qPMS7: A Fast Algorithm for Finding (ℓ, d) -Motifs in DNA and Protein Sequences

Hieu Dinh^{1*}, Sanguthevar Rajasekaran^{1*}, Jaime Davila²

1 Department of Computer Science and Engineering, University of Connecticut, Storrs, Connecticut, United States of America, **2** Division of Biomedical Statistics and Informatics, Mayo Clinic, Rochester, Minnesota, United States of America

Abstract

Detection of rare events happening in a set of DNA/protein sequences could lead to new biological discoveries. One kind of such rare events is the presence of patterns called motifs in DNA/protein sequences. Finding motifs is a challenging problem since the general version of motif search has been proven to be intractable. Motifs discovery is an important problem in biology. For example, it is useful in the detection of transcription factor binding sites and transcriptional regulatory elements that are very crucial in understanding gene function, human disease, drug design, etc. Many versions of the motif search problem have been proposed in the literature. One such is the (ℓ, d) -motif search (or *Planted Motif Search (PMS)*). A generalized version of the PMS problem, namely, *Quorum Planted Motif Search (qPMS)*, is shown to accurately model motifs in real data. However, solving the qPMS problem is an extremely difficult task because a special case of it, the PMS Problem, is already NP-hard, which means that any algorithm solving it can be expected to take exponential time in the worse case scenario. In this paper, we propose a novel algorithm named qPMS7 that tackles the qPMS problem on real data as well as challenging instances. Experimental results show that our Algorithm qPMS7 is on an average 5 times faster than the state-of-art algorithm. The executable program of Algorithm qPMS7 is freely available on the web at <http://pms.engr.uconn.edu/downloads/qPMS7.zip>. Our online motif discovery tools that use Algorithm qPMS7 are freely available at <http://pms.engr.uconn.edu> or <http://motifsearch.com>.

Citation: Dinh H, Rajasekaran S, Davila J (2012) qPMS7: A Fast Algorithm for Finding (ℓ, d) -Motifs in DNA and Protein Sequences. PLoS ONE 7(7): e41425. doi:10.1371/journal.pone.0041425

Editor: Vladimir Brusic, Dana-Farber Cancer Institute, United States of America

Received: March 12, 2012; **Accepted:** June 21, 2012; **Published:** July 24, 2012

Copyright: © 2012 Dinh et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Funding: This work has been supported in part by the following grants: NSF 0829916 and NIH R01-LM010101. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. No additional external funding was received for this study.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: hdinh@engr.uconn.edu (HD); rajasek@engr.uconn.edu (SR)

Introduction

Detection of rare events happening in a set of DNA/protein sequences often provides the main clue leading to new biological discoveries. One kind of such rare events is the presence of patterns called motifs. For example, regulatory regions in a genome such as promoters, enhancers, locus control regions, etc., contain motifs that control many biological processes such as gene expression (see [1]). Basically, proteins known as transcription factors regulate the expression of a gene by binding to locations of motifs in regulatory regions. For instance, transcription factors such as TFIID, TFIIA and TFIIB usually bind to sequence 5'-TATAAA-3' in the promoter region of a gene in order to initiate its transcription. Such motifs and their locations in regulatory regions, i.e., binding sites, are important and helpful to decipher the regulatory mechanism of gene expression, which is very sophisticated. As a result, motif identification plays an important role in biological studies.

Motif prediction is usually the first stage in the process of identifying motifs. An extensive amount of research has been done on this topic over the past twenty years. In the literature, many approaches for motif prediction have been proposed. One of them is a combinatorial approach that has proven to be more accurate than the others. Even in this combinatorial approach, many variations can be found such as *Planted Motif Search (PMS)*, *Simple*

Motif Search (SMS), and *Edit-distance-based Motif Search (EMS)* (see e.g., [2]).

Among the combinatorial variations, the PMS Problem has been the most widely studied perhaps because it offers a higher level of accuracy in modeling the true motifs than the others. Motifs typically occur with mutations at binding sites. The binding sites are referred to as *instances* of a motif. A motif in this model is referred to as a (ℓ, d) -motif where ℓ is its length and d is the maximum number of mutations allowed for its instances. Given a set of sequences, the objective of the PMS problem is to find all the (ℓ, d) -motifs in them. The formal definition of the PMS Problem is given in Section 0.1. An algorithm that solves the PMS problem is called a *PMS Algorithm*.

Owing to its importance, the PMS problem has been extensively studied in the past twenty years. Many PMS algorithms have been proposed in the literature. There are two kinds of PMS algorithms, namely, *exact* and *approximate*. An exact algorithm always finds all the (ℓ, d) -motifs present in the input sequences. An approximate algorithm may not find all the motifs. In this paper we only consider exact algorithms. The (exact variant of the) PMS problem has been shown to be NP-hard which means that there is unlikely to be a PMS algorithm that takes only polynomial time. As a result, all the existing exact PMS Algorithms take time that is exponential time in some of the parameters in the worst case. In practice, all known PMS Algorithms (both exact and approximate)

are only able to find (ℓ, d) -motifs for up to certain values of ℓ and d . The most recent exact algorithms that have been proposed in the literature are Algorithm PMS6 due to [3], Algorithm PMS5 due to [4], Algorithm Pampa due to [5], Algorithm PMSPrune due to [6], Algorithm PMS3 due to [7], Algorithm Voting due to [8], and Algorithm RISSOTO due to [9]. Some earlier PMS algorithms are due to [10], [11], [12], [13], [14], [15], [16], and [17]. Among these known algorithms, Algorithm PMS6 is considered the fastest one and has been developed closely following the ideas of Algorithm PMS5.

Approximate PMS algorithms usually tend to be faster than exact PMS algorithms. Typically, approximate PMS algorithms employ heuristics such as local search, Gibbs sampling, expectation optimization, etc. Examples of approximate algorithms are Algorithm MEME due to [18], Algorithm PROJECTION due to [19], Algorithm GibbsDNA due to [20], Algorithm WINNOWER due to [21], and Algorithm RandomProjection due to [22]. Some other approximate PMS algorithms are Algorithm MULTIPROFILER due to [23], Algorithm PatternBranching due to [24], Algorithm ProfileBranching due to [24], and Algorithm CONSENSUS due to [25].

