this mismatch occurs randomly at one of the $l$ positions, and then $e(k)$ is equal to $e(l)$ multiplied by $k/l$.

$$e(k) = \left(\sum_{i=0}^{d} \Pr_i \times i\right) \times \frac{k}{l} \tag{12}$$

Algorithm 5 describes the process of filtering out redundant $k$-mers in $A$ for an obtained motif $m$.

### F. OVERALL ALGORITHM AND ANALYSIS

Algorithm 6 describes the overall process of APMS.

**Algorithm 6** APMS($D$,$l$, $d$, $q$)

```
Input: the input dataset $D$ and the parameters $l$, $d$ and $q$
Output: a group of $(l, d)$ motifs
1: $M \leftarrow$ empty
2: $A \leftarrow$ ExtractHighFrequencySubstring($D$,$l$, $d$, $q$) // Algorithm 1
3: while$|A| \neq 0$ do
4:   $x \leftarrow$ the $k$-mer with the maximum frequency in $A$
5:   $A \leftarrow A - x$
6:   $m' \leftarrow$ GenerateSeed($x$, $D$) // Algorithm 2
7:   $m \leftarrow$ RefineSeed($m'$, $D$) // Algorithm 3
8:   if $m$ is not an empty string then
9:     $M \leftarrow M + m$
10:    $A \leftarrow$ FilterRedundancy($m$, $A$) // Algorithm 5
11: return $(l, d)$ motifs in $M$ in descending order according to their consensus score
```

The space overhead of APMS mainly lies in storing the array $F$ of size $4^k$ and all $l$-mers in $D$ ($t$ $n$-length sequences). Each element in $F$ is a 4-byte integer and each $l$-mer in $D$ takes up $l$ bytes. Therefore, the space complexity of APMS is $O(4^{k+1} + tnl)$.

The time complexity $T_{\text{APMS}}$ of APMS mainly depends on seed refinement. Let $\beta$ denote the number of seeds that need to be refined. Considering the filtration of redundant substrings, $\beta$ is taken as $O(n)$. The overhead for refining a seed is equal to the overhead $O(tnl)$ for verifying each $d$-neighbor multiplied by the number of verified $d$-neighbors of $m$'. According to Algorithm 3, the root node of the tree, $3l$ $d$-neighbors in the first layer and up to $N_{mm}(i-1) \times 3l$ $d$-neighbors in the $i$th ($1 < i \leq d$) layer of the tree need to be verified. Therefore, we derive the following $T_{\text{APMS}}$, from which a conclusion can be drawn that the time complexity of APMS is linearly proportional to the number of input sequences.

$$\begin{aligned} T_{\text{APMS}} &= O\left(\beta \times \left(1 + 3l + \sum_{i=1}^{d-1} N_{mm}(i) \times 3l\right) \times tnl\right) \\ &= O\left(\sum_{i=1}^{d-1} \binom{d}{i} \times 3tn^2 l^2\right) \end{aligned} \tag{13}$$

## III. RESULTS AND DISCUSSION

### A. RESULTS ON SIMULATED DATA

The experiments on simulated data are mainly to test the efficiency of APMS by comparing the running time of APMS with that of existing algorithms, and to validate whether APMS can identify the implanted motif. We use $mPC$, the performance coefficient at motif level, to measure the similarity between the predicted $(l, d)$ motif $m_{\text{p}}$ and the implanted $(l, d)$ motif $m_{\text{k}}$, where $len_{\text{overlap}}(m_{\text{p}}, m_{\text{k}})$ represents the length of the overlapped part of $m_{\text{p}}$ and $m_{\text{k}}$.

$$mPC = \frac{len_{\text{overlap}}(m_{\text{p}}, m_{\text{k}})}{2l - len_{\text{overlap}}(m_{\text{p}}, m_{\text{k}})} \tag{14}$$