A generalized version of the PMS Problem, namely *Quorum Planted Motif Search* (qPMS) Problem, was first considered in [6]. The qPMS problem is to find all the motifs that have motif instances present in q out of the n input sequences. The qPMS problem captures the nature of motifs more precisely than the PMS problem does because, in practice, some motifs may not have motif instances in all of the input sequences. The qPMS problem is formally defined in Section 0.1. An algorithm that solves the qPMS problem is called a qPMS algorithm. qPMS algorithms can be used to find DNA motifs and protein motifs as well as transcription factor binding sites. The larger the values of ℓ and d that a qPMS algorithm can handle, the more accurate will be the motifs it finds. So it is important to solve the qPMS problem instances with large values of ℓ and d . However, solving the qPMS problem is a difficult task since it is even harder than the PMS problem. To the best of our knowledge, the currently best exact qPMS algorithm is Algorithm qPMSPrune due to [6] that can only solve instances up to $\ell = 17$ and $d = 5$ for $q = \frac{n}{2}$, where n is the number of input sequences. In this paper, we propose a new algorithm named Algorithm qPMS7 that can solve larger instances. Also, qPMS7 is ten times as fast as qPMSPrune. In addition, when applied to the PMS problem, our algorithm is faster than the best PMS algorithm, i.e., Algorithm PMS6 due to [3].

Methods

0.1 Problems Definition and Notations

Definition 0.1 A string $x = x[1] \dots x[\ell]$ of length ℓ is called an ℓ -mer.

Definition 0.2 Given two $x = x[1] \dots x[\ell]$ and $s = s[1] \dots s[m]$ with $\ell < m$, we use the notation $x \in_\ell s$ if x is a contiguous substring of s . In other words, $x \in_\ell s$ if there exists $1 \leq i \leq m - \ell + 1$ such that $x[j] = s[j + i - 1]$ for every $1 \leq j \leq \ell$. We also say that x is an ℓ -mer in s .

Definition 0.3 Given two strings $x = x[1] \dots x[\ell]$ and $y = y[1] \dots y[\ell]$ of equal length, the Hamming distance between x and y , denoted by $d_H(x, y)$, is the number of mismatches between them. In other words, $d_H(x, y) = \sum_{1 \leq i \leq \ell} I_i$, where I_i is the indicator at position i . $I_i = 1$ if $x[i] \neq y[i]$, and $I_i = 0$ otherwise.

Definition 0.4 Given two strings x and s with $|x| < |s|$, the Hamming distance between x and s , denoted by $d_H(x, s)$, is $\min_{y \in_\ell s} d_H(x, y)$.

Definition 0.5 Given a set of n strings $s_1, \dots, s_n$ of length m each, a string M of length ℓ is called an (ℓ, d, q) -motif of the strings if there are at least q out of the n strings such that the Hamming distance between each one of them and M is no more than d . M is called an (ℓ, d, q) -motif for short if the set of strings is clear.

The definition of *Quorum Planted Motif Search* (qPMS) Problem is as follows. Given n input strings $s_1, \dots, s_n$ of length m each, three integer parameters ℓ , d and q , find all the (ℓ, d, q) -motifs of the input strings. The *Planted Motif Search* (PMS) problem is a special case of the qPMS problem when $q = n$. In this paper, we propose a fast algorithm for the qPMS problem.

0.2 The Existing Algorithm qPMSPrune

Algorithm qPMSPrune for the qPMS problem was proposed by [6]. For the sake of completeness, we will describe Algorithm qPMSPrune in this section briefly because our new algorithm is partially based on it. For more details on Algorithm qPMSPrune, the readers are referred to [6].

Algorithm qPMSPrune uses the d -neighborhood concept defined as follows.

Definition 0.6 Given a string $x = x[1] \dots x[\ell]$, we define the d -neighborhood of x , $B_d(x)$, to be $\{y | d_H(x, y) \leq d\}$.

It is easy to see that $|B_d(x)| = \sum_{i=0}^d \binom{\ell}{i} (|\Sigma| - 1)^i$, where Σ is the alphabet of interest. Notice that $|B_d(x)|$ depends only on ℓ , d and $|\Sigma|$. For this reason, we define $\mathcal{N}(\ell, d) = \sum_{i=0}^d \binom{\ell}{i} (|\Sigma| - 1)^i$.

Algorithm qPMSPrune is based on the following observation. Any (ℓ, d, q) -motif of the input strings must be in $B_d(x)$ for some ℓ -mer x in some input string s_i and also it must be a $(\ell, d, q-1)$ -motif of the input strings excluding s_i . This observation can be rewritten formally as follows.

Observation 0.1 Let M be any (ℓ, d, q) -motif of the input strings $s_1, \dots, s_n$. Then there exists an i (with $1 \leq i \leq n$) and a ℓ -mer $x \in_\ell s_i$ such that M is in $B_d(x)$ and M is a $(\ell, d, q-1)$ -motif of the input strings excluding s_i .

The above observation suggests the following algorithm. Compute $B_d(x)$ for every ℓ -mer x in each input string s_i for $1 \leq i \leq n$. For each ℓ -mer in the neighborhoods thus computed, check if it is a $(\ell, d, q-1)$ -motif of the input strings excluding s_i . This simple algorithm can be improved further as shown in [6]. The key observation is that it is sufficient to consider each input string s_i for $1 \leq i \leq n - q + 1$:

Observation 0.2 Let M be any (ℓ, d, q) -motif of the input strings $s_1, \dots, s_n$. Then there exists an i (with $1 \leq i \leq n - q + 1$) and a ℓ -mer $x \in_\ell s_i$ such that M is in $B_d(x)$ and M is a $(\ell, d, q-1)$ -motif of the input strings excluding s_i .

Algorithm qPMSPrune is based on the above observation. For any ℓ -mer x , it represents $B_d(x)$ as a tree $\mathcal{T}_d(x)$ using the following rules.