**TABLE 1. Results on the first group of simulated datasets.**

| *(l, d)* | APMS | | FMotif | | PairMotifChIP | | GADEM | | MEME-ChIP | |
|---|---|---|---|---|---|---|---|---|---|---|
| | time | *mPC* | time | *mPC* | time | *mPC* | time | *mPC* | time | *mPC* |
| (9, 2) | 2.4 s | 1.0 | 1.7 m | 1.0 | 7.3 m | 1.0 | 1.3 h | 1.0 | 3.9 h | 1.0 |
| (11, 3) | 4.0 s | 1.0 | 18.7 m | 1.0 | 7.3 m | 1.0 | 1.4 h | 1.0 | 3.9 h | 1.0 |
| (13, 4) | 5.2 s | 1.0 | 2.6 h | 1.0 | 7.6 m | 1.0 | 1.4 h | 1.0 | 3.8 h | 1.0 |
| (15, 5) | 8.5 s | 1.0 | 26.4 h | 1.0 | 7.6 m | 1.0 | 1.4 h | 1.0 | 3.7 h | 1.0 |
| (17, 6) | 17.5 s | 1.0 | N | N | 7.5 m | 1.0 | 1.3 h | 1.0 | 3.6 h | 1.0 |
| (19, 7) | 18.9 s | 1.0 | N | N | 7.7 m | 1.0 | 1.3 h | 1.0 | 3.6 h | 1.0 |
| (21, 8) | 28.9 s | 1.0 | N | N | 7.7 m | 1.0 | 1.3 h | 1.0 | 3.5 h | 1.0 |

s: seconds; m: minutes; h: hours; N: no result because the running time exceeds 48 hours.

**TABLE 2. Results on the second group of simulated datasets.**

| *q* | *g* | APMS | | FMotif | | PairMotifChIP | | GADEM | | MEME-ChIP | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | time | *mPC* | time | *mPC* | time | *mPC* | time | *mPC* | time | *mPC* |
| 0.2 | 0.2 | 13.0 s | 1.0 | N | N | 5.8 m | 1.0 | 1.5 h | 1.0 | 3.7 h | 1.0 |
| | 0.5 | 10.2 s | 1.0 | N | N | 5.7 m | 1.0 | 1.5 h | 1.0 | 3.7 h | 1.0 |
| | 0.8 | 9.0 s | 1.0 | N | N | 5.7 m | 1.0 | 1.4 h | 1.0 | 3.7 h | 1.0 |
| 0.5 | 0.2 | 10.7 s | 1.0 | 28.9 h | 1.0 | 6.1 m | 1.0 | 1.3 h | 1.0 | 3.7 h | 1.0 |
| | 0.5 | 9.0 s | 1.0 | 26.4 h | 1.0 | 5.9 m | 1.0 | 1.3 h | 1.0 | 4.6 h | 1.0 |
| | 0.8 | 8.9 s | 1.0 | 28.4 h | 1.0 | 5.8 m | 1.0 | 1.4 h | 1.0 | 3.7 h | 1.0 |
| 0.8 | 0.2 | 10.6 s | 1.0 | 19.3 h | 1.0 | 6.4 m | 1.0 | 1.3 h | 1.0 | 3.7 h | 1.0 |
| | 0.5 | 8.4 s | 1.0 | 19.0 h | 1.0 | 6.0 m | 1.0 | 1.4 h | 1.0 | 3.7 h | 1.0 |
| | 0.8 | 9.1 s | 1.0 | 19.1 h | 1.0 | 5.7 m | 1.0 | 1.4 h | 1.0 | 3.7 h | 1.0 |

s: seconds; m: minutes; h: hours; N: no result because the running time exceeds 48 hours.

We generate the simulated data as follows [3], [4]: randomly generate an $l$-length motif $m$ and a dataset $D$ of $t$ $n$-length DNA sequences; then, randomly select $qt$ sequences from $D$; for each selected sequence $s$, implant a random instance $m$' of $m$ at a random position $j$ ($1 \leq j < |s| - l + 1$) of $s$. We generate a random instance of $m$ as follows [23]: randomly select $d$ positions from $m$, and then, for each selected position $j$, vary $m[j]$ to a different character with probability $g$.

In order to carry out a comprehensive testing, three groups of simulated datasets are generated by controlling values of $t$, $n$, $l$, $d$, $q$ and $g$. The first group of simulated datasets corresponds to the data with different $(l, d)$ motifs obtained by fixing $t = 3000$, $n = 200$, $q = 0.5$ and $g = 0.5$ and varying $(l, d)$ from (9, 2) to (21, 8). The second group of simulated datasets corresponds to the data with different motif signal strength obtained by fixing $t = 3000$, $n = 200$ and $(l, d) =$ (15,5) and taking $q$ / $g$ as 0.2, 0.5 and 0.8. The third group of simulated datasets corresponds to the data at different scales obtained by fixing $n = 200$, $(l, d) = (15, 5)$, $q = 0.5$ and $g = 0.5$ and varying $t$ from 3000 to 30000.