- Each node in $\mathcal{T}_d(x)$ is a pair (t, p) where $t = t[1] \dots t[\ell]$ is an ℓ -mer and p is an integer between 0 and ℓ such that $t[p+1] \dots t[\ell] = x[p+1] \dots x[\ell]$. A node (t, p) is referred to as a ℓ -mer t if p is clear.
- Let $t = t[1] \dots t[\ell]$ and $t' = t'[1] \dots t'[\ell]$. A node (t, p) is the parent of a node (t', p') if and only if
 - $p' > p$.
 - $t'[p'] \neq t[p']$ (From Rule 1, $t[p'] = x[p']$).
 - $t'[i] = t[i]$ for any $i \neq p'$.
- The root of $\mathcal{T}_d(x)$ is $(x, 0)$.

4. The depth of $\mathcal{T}_d(x)$ is d .

For example, the tree $\mathcal{T}_2(1010)$ with alphabet $\Sigma = \{0,1\}$ is illustrated in Figure 1.

Clearly, the following properties of $\mathcal{T}_d(x)$ can be inferred directly from the rules.

1. Each node in $\mathcal{T}_d(x)$ is uniquely associated with an ℓ -mer in $B_d(x)$ and vice versa.
2. If a node (t, p) is a parent of a node (t', p') , then $d_H(x, t') - d_H(x, t) = d_H(t, t') = 1$. As a result, if a node (t, p) is at level h , then $d_H(x, t) = h$.

The algorithm traverses the tree $\mathcal{T}_d(x)$ in a depth-first manner. At each node (t, p) , it computes $d_H(t, s_j)$ incrementally from its parent for $1 \leq j \leq n, j \neq i$. This operation can be done in $O(nm)$ time by the incremental computation shown in [6]. Let q' be the number of input strings s_j such that $d_H(t, s_j) \leq d$. Obviously if $q' \geq q - 1$ then t is a $(\ell, d, q - 1)$ -motif of the input strings excluding s_i . If this condition is satisfied, it outputs t as a (ℓ, d, q) -motif of the entire input strings.

Algorithm qPMSPrune prunes certain nodes (and their descendants) in $\mathcal{T}_d(x)$ that cannot possibly be (ℓ, d, q) -motifs. Under what conditions can we prune the node (t, p) ? Let q'' be the number of input strings s_j such that $d_H(t, s_j) \leq 2d - d_H(t, x)$. Observe that if $q'' < q - 1$ then none of the nodes in the subtree rooted at node (t, p) could be a $(\ell, d, q - 1)$ -motif. This is because if there is a node (t', p') in the subtree which is a $(\ell, d, q - 1)$ -motif, then there are at least $q - 1$ input strings s_j such that $d_H(t', s_j) \leq d$. Consider such an input string s_j . By the triangle inequality, $d_H(t, s_j) \leq d_H(t', s_j) + d_H(t, t') \leq d + (d_H(t', x) - d_H(t, x)) \leq 2d - d_H(t, x)$. This inequality will infer that $q'' \geq q - 1$. Therefore, if the condition $q'' < q - 1$ occurs, it can safely prune the subtree rooted at node (t, p) without missing any $(\ell, d, q - 1)$ -motif. The pseudo-code of Algorithm qPMSPrune is described as follows.

Algorithm qPMSPrune

For each $x \in \mathcal{I}_\ell, 1 \leq i \leq n - q + 1$ do:

Traverse the tree $\mathcal{T}_d(x)$ in a depth-first manner. At each node (t, p) , do the following steps.

- i. Incrementally compute $d_H(t, s_j)$ from its parent for $1 \leq j \leq n, j \neq i$.
- ii. Let q' be the number of input strings s_j such that $d_H(t, s_j) \leq d$. If $q' \geq q - 1$, output t .
- iii. Let q'' be the number of input strings s_j such that $d_H(t, s_j) \leq 2d - d_H(t, x)$. If $q'' < q - 1$, then prune the subtree rooted at node (t, p) . Otherwise, explore its children.

Figure 1. Traverse the tree in qPMSPrune. $\mathcal{T}_2(1010)$ with alphabet $\Sigma = \{0,1\}$. The value of p at each node is the location of its shaded letter. For example, $p = 1$ at node 0010, $p = 4$ at node 0011. doi:10.1371/journal.pone.0041425.g001

It is easy to see that the time and space complexities of Algorithm qPMSPrune are $O((n - q + 1)nm^2\mathcal{N}(\ell, d))$ and $O(nm^2)$, respectively.

0.3 A Computational Technique Improving upon

Algorithm qPMSPrune

In this section, we propose a speedup technique to improve the runtime of Algorithm qPMSPrune. Specifically, the technique will reduce the time taken for computing Hamming distances $d_H(t, s_j)$ in step (1) of Algorithm qPMSPrune. Recall that the operation takes at least $\Omega(nm)$ time in Algorithm qPMSPrune because it considers every ℓ -mer in each input string s_j . We observe that some ℓ -mers can be ignored without changing the result since we notice that we just need to count q' and q'' . Any ℓ -mer z in s_j can be ignored, as far as a node (t, p) in the tree $\mathcal{T}_d(x)$ is concerned, if $d_H(t, z) > 2d - d_H(t, x)$. The reason for this will be given in the next paragraph. Based on this observation, the technique is implemented as follows. At each node (t, p) , we store a list of surviving ℓ -mers for each input string s_j . It is sufficient to store the positions of the ℓ -mers in the input strings. If the list of surviving ℓ -mers of s_j is empty, then we set $d_H(t, s_j) = \infty$. In terms of the incremental distance computation, only the surviving ℓ -mers are considered. The runtime of the operation now depends on the sizes of the lists of surviving ℓ -mers.

The reason for ignoring any ℓ -mer z in s_j , as far as a node (t, p) in the tree $\mathcal{T}_d(x)$ is concerned, if $d_H(t, z) > 2d - d_H(t, x)$ is as follows. If this condition occurs, then for any node (t', p') in the subtree rooted at node (t, p) we have: $d_H(t', z) \geq d_H(t, z) - d_H(t, t') > 2d - d_H(t, x) - d_H(t, t') = 2d - d_H(t', x) \geq d$. Therefore, ignoring ℓ -mer z at any node (t', p') in the subtree rooted at node (t, p) will not change its q' . The value of q'' at node (t', p') may become smaller as a result of ignoring the ℓ -mer z . However, the pruning condition based on q'' in step (3) in the pseudo-code still holds.

Another way to view the ignoring condition is as follows. Consider a node (t, p) in the tree $\mathcal{T}_d(x)$ and an ℓ -mer z in the input string s_j . Let us separate each of t, x and z into two parts based on p , namely, $t = t_1 t_2, x = x_1 x_2$ and $z = z_1 z_2$ where $p = |t_1| = |x_1| = |z_1|$ and $\ell - p = |t_2| = |x_2| = |z_2|$. Notice that $t_2 = x_2$. Then the inequality $d_H(t, z) > 2d - d_H(t, x)$ is equivalent to $d_H(x_2, z_2) > d - d_H(t_1, x_1) + d - d_H(t_1, z_1)$. In other words, $B_{d-d_H(t_1, x_1)}(x_2)$ and $B_{d-d_H(t_1, z_1)}(z_2)$ are disjoint. Notice that this condition is independent of t_2 . This view helps us in designing our best algorithm qPMS7 which is described in Section 0.4.

The speedup technique reduces the runtime of Algorithm qPMSPrune drastically because the deeper a node is, the smaller will be the size of its list of surviving ℓ -mers. Note that the number of nodes at a depth of h from the root will be exponential in h . In practice, the runtime of Algorithm qPMSPrune is improved by a factor of around 5 when this technique is used (see Table 1 and Table 2). However, it does not change the worst case time complexity of Algorithm qPMSPrune, theoretically.

0.4 Our Best Algorithm qPMS7

In this section, we propose a fast algorithm called qPMS7 for the qPMS problem. Algorithm qPMS7 is a generalized version of Algorithm qPMSPrune combined with the core idea of Algorithm PMS5 which was introduced in [4].

Recall that Algorithm qPMSPrune considers one ℓ -mer x in a specific input string s_i at a time. Algorithm qPMS7 extends Algorithm qPMSPrune by considering two ℓ -mers x and y in two different input strings s_i and s_j . An observation similar to that of Algorithm qPMSPrune can be obtained as follows.

Table 1. Time comparison of different algorithms on the challenging instances of DNA sequences for the special case - PMS Problem.

Algorithm	(13,4)	(15,5)	(17,6)	(19,7)	(21,8)	(23,9)
qPMS7	47 s	2.6 m	11 m	0.9 h	4.3 h	24 h
PMS6	67 s	3.2 m	14 m	1.16 h	5.8 h	–
PMS5	117 s	4.8 m	21.7 m	1.7 h	9.7 h	54 h
qPMSPrune1	17 s	2.6 m	22.6 m	3.4 h	29 h	–
Pampa	35 s	6 m	40 m	4.8 h	–	–
qPMSPrune	45 s	10.2 m	78.7 m	15.2 h	–	–
Voting	104 s	21.6 m	–	–	–	–
RISOTTO	772 s	106 m	–	–	–	–

The alphabet $\Sigma = \{A, C, G, T\}$, $n = 20$, $m = 600$, and $q = n = 20$.
doi:10.1371/journal.pone.0041425.t001

Observation 0.3 Let M be any (ℓ, d, q) -motif of the input strings $s_1, \dots, s_n$. Then there exist $1 \leq i \neq j \leq n$ and ℓ -mer $x \in \ell s_i$ and ℓ -mer $y \in \ell s_j$ such that M is in $B_d(x) \cap B_d(y)$ and M is a $(\ell, d, q-2)$ -motif of the input strings excluding s_i and s_j .

Using an argument similar to the one in [6], we infer that it is enough to consider every pair of input strings s_i and s_j with $1 \leq i, j \leq (n-q+2)$. As a result, the above observation gets strengthened as follows.

Observation 0.4 Let M be any (ℓ, d, q) -motif of the input strings $s_1, \dots, s_n$. Then there exist $1 \leq i \neq j \leq n-q+2$ and ℓ -mer $x \in \ell s_i$ and ℓ -mer $y \in \ell s_j$ such that M is in $B_d(x) \cap B_d(y)$ and M is a $(\ell, d, q-2)$ -motif of the input strings excluding s_i and s_j .

Like Algorithm qPMSPrune, Algorithm qPMS7 uses a routine that finds all of the motifs M such that M is in $B_d(x) \cap B_d(y)$ and is a $(\ell, d, q-2)$ -motif of the input strings excluding s_i and s_j . Recall that Algorithm qPMSPrune explores $B_d(x)$ by traversing the tree $\mathcal{T}_d(x)$. In Algorithm qPMS7, we also explore $B_d(x) \cap B_d(y)$ by traversing an acyclic graph, denoted as $\mathcal{G}_d(x, y)$, with similar construction rules. The rules for constructing $\mathcal{G}_d(x, y)$ are given below.

- Each node in $\mathcal{G}_d(x, y)$ is a pair (t, p) where t is an ℓ -mer and p is an integer between 0 and ℓ . A node (t, p) is referred to as ℓ -mer t if p is clear. Let $t = t_1 t_2, x = x_1 x_2$ and $y = y_1 y_2$ where $p = |t_1| = |x_1| = |y_1|$ and $\ell - p = |t_2| = |x_2| = |y_2|$. Node (t, p) must satisfy the following constraints:

- $t_2 = x_2$ if $d_H(t_1, x_1) > d_H(t_1, y_1)$, otherwise, $t_2 = y_2$.
- $d_H(t, x) \leq d$ and $d_H(t, y) \leq d$.

Table 2. Time comparison of different algorithms on the challenging instances of DNA sequences for the generalized case - qPMS Problem.

Algorithm	(13,3)	(15,4)	(17,5)	(19,6)	(21,7)
qPMS7	34 s	2.4 m	16 m	1.8 h	11.6 h
qPMSPrune1	14 s	2 m	21 m	3.9 h	–
qPMSPrune	32 s	9 m	2.6 h	–	–

The alphabet $\Sigma = \{A, C, G, T\}$, $n = 20$, $m = 600$, and $q = \frac{n}{2} = 10$.
doi:10.1371/journal.pone.0041425.t002

- Let $t = t[1] \dots t[\ell]$ and $t' = t'[1] \dots t'[\ell]$. There is an arc from a node (t, p) to a node (t', p') if and only if

- $p' > p$.
- $t'[p'] \neq t[p']$.
- $t'[i] = t[i]$ for any $i < p'$.