The compared algorithms include FMotif [18], PairMotifChIP [23], GADEM [22] and MEME-ChIP [19]. FMotif is the most efficient exact qPMS algorithm for processing large DNA datasets; PairMotifChIP is an approximate qPMS algorithm proposed recently with an ability to handle large DNA datasets; GADEM is designed for motif discovery on large DNA datasets by combining a genetic algorithm and an EM algorithm; MEME-ChIP is one of the most famous motif discovery algorithms. All these algorithms are implemented in C/C++. We execute them on an environment with a single 2.7GHz CPU and a 12GB memory. For each setting of $t$, $n$, $(l, d)$, $q$ and $g$, we randomly generate three datasets and take the average of the results on the three datasets as the reported result.

We show the results on the first, second and third groups of simulated datasets in Tables 1, 2 and 3, respectively. As FMotif is limited to perform a maximum number of 3000 sequences, it is absent from the comparison on the third group of datasets. Since the value of the reported *mPC* is 1.0 on each dataset, all these algorithms are able to identify the implanted motifs exactly. The main reason for this phenomenon is that the large datasets contain quite sufficient motif information, even when the signal strength of the motif is relatively weak ($q$ is small while $g$ is large) as shown in Table 2. However, some compared algorithms (FMotif and MEME-ChIP) fail to make predictions on some datasets because their running time exceeds 48 hours. Therefore, the running time is regarded as a critical factor in evaluating algorithms for motif discovery on large DNA datasets.

We compare the running time of these algorithms as follows. On the whole, APMS can complete the calculation in the shortest time on all these datasets, and it is orders of magnitude faster than the compared algorithms. It just takes two minutes even when processing the largest dataset (30000 sequences). In processing 3000 sequences, APMS is 15 times faster or even more than PairMotifChIP, and is much faster than FMotif, GADEM and MEME-ChIP. More importantly, in processing 6000 or more sequences, the advantage of APMS increases with the increase of the number of sequences. As shown in Fig. 2, the running time of APMS shows linear growth, which is consistent with the time complexity of APMS, while the running time of PairMotifChIP and MEME-ChIP shows approximately quadratic growth as the number of sequences increases. Although the running

**TABLE 3. Results on the third group of simulated datasets.**

| *t* | APMS | | PairMotifChIP | | GADEM | | MEME-ChIP | |
|---|---|---|---|---|---|---|---|---|
| | time | *mPC* | time | *mPC* | time | *mPC* | time | *mPC* |
| 6000 | 19.4 s | 1.0 | 30.2 m | 1.0 | 2.8 h | 1.0 | 17.3 h | 1.0 |
| 9000 | 28.3 s | 1.0 | 1.1 h | 1.0 | 3.5 h | 1.0 | 37.0 h | 1.0 |
| 12000 | 42.4 s | 1.0 | 1.9 h | 1.0 | 5.8 h | 1.0 | N | N |
| 15000 | 53.1 s | 1.0 | 3.1 h | 1.0 | 7.0 h | 1.0 | N | N |
| 18000 | 1.1 m | 1.0 | 4.4 h | 1.0 | 7.2 h | 1.0 | N | N |
| 21000 | 1.4 m | 1.0 | 6.2 h | 1.0 | 9.8 h | 1.0 | N | N |
| 24000 | 1.6 m | 1.0 | 8.0 h | 1.0 | 10.0 h | 1.0 | N | N |
| 27000 | 1.7 m | 1.0 | 10.1 h | 1.0 | 12.8 h | 1.0 | N | N |
| 30000 | 2.0 m | 1.0 | 12.5 h | 1.0 | 14.3 h | 1.0 | N | N |

s: seconds; m: minutes; h: hours; N: no result because the running time exceeds 48 hours.