It is not hard to see that if we traverse the graph $\mathcal{G}_d(x, y)$ in a depth-first manner starting from node $(x, 0)$, then all the ℓ -mers in $B_d(x) \cap B_d(y)$ will be visited. For example, Figure 2 illustrates the visited nodes in the graph $\mathcal{G}_2(1010, 1100)$ in a depth-first manner starting from node $(x, 0) = (1010, 0)$ where the alphabet $\Sigma = \{0, 1\}$.

Algorithm qPMS7 traverses the graph $\mathcal{G}_d(x, y)$ in a depth-first manner with the starting node $(x, 0)$. During the traversal, at each node (t, d) it computes $d_H(t, s_k)$ incrementally from its parent for $1 \leq j \leq n, k \neq i, k \neq j$. With the same method as the one in Algorithm qPMSPrune, we can achieve this task in $O(nm)$ time. Also, it is easy to see that if $q' \geq q-2$ then t is a $(\ell, d, q-2)$ -motif of the input strings excluding s_i and s_j , where q' is the number of input strings $s_k \neq s_i, s_k \neq s_j$ such that $d_H(t, s_k) \leq d$. If this is the case, it outputs t as a (ℓ, d, q) -motif of the entire input strings.

Algorithm qPMS7 also uses a similar pruning strategy to that of Algorithm qPMSPrune and the speedup technique discussed in Section 0.3. In this case, the speedup technique ignores some ℓ -mers in s_k when computing $d_H(t, s_k)$ at each node (t, p) during the traversal of the graph $\mathcal{G}_d(x, y)$. The ignoring condition of an ℓ -mer z in s_k for this case resembles that in Section 0.3. Let $t = t_1 t_2, x = x_1 x_2, y = y_1 y_2$ and $z = z_1 z_2$ where $p = |t_1| = |x_1| = |y_1| = z_1$. It is not hard to see that ℓ -mer z can be safely ignored if $B_{d-d_H(t_1, x_1)}(x_2) \cap B_{d-d_H(t_1, y_1)}(y_2) \cap B_{d-d_H(t_1, z_1)}(z_2)$ is empty. Checking for this condition can be done in $O(1)$ time using the incremental computation shown in [4]. During the traversal of the graph, at each node (t, p) we also store a list of surviving ℓ -mers for each input string s_k . At node (t, p) , if the list of surviving ℓ -mers of an input string is empty, then the input string will contribute nothing to any descendant node of (t, p) in order for that descendant to be a $(\ell, d, q-2)$ -motif. Therefore, the pruning condition is $q'' < q-2$ where q'' is the number of input strings whose lists of surviving ℓ -mers are not empty. The following pseudo-code describes Algorithm qPMS7.

Algorithm qPMS7

- For each $x \in \ell s_i, y \in \ell s_j, 1 \leq i < j \leq n-q+2$ do:
 - Traverse the graph $\mathcal{G}_d(x, y)$ in a depth-first manner starting from node $(x, 0)$. At each node (t, p) , do the following steps.
- Incrementally compute $d_H(t, s_k)$ from its parent for $1 \leq k \leq n, k \neq i, k \neq j$.

Figure 2. Traverse the graph in qPMS7. Visited nodes in $\mathcal{G}_2(1010, 1100)$ in a depth-first manner when the starting node is $(x, 0) = (1010, 0)$. In this example, $\Sigma = \{0, 1\}$. The value of p at each node is the location of its shaded letter. For example, $p = 1$ at node 0100.
doi:10.1371/journal.pone.0041425.g002

- ii. Let q' be the number of input strings s_j such that $d_H(t, s_j) \leq d$. If $q' \geq q-2$, output t .
- iii. Let q'' be the number of input strings whose lists of surviving ℓ -mers are not empty. If $q'' < q-2$, then backtrack. Otherwise, explore its children.

Theoretically, the time and space complexities of Algorithm qPMS7 are $O((n-q+1)^2 nm^2 \mathcal{N}(\ell, d))$ and $O(nm^2)$, respectively. In the worst case scenario, the runtime of Algorithm qPMS7 is worse than that of Algorithm qPMSPrune by a factor of $n-q+1$. However, Algorithm qPMS7 is much faster than Algorithm qPMSPrune in practice, as shown in Section 0.5.

Algorithm qPMS7 also employs the following observation which has been used in many prior works such as [6] and [26]. Let M be any (ℓ, d) motif in inputs strings $s_1, s_2, \dots, s_n$. Let M_i be an instance of M in s_i (for $1 \leq i \leq n$). Then the Hamming distance between M_i and M_j is $\leq 2d$ for any i and j (with $1 \leq i, j \leq n$). In other words, if M_i is any ℓ -mer in some s_i , then it could possibly be an instance of M only if there are at least $q-1$ out of $n-1$ sequences s_j 's, $j \neq i$, that have an ℓ -mer M_j such that the Hamming distance between M_i and M_j is $\leq 2d$. This observation can be utilized to preprocess the input strings so that for any input string only those ℓ -mers that satisfy the above condition are kept (and the other ℓ -mers are ignored from further processing).

0.5 Transcription Factor Binding Sites Discovery

In this section we will discuss how to use a qPMS Algorithm, e.g., Algorithm qPMS7, to discover transcription factor-binding sites. Given a set of DNA strings that likely contains transcription factor-binding sites, we propose a general framework to find them. The framework consists of two phases. The first phase will select a set of motifs by repeatedly calling the qPMS Algorithm on different values of ℓ, d , and q . The second phase will use a scoring function to eliminate some of the motifs returned in the first phase, and then identify the transcription factor-binding sites based on the surviving motifs.

In the first phase we employ different values, ranging between ℓ_{min} and ℓ_{max} , for the length ℓ of motifs, where ℓ_{min} and ℓ_{max} are user-specified parameters. For each value of ℓ , we let d range from 0 to d_{max} , where d_{max} is another user-specified parameter, and call the best qPMS algorithm (let it be Algorithm $\mathcal{A}$) to find (ℓ, d, q) -motifs. In this process, if some (ℓ, d, q) -motif(s) are found, we add them to the set of motifs. The pseudo-code of the first phase follows.