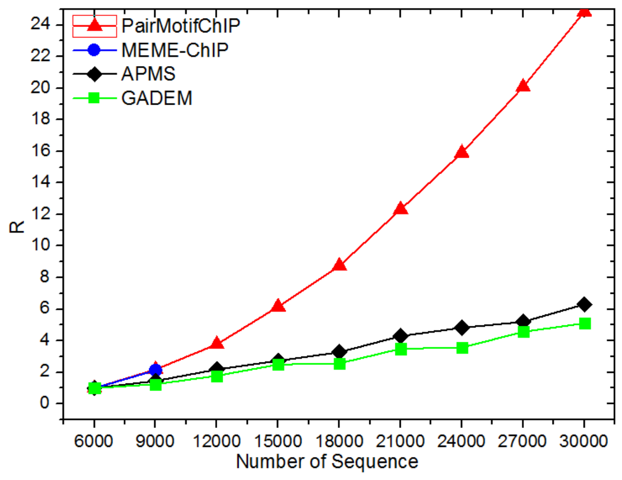


**FIGURE 2.** **Increasing trend of the running time of the compared algorithms with data scale. *R* is the ratio of running time to that on 6000 sequences.**

time of GADEM also shows linear growth, it is already relatively long (close to 3 hours) in processing 6000 sequences.

In addition, we can analyze how the performance of these algorithms is affected by the parameters $t$, $l$, $d$ and $q$ from the experimental results. For PairMotifChIP, GADEM and MEME-ChIP, their running time is mainly affected by the number of sequences $t$. APMS is slightly affected by $(l, d)$ besides the influence of $t$. Specifically, as shown in Table 1, the running time of APMS increases with the increase of $(l, d)$. The reason for this phenomenon is that, as $(l, d)$ grows, more $d$-neighbors need to be verified in the refinement process of APMS. FMotif is heavily affected by $(l, d)$, as the number of candidate motifs of FMotif shows exponential growth with the increase of $(l, d)$. FMotif is also affected by $q$. As $q$ decreases, FMotif requires longer running time because the pruning effect during the searching process is weakened.

APMS is an approximate qPMS algorithm. Since it only records the detected $(l, d)$ motif with the maximum score in refining each seed and performs filtration of redundant seeds, it usually just reports one $(l, d)$ motif with high score for the simulated data. It should be noted that, the $(l, d)$ motif reported by APMS is usually the desired motif, i.e., the implanted $(l, d)$ motif. Although exact qPMS algorithms can report all $(l, d)$ motifs present in the dataset, most of these reported motifs are redundant. For example, for the first tested dataset under the (9, 2) problem instance in Table 1, FMotif (the exact qPMS algorithm) reports 36 $(l, d)$ motifs, including the implanted $(l, d)$ motif $m$ = AAACTCGAG. APMS just reports $m$. The 35 $(l, d)$ motifs missed by APMS consist of 8 motifs (e.g., CAAACTCGA) that overlap $m$ by 8 bases and 27 motifs (e.g., AAACTCAAG) that differ from $m$ in one position.

### B. RESULTS ON REAL DATA

Experiments on real data are mainly used to verify the validity of APMS, that is, to verify whether APMS can efficiently identify motifs in real biological data. We use the mouse embryonic stem cell (mESC) ChIP-seq data [30] as the first group of real datasets, which are widely used for the verification of the validity of motif discovery algorithms. The mESC data contain 12 datasets, each of which is named after the ChIP-ed TF. Moreover, we collect 149 ChIP-seq datasets of the human species from the ENCODE database [31] as the second group of real datasets. Each of these collected dataset contains a TF-binding motif documented in JASPAR database [32]. In the experiments, the first 3000 sequences of each dataset are taken as the input. We run APMS on different datasets by using uniform parameter settings, that is, $(l, d) = (13, 4)$, $q = 0.3$ and $g = 0.5$.

Fig. 3 shows the experimental results of APMS on the first group of real datasets. For each dataset, we have shown the running time of APMS, the published motif (the one on top) and the predicted motif (the one below) in the form of the sequence logo [27]. First, by comparing the predicted motifs with the published motifs, we find that APMS can identify the motifs similar to real motifs on all these datasets. Second, the running time on all these datasets is within 6 minutes.