Phase I: selecting candidate motifs

Input: a set of strings

Parameters: $\ell_{min}, \ell_{max}, d_{max}$ and q

Output: a set of (ℓ, d, q) -motifs $\mathcal{M}$

```

1:  $\mathcal{M} \leftarrow \emptyset$ 
2: for  $\ell = \ell_{min}$  to  $\ell_{max}$  do  $d = 0$  to  $d_{max}$  do
3:   for  $d = 0$  to  $d_{max}$  do
4:     Run the fastest qPMS Algorithm  $\mathcal{A}$  to find  $(\ell, d, q)$ -motifs
       of the input strings
5:     if algorithm  $\mathcal{A}$  takes too long then
6:       Terminate algorithm  $\mathcal{A}$ 
7:       break the for loop of  $d$ 
8:   end if
9:   Let  $\mathcal{M}(\ell, d, q)$  be the set of  $(\ell, d, q)$ -motifs returned by
       algorithm  $\mathcal{A}$ 
10:  if  $\mathcal{M}(\ell, d, q)$  is NOT empty then
11:     $\mathcal{M} \leftarrow \mathcal{M}(\ell, d, q)$ 
12:  break the for loop of  $d$ 
13:  end if
14: end for
```

15: **end for**

In the second phase, we sort the (ℓ, d, q) -motifs according to their scores and pick the top k motifs, where k is a user-specified parameter. For each picked (ℓ, d, q) -motif M and each input string s_i , transcription binding sites in s_i are identified as follows. We consider every ℓ -mer z in s_i and output the location of z in s_i as a transcription binding site if $d_H(M, z) \leq d$. The following pseudo-code describes the second phase.

Phase II: identifying transcription factor binding sites

Input: a set of strings and a set of (ℓ, d, q) -motifs $\mathcal{M}$

Parameters: a scoring function and k

Output: a set of binding sites on the input strings

```

1: Sort  $(\ell, d, q)$ -motifs in  $\mathcal{M}$  according to the scoring function
2: Pick the top  $k$   $(\ell, d, q)$ -motifs in  $\mathcal{M}$  after sorting
3: for each picked  $(\ell, d, q)$ -motif  $M$  do
4:   for each input string  $s_i$  do
5:     Identify all the  $\ell$ -mers  $z$  in  $s_i$  such that  $d_H(M, z) \leq d$ 
6:   Output the location of each such  $\ell$ -mer  $z$  in  $s_i$  as a
       transcription factor binding site
7:   end for
8: end for
```

The accuracy of the framework in discovering transcription factor-binding sites heavily depends on two factors: the qPMS Algorithm and the scoring function. Of course, the faster the qPMS Algorithm is, the more accurate will be the results it provides. Designing fast qPMS algorithms is our main focus because it is a difficult task. On the other hand, the choice of the scoring function is also critical. In general, the scoring function should measure the biological significance of a candidate motif possibly via a probabilistic model. As a rule of thumb, the smaller the probability that a motif appears (by random chance) is, the more likely will it be to be biologically significant. In addition, the impact of the scoring function on the accuracy also depends on the size of the list of candidate motifs $\mathcal{M}$. The larger the size is, the more will be the scoring function's impact. For example, the scoring function called "sequence specificity" is usually used. It is defined to be $f(M) = -\sum_{i=1}^n \log(E(d_H(M, s_i)))$ where $E(d_H(M, s_i))$ is the expected number of times a motif appears in string s_i with up to $d_H(M, s_i)$ mismatches [6].

Results

In this section we evaluate the performance of Algorithm qPMS7 on simulated as well as real data. With simulated data, we compare its runtime with that of other existing algorithms. With real data, we measure the accuracy of qPMS7 in detecting real motifs. Of course, the larger the values of ℓ and d that an algorithm can solve, the more accurate will be the results it yields because it covers a larger search space of motifs.

0.6 Experiments on Simulated Data

We compared the runtime of Algorithm qPMS7 with other well-known algorithms such as Algorithm qPMSPrune of [6], Algorithm PMS6 of [3], Algorithm PMS5 of [4], Algorithm Pampa of [5], Algorithm Voting of [8], and Algorithm RISSOTO of [9]. Recall that among these algorithms, only Algorithm qPMSPrune deals with the qPMS problem. The rest of the algorithms deal with the simpler version, i.e., the PMS problem. The improved Algorithm qPMSPrune in Section 0.3 is named qPMSPruneI. To evaluate the performance of algorithms, we usually test them on challenging and hard instances of the problem. All of these algorithms have been run on the same machine running Windows XP Operating System with a Dual Core Pentium 2.4GHz CPU and 3GB RAM. The experimental

Table 3. Time comparison of different algorithms on the challenging instances of protein sequences for the special case - PMS Problem.

Algorithm	(11,5)	(13,6)	(15,7)	(17,8)	(19,9)
qPMS7	1 m	1.4 m	1.9 m	6.8 m	7.5 m
qPMSPrunel	4.5 m	21 m	2.4 m	17 h	–
qPMSPrunel	12 m	104 m	16 h	–	–

The alphabet size $|\Sigma| = 20$, $n = 20$, $m = 600$, and $q = n = 20$.
doi:10.1371/journal.pone.0041425.t003

results below show that Algorithm qPMS7 is better than any other algorithm.

0.6.1 DNA sequences. Following [21] and [6], the set of input strings of a challenging instance is typically generated as follows. Each input string is a random DNA string drawn according to the i.i.d model. A random ℓ -mer M is chosen as a (ℓ, d, q) -motif and mutations of this ℓ -mer M are planted in q out of the n input strings at random positions. The Hamming distance between M and any of these mutations is at most d . The number of input strings n and the length of each of them m are chosen to be 20 and 600, respectively.

In the case of the PMS problem, $q = n$. The pairs (ℓ, d) corresponding to challenging instances are (13,4), (15,5), (17,6), (19,7), (21,8), (23,9), (25,10), and so on. To the best of our knowledge, there has not been any algorithm that can solve the challenging instance (25,10). Therefore, Table 1 reports the runtime of the algorithms on the challenging instances up to (23,9). Algorithms qPMS7, PMS6 and PMS5 can solve any of these challenging instances. In Table 1, the letter ‘–’ indicates that the corresponding algorithm either takes too long or uses too much memory on the corresponding challenging instance.

Following [6] and [4], we have tested the qPMS algorithms on the most difficult case of $q = \frac{n}{2}$. The challenging instances for this case are identified as (13,3), (15,4), (17,5), (19,6), (21,7), (23,8), (25,9), and so on. Since among the existing algorithms, only Algorithm qPMSPrunel deals with the qPMS problem, we compare Algorithm qPMS7 to it. Algorithm qPMS7 can solve any challenging instance up to (21,7). Table 2 shows the results for these challenging instances.

0.6.2 Protein sequences. We have also tested the algorithms on synthetic protein sequences. These sequences have been generated in a manner similar to the generation of DNA strings as explained in Section 0.6.1. The number of the protein strings in each testing dataset n and the length of each protein string m are chosen to be the same: $n = 20$ and $m = 600$. For the case of $q = n = 20$, the (ℓ, d) pairs that correspond to challenging instances

Table 4. Time comparison of different algorithms on the challenging instances of protein sequences for the generalized case - qPMS Problem.

Algorithm	(11,4)	(13,5)	(15,6)	(17,7)	(19,8)
qPMS7	27 s	3 m	18 m	3.8 h	11 h
qPMSPrunel	62 s	16 m	3.7 h	–	–
qPMSPrunel	181 s	113 m	29 h	–	–

The alphabet size $|\Sigma| = 20$, $n = 20$, $m = 600$, and $q = \frac{n}{2} = 10$.
doi:10.1371/journal.pone.0041425.t004

Table 5. Results on real datasets.

Data	Predicted Motifs	Known Motifs	(ℓ, d)
1	CCTCAGCCCC	CCTCAGCCCC	(10,2)
2	ATTTCGTGGCA	ATTTCnnGCCA	(13,2)
3	CCATATTAGGACATCT	CCATATTAGGACATCT	(16,3)
4	TTTCCCATTAAGGAAA	TTTCCnnTnAGGAAA	(16,3)

Data 1: Preproinsulin; Data 2: DHFR; Data 3: c-fos; Data 4: Yeast ECB. Parameter q is set to $q = \frac{n}{2}$.
doi:10.1371/journal.pone.0041425.t005

are (11,5), (13,6), (15,7), (17,8), (19,9), and so on. For the case of $q = \frac{n}{2} = 10$, the challenging instances are (11,4), (13,5), (15,6), (17,7), (19,8), and so on.

Since none of the exact algorithms reported in the literature deals with the qPMS problem for protein sequences, we restrict our comparison to the algorithms qPMSPrunel, qPMSPrunel, and qPMS7. Table 3 and Table 4 show the runtimes of these algorithms for the two cases $q = n = 20$ and $q = \frac{n}{2} = 10$, respectively. As the results show, Algorithm qPMS7 outperforms Algorithms qPMSPrunel and qPMSPrunel on all the cases.

0.7 Experiments on Real Data

0.7.1 Finding real DNA motifs. We tested Algorithm qPMS7 on the real datasets discussed in [27] which is commonly used to measure the accuracy of the existing algorithms (see e.g., [27], [19], and [7]). Each of the datasets is a collection of DNA orthologous sequences from many organisms. These real datasets are substantially different from the simulated data because they contain known transcription regulatory elements, i.e., known motifs. Algorithm qPMS7 was able to identify these known motifs for appropriate values of the parameters ℓ, d and $q = \frac{n}{2}$. We report these motifs in Table 5. However, we should mention that our results are similar to those published in [7], [6] as well as other papers.

0.7.2 Detecting transcription factor-binding sites. We have also tested our algorithms on the biological datasets described in [28]. In this collection there are several datasets. Some strings of each of these datasets contain known transcription factor-binding sites of different lengths and the others do not. Therefore, in order to test these real datasets we rely on the framework for transcription factor-binding sites discovery described in Section 0.5. Recall that this framework needs a qPMS algorithm and a scoring function. Since Algorithm qPMS7 is currently the fastest, we employ it in this framework. Regarding the scoring function, we use the function called “sequence specificity” which is also the one used in [6], which basically is defined to be $f(M) = -\sum_{i=1}^n \log(E(d_H(M, s_i)))$ where $E(d_H(M, s_i))$ is the expected number of times a motif appears in string s_i with up to $d_H(M, s_i)$ mismatches, assuming the i.i.d model. To complete the tests, we need to choose the parameters of the framework $\ell_{\min}$, $\ell_{\max}$, $d_{\max}$, q , and k . We set $\ell_{\min} = 10$, $\ell_{\max} = 21$, $d_{\max} = 7$, $q = \frac{n}{2}$, and $k = 5$. With this setting, we obtain good results like those in [28], [6], and [4] with many transcription factor-binding sites predicted correctly. Table 6 reports some of these correctly predicted binding sites together with the predicted motifs.

Table 6. Results on real datasets for transcription factor-binding sites discovery.

Data	Predicted Motifs	Matched Binding Sites
mus05r	AGAGGAAAAAAAAGGAG	s_1 : GGAAAAACAAGGTAATG
mus07r	CTGCCACCTCTGCAACCC	s_4 : CCCAACACCTGCTGCCTGAGCC
mus11r	AGGGCGGGGGCGGAGCG	s_2 : GCCGCCGGGTGGGGCTGAG
		s_3 : GGGGGGGGGCGGGGC
		s_4 : GTGGGGCGGGGCCTT
		s_9 : GAACAGGAAGTGAGGCGG
hm03r	AAAAGAAAAAATAAACAA	s_1 : TCAAGCAAAAAATAAATAACCTATGCAA
		s_2 : ACAAGCAACAAAATAATCTGTGCAATAT
		s_3 : TATGAGCAACAAAATAACCTGTGCAA
hm08r	CGTGCAGTCCCCCTCAT	s_{10} : TATGGTCATGACGCTGTACAGAGC
hm19r	CCCTCCACCCACCCACAG	s_2 : CACTTTTAGCTCTCCCCCA
hm26r	CCCCCGCCTCCCGTCCC	s_3 : CCCCCTCAGGCTCCCGGG
		s_7 : CTCAGCTGCCCTCCAGGGATTAAG
		s_8 : GCGCCGAGGCGTCCCGAGGCGC