Table 4 shows the experimental results of APMS on the second group of real datasets. These datasets cover seven cell lines. For each cell line, we have shown the number of collected datasets, the number of datasets with the real motif found and the average running time. More detailed results of APMS for each dataset (including the running time, the predicted motif in the form of both the sequence logo and PWM, and the published motif in the form of both the sequence logo and PWM) are included in detailedResults document at https://github.com/qyu071/apms. APMS can find real motifs

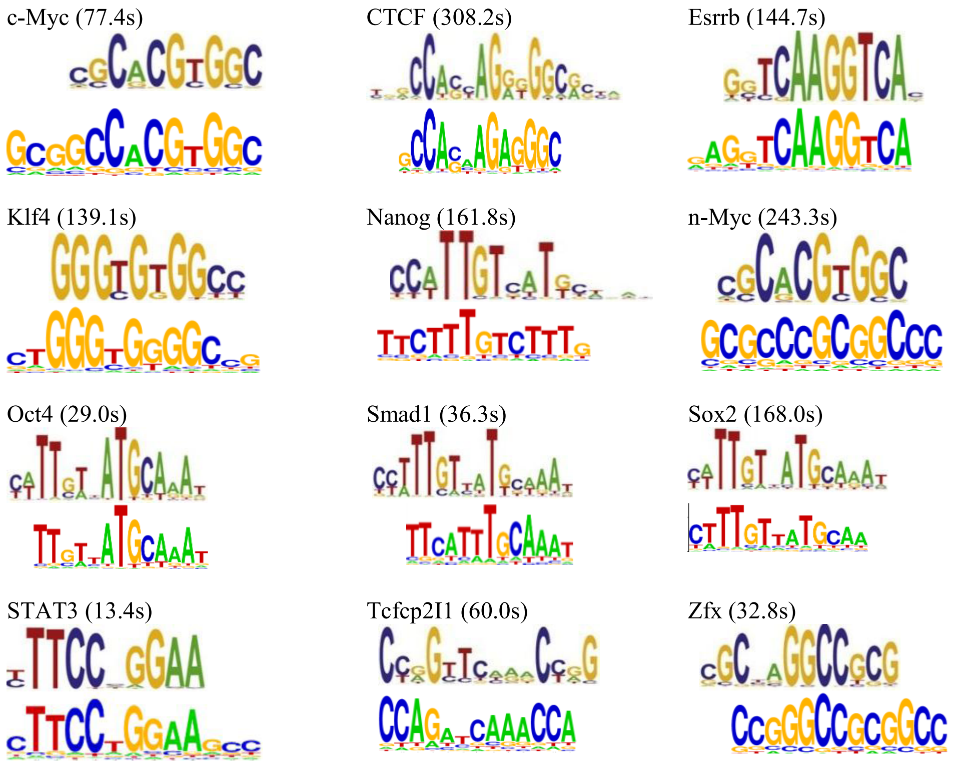


**FIGURE 3.** Results on the first group of real datasets (the mESC data).

**TABLE 4.** Results on the second group of real datasets (the datasets of human species collected from ENCODE database).

| Cell line | Number of collected datasets | Number of datasets with the real motif found | | Average running time | |
|---|---|---|---|---|---|
| | | APMS | GADEM | APMS | GADEM |
| GM12878 | 36 | 31 | 24 | 5.5 m | 3.3 h |
| GM12891 | 3 | 2 | 2 | 3.4 m | 2.8 h |
| H1 | 21 | 19 | 18 | 4.3 m | 3.0 h |
| HeLa-S3 | 18 | 13 | 11 | 5.3 m | 3.4 h |
| HepG2 | 29 | 23 | 23 | 2.9 m | 2.5 h |
| IMR90 | 3 | 3 | 2 | 1.5 m | 2.0 h |
| K562 | 39 | 34 | 32 | 5.1 m | 3.1 h |

on the majority of the datasets for each cell line and on the 83.9% of all these 149 datasets. The average running time of APMS is still within 6 minutes. Moreover, we take GADME as a reference algorithm and run it by using default parameter settings. Compared with GADEM, APMS can not only find real motifs on more datasets, but also requires much less time. In summary, APMS can effectively and efficiently process large datasets in reality.

## IV. CONCLUSION

In order to efficiently solve the planted motif search on large DNA datasets, we propose an approximate qPMS algorithm called APMS by designing new methods for generating seeds, refining seeds and filtering redundant seeds. The running time of the proposed algorithm shows linear growth with the increase of the data scale. Experimental results show that the proposed algorithm can not only successfully find the implanted or real motifs, but also run much faster than the compared algorithms. The source code of APMS, the python wrapper for the code and the test data are available for download at https://github.com/qyu071/apms. The limitations of APMS are that the type of motifs it can search is the ($l$, $d$) motifs and the parameters $l$, $d$ and $q$ need to be provided.

## ACKNOWLEDGMENTS

The authors express their sincere appreciation to the editors and specialist reviewers for their instructions and help improving the article.

## REFERENCES

[1] A. M. Khamis, O. Motwalli, R. Oliva, B. R. Jankovic, Y. A. Medvedeva, H. Ashoor, M. Essack, X. Gao, and V. B Bajic, "A novel method for improved accuracy of transcription factor binding site prediction," *Nucleic Acids Res.*, vol. 46, no. 12, p. e72, 2018.

[2] X. Yuan, M. Gao, J. Bai, and J. Duan, "SVSR: A program to simulate structural variations and generate sequencing reads for multiple platforms," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, to be published. doi: 10.1109/TCBB.2018.2876527.

[3] P. A. Pevzner and S. H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," in *Proc. 8th Int. Conf. Intell. Syst. Mol. Biol.*, 2000, pp. 269–278.

[4] J. Davila, S. Balla, and S. Rajasekaran, "Fast and practical algorithms for planted $(l, d)$ motif search," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 4, no. 4, pp. 544–552, Oct./Dec. 2007.

[5] P. A. Evans, A. D. Smith, and H. T. Wareham, "On the complexity of finding common approximate substrings," *Theor. Comput. Sci.*, vol. 306, pp. 407–430, Sep. 2003.

[6] X. Yuan, J. Zhang, and L. Yang, "IntSIM: An integrated simulator of next-generation sequencing data," *IEEE Trans. Biomed. Eng.*, vol. 64, no. 2, pp. 441–451, Feb. 2017.

[7] T. Bailey, P. Krajewski, I. Ladunga, C. Lefebvre, Q. Li, T. Liu, P. Madrigal, C. Taslim, and J. Zhang, "Practical guidelines for the comprehensive analysis of ChIP-Seq data," *PLoS Comput. Biol.*, vol. 9, no. 11, 2013, Art. no. e1003326.

[8] F. Zambelli, G. Pesole, and G. Pavesi, "Motif discovery and transcription factor binding sites before and after the next-generation sequencing era," *Briefings Bioinf.*, vol. 14, no. 2, pp. 225–237, 2013.

[9] B. Liu, J. Yang, Y. Li, A. Mcdermaid, and Q. Ma, "An algorithmic perspective of *de novo* cis-regulatory motif finding based on ChIP-Seq data," *Briefings Bioinf.*, vol. 19, no. 5, pp. 1069–1081, 2018.

[10] N. Jayaram, D. Usvyat, and A. C. R. Martin, "Evaluating tools for transcription factor binding site prediction," *BMC Bioinf.*, vol. 17, Nov. 2016, Art. no. 1298.

[11] Q. Yu, H. Huo, J. S. Vitter, J. Huan, and Y. Nekrich, "An efficient exact algorithm for the motif stem search problem over large alphabets," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 12, no. 2, pp. 384–397, Mar. 2015.

[12] H. Dinh, S. Rajasekaran, and J. Davila, "qPMS7: A fast algorithm for finding $(\ell, d)$-motifs in DNA and protein sequences," *PLoS ONE*, vol. 7, no. 7, 2012, Art. no. e41425.

[13] S. Tanaka, "Improved exact enumerative algorithms for the planted $(l, d)$-motif search problem," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 11, no. 2, pp. 361–374, Mar./Apr. 2014.

[14] M. Nicolae and S. Rajasekaran, "Efficient sequential and parallel algorithms for planted motif search," *BMC Bioinf.*, vol. 15, Jan. 2014, Art. no. 34.

[15] M. Nicolae and S. Rajasekaran, "qPMS9: An efficient algorithm for quorum planted motif search," *Sci. Rep.*, vol. 5, Jan. 2015, Art. no. 7813.

[16] G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole, "Weeder Web: Discovery of transcription factor binding sites in a set of sequences from co-regulated genes," *Nucleic Acids Res.*, vol. 32, pp. W199–W203, Jul. 2004.

[17] N. Pisanti, A. M. Carvalho, L. Marsan, and M.-F. Sagot, "RISOTTO: Fast extraction of motifs with mismatches," in *Proc. 7th Latin Amer. Symp. Theor. Inform.*, in Lecture Notes in Computer Science, 2006, pp. 757–768.