The datasets are from mouse (resp. human) if their names start with “mus” (resp. “hm”).
doi:10.1371/journal.pone.0041425.t006

Discussion

In this paper we have presented Algorithm qPMS7 for the qPMS problem and tested it on DNA as well as protein sequences. Experimental results indicate that Algorithm qPMS7 is faster than other existing algorithms, especially for large values of ℓ and d . Since Algorithm qPMS7 is a search-based algorithm, it uses a small amount of memory. This feature of Algorithm qPMS7 is a major advantage compared to other algorithms such as RISOTTO, Voting, PMS5, and PMS6 which require a large amount of memory when solving instances with large values of ℓ and d . Another advantage of Algorithm qPMS7 over these algorithms is that they cannot deal with the qPMS problem and in particular they only handle the PMS problem.

Algorithm qPMS7 is the result of a combination of an extension of Algorithm qPMSPrune and the core idea of algorithm PMS5.

References

- Laurent D, Philipp B (1997) Searching for regulatory elements in human noncoding sequences. *Current Opinion in Structural Biology* 7: 399–406.
- Rajasekaran S (2009) Computational techniques for motif search. *Frontiers in Bioscience* 14: 5052–5065.
- Bandyopadhyay S, Sahni S, Rajasekaran S (2012) Pms6: A faster algorithm for motif discovery. In: *Proceedings of the second IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCBMS 2012)*, 1–6.
- Dinh H, Rajasekaran S, Kundeti V (2011) Pms5: an efficient exact algorithm for the (ℓ , d)-motif-finding problem. *BMC Bioinformatics* 12.
- Davila J, Balla S, Rajasekaran S (2007) Pampa: An improved branch and bound algorithm for planted (ℓ , d) motif search. Technical report.
- Davila J, Balla S, Rajasekaran S (2007) Fast and practical algorithms for planted (ℓ , d) motif search. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp 544–552.
- Rajasekaran S, Balla S, Huang CH (2005) Exact algorithms for planted motif challenge problems. *Journal of Computational Biology* 12: 1117–1128.
- Chin F, Leung H (2005) Algorithms for discovering long motifs. In: *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC2005)*, Singapore, pp 261–271.
- Pisanti N, Carvalho A, Marsan L, Sagot MF (2006) Risotto: Fast extraction of motifs with mismatches. In: *Proceedings of the 7th Latin American Theoretical Informatics Symposium*, pp 757–768.
- Blanchette M, Schwikowski B, Tompa M (2000) An exact algorithm to identify motifs in orthologous sequences from multiple species. In: *Proceedings of Eighth International Conference on Intelligent Systems for Molecular Biology*, pp 37–45.
- Brazma A, Jonassen I, Vilo J, Ukkonen E (1998) Predicting gene regulatory elements in silico on a genomic scale. *Genome Research* 15: 1202–1215.
- Eskin E, Pevzner P (2002) Finding composite regulatory patterns in dna sequences. *Bioinformatics* 18: 354–363.
- Galas DJ, Eggert M, Waterman MS (1985) Rigorous pattern-recognition methods for dna sequences: Analysis of promoter sequences from *escherichia coli*. *Journal of Molecular Biology* 186: 117–128.
- Sinha S, Tompa M (2000) A statistical method for finding transcription factor binding sites. In: *Proceedings of Eighth International Conference on Intelligent Systems for Molecular Biology*, pp 344–354.
- Staden R (1989) Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences* 5: 293–298.
- Tompa M (1999) An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In: *Proc. Seventh International Conference on Intelligent Systems for Molecular Biology*, pp 262–271.
- Helden J, André B, Collado-Vides J (1998) Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology* 281: 827–842.
- Bailey T, Elkan C (1994) Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In: *Proceedings of Second International Conference on Intelligent Systems for Molecular Biology*, pp 28–36.
- Buhler J, Tompa M (2001) Finding motifs using random projections. In: *Proceedings of Fifth Annual International Conference on Computational Molecular Biology (RECOMB)*, pp 69–76.

In Algorithm qPMSPrune, a “pivot” ℓ -mer is used. In Algorithm qPMS7, we extended this idea by considering two pivot ℓ -mers. This idea can be further generalized by considering more than two pivot ℓ -mers,

In this paper we have also proposed a framework for transcription factor-binding sites discovery. It should be mentioned that our framework together with Algorithm qPMS7 is currently deployed in our online tools at <http://pms.engr.uconn.edu> or at <http://motifsearch.com>. We will be very happy to receive any comments and feedback from users.

Author Contributions

Conceived and designed the experiments: HD SR JD. Performed the experiments: HD. Analyzed the data: HD SR. Contributed reagents/materials/analysis tools: HD SR JD. Wrote the paper: HD SR.

20. Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, et al. (1993) Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science* 262: 208–214.
21. Pevzner P, Sze SH (2000) Combinatorial approaches to finding subtle signals in dna sequences. In: *Proceedings of Eighth International Conference on Intelligent Systems for Molecular Biology*. pp 269–278.
22. Rocke E, Tompa M (1998) An algorithm for finding novel gapped motifs in dna sequences. In: *Proceedings of Second International Conference on Computational Molecular Biology (RECOMB)*. pp 228–233.
23. Keich U, Pevzner P (2002) Finding motifs in the twilight zone. *Bioinformatics* 18: 1374–1381.
24. Price A, Ramabhadran S, Pevzner PA (2003) Finding subtle motifs by branching from sample strings. *Bioinformatics* 1: 1–7.
25. Hertz G, Stormo G (1999) Identifying dna and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics* 15: 563–577.
26. Kuksa P, Pavlovic V (2010) Efficient motif finding algorithms for large-alphabet inputs. *BMC Bioinformatics* 11.
27. Blanchette M (2001) Algorithms for phylogenetic footprinting. In: *Proceedings of Fifth International Conference Computational Biology (RECOMB 2001)*. 49–58.
28. Tompa M, Li N, Bailey T, Church G, Moor BD, et al. (2005) Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology* 23: 137–144.