[18] C. Jia, M. B. Carson, Y. Wang, Y. Lin, and H. Lu, "A new exhaustive method and strategy for finding motifs in ChIP-enriched regions," *PLoS ONE*, vol. 9, no. 1, 2014, Art. no. e86044.

[19] P. Machanick and T. L. Bailey, "MEME-ChIP: Motif analysis of large DNA datasets," *Bioinformatics*, vol. 27, no. 12, pp. 1696–1697, 2011.

[20] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment," *Science*, vol. 262, pp. 208–214, Oct. 1993.

[21] N. K. Lee, X. Li, and D. Wang, "A comprehensive survey on genetic algorithms for DNA motif prediction," *Inf. Sci.*, vol. 466, pp. 25–43, Oct. 2018.

[22] L. Li, "GADEM: A genetic algorithm guided formation of spaced dyads coupled with an EM algorithm for motif discovery," *J. Comput. Biol.*, vol. 16, no. 2, pp. 317–329, 2009.

[23] Q. Yu, H. Huo, and D. Feng, "PairMotifChIP: A fast algorithm for discovery of patterns conserved in large ChIP-Seq data sets," *BioMed Res. Int.*, vol. 2016, Sep. 2016, Art. no. 4986707.

[24] V. Boeva, D. Surdez, N. Guillon, F. Tirode, A. P. Fejes, O. Delattre, and E. Barillot, "*De novo* motif identification improves the accuracy of predicting transcription factor binding sites in ChIP-Seq data analysis," *Nucleic Acids Res.*, vol. 38, p. e126, Jun. 2010.

[25] P. Xiao, S. Pal, and S. Rajasekaran, "Randomised sequential and parallel algorithms for efficient quorum planted motif search," *Int. J. Data Mining Bioinf.*, vol. 18, no. 2, pp. 105–124, 2017.

[26] Q. Yu, D. Wei, and H. Huo, "SamSelect: A sample sequence selection algorithm for quorum planted motif search on large DNA datasets," *BMC Bioinf.*, vol. 19, Jun. 2018, Art. no. 228.

[27] G. E. Crooks, G. Hon, J.-M. Chandonia, and S. E. Brenner, "WebLogo: A sequence logo generator," *Genome Res.*, vol. 14, pp. 1188–1190, Jun. 2004.

[28] G. Ciriello and C. Guerra, "A review on models and algorithms for motif discovery in protein–protein interaction networks," *Briefings Funct. Genomics Proteomics*, vol. 7, no. 2, pp. 147–156, 2008.

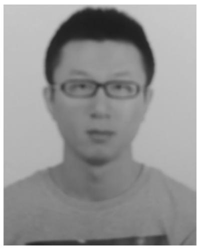
[29] H. Hartmann, E. W. Guthöhrlein, M. Siebert, S. Luehr, and J. Söding, "*P*-value-based regulatory motif discovery using positional weight matrices," *Genome Res.*, vol. 23, no. 1, pp. 181–194, 2013.

[30] X. Chen *et al.*, "Integration of external signaling pathways with the core transcriptional network in embryonic stem cells," *Cell*, vol. 133, no. 6, pp. 1106–1117, 2008.

[31] P. Kheradpour and M. Kellis, "Systematic discovery and characterization of regulatory motifs in ENCODE TF binding experiments," *Nucleic Acids Res.*, vol. 42, no. 5, pp. 2976–2987, 2014.

[32] A. Khan, O. Fornes, A. Stigliani, M. Gheorghe, J. A. Castro-Mondragon, R. van der Lee, S. R. Kulkarni, G. Tan, D. Baranasic, D. J. Arenillas, A. Sandelin, K. Vandepoele, B. Lenhard, B. Ballester, W. W. Wasserman, F. Parcy, and A. Mathelier, "JASPAR 2018: Update of the open-access database of transcription factor binding profiles and its Web framework," *Nucleic Acids Res.*, vol. 46, pp. D260–D266, Jan. 2018.

**QIANG YU** received the B.S., M.S., and Ph.D. degrees from Xidian University, China, in 2006, 2009, and 2014, respectively, where he is currently an Associate Professor of computer science and technology. His research interests include the design and analysis of algorithms, computational biology, and bioinformatics.



**XIAO ZHANG** received the B.S. degree from the Chongqing University of Posts and Telecommunications, China, in 2018. He is currently pursuing the M.S. degree with Xidian University, China. His research interests include the design and analysis of algorithms, computational biology, and bioinformatics.