Xiao *et al. BMC Genomics* 2019, **20**(Suppl 5):424
https://doi.org/10.1186/s12864-019-5701-6

BMC Genomics

RESEARCH Open Access

Novel algorithms for LDD motif search

Peng Xiao[1], Martin Schiller[2] and Sanguthevar Rajasekaran[1*]

From 7th IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS 2017)
Orlando, FL, USA. 19–21 October 2017

Abstract

Background: Motifs are crucial patterns that have numerous applications including the identification of transcription factors and their binding sites, composite regulatory patterns, similarity between families of proteins, etc. Several motif models have been proposed in the literature. The (l, d)-motif model is one of these that has been studied widely. However, this model will sometimes report too many spurious motifs than expected. We interpret a motif as a biologically significant entity that is evolutionarily preserved within some distance. It may be highly improbable that the motif undergoes the same number of changes in each of the species. To address this issue, in this paper, we introduce a new model which is more general than (l, d)-motif model. This model is called (l, d_1, d_2)-motif model (LDDMS) and is NP-hard as well. We present three elegant as well as efficient algorithms to solve the LDDMS problem, i.e., LDDMS1, LDDMS2 and LDDMS3. They are all exact algorithms.

Results: We did both theoretical analyses and empirical tests on these algorithms. Theoretical analyses demonstrate that our algorithms have less computational cost than the pattern driven approach. Empirical results on both simulated datasets and real datasets show that each of the three algorithms has some advantages on some (l, d_1, d_2) instances.

Conclusions: We proposed LDDMS model which is more practically relevant. We also proposed three exact efficient algorithms to solve the problem. Besides, our algorithms can be nicely parallelized. We believe that the idea in this new model can also be extended to other motif search problems such as Edit-distance-based Motif Search (EMS) and Simple Motif Search (SMS).

Keywords: Motif search, Radix sort, Neighborhood tree

Background

Motif search has many applications in solving some crucial biological problems. For example, finding DNA motifs is very important for the determination of open reading frames, identification of gene promoter elements, location of RNA degradation signals, and the identification of alternative splicing sites [1, 2]. For more than 15 years, motif search has stimulated a lot of interest from researchers in different areas.

There are many models of motif search. One popular model that has been studied extensively is the (l, d)-motif model. The corresponding motif search problem is called LDMS. The input for the LDMS problem consists of n input sequences each of length m, and two integers l and d. The task is to find all the strings (also called (l, d)-motifs) of length l each that occur in each of the input sequences within a hamming distance of d. The LDMS problem is known to be NP-hard [3, 4].

Motifs can be thought of as evolutionarily preserved biological information. This information might have undergone different changes in different species. The (l, d)-motif model captures this possibility by requiring that the motif occur within a hamming distance of d in **each** sequence. However, this requirement may be more stringent than needed. When some biological information undergoes changes (e.g., mutations) in various species, the amount of change may not be the same across all the species. Some might have undergone more changes than the others. If we think of d as an upper bound on the amount of change, then it is conceivable (and very likely) that some of the species have undergone less changes. As a result, the (l, d)-motif model is likely to admit many

*Correspondence: rajasek@engr.uconn.edu
[1]Department of Computer Science and Engineering, University of Connecticut, 371 Fairfield Road, 06269 Storrs, CT, USA
Full list of author information is available at the end of the article

spurious strings as motifs. These strings might occur by random chance and get qualified as motifs. Because of this, the LDMS algorithms might take longer time than actually needed. To rectify these shortcomings, in this paper we propose a new model of motifs. This model is called (l, d_1, d_2)-model. The corresponding motif search problem is called the LDDMS problem and defined next.

Definition 1 *The input for the LDDMS problem has n biological sequences each of length m and three integers l, d_1, and d_2. The problem is to find all the strings M of length l that have the following two properties: 1) M should occur in each of the n input strings within a hamming distance of d_1. This requirement is referred to as the (l, d_1)-condition; and 2) M should occur in at least one of the n input strings within a hamming distance of d_2. This requirement is referred to as the (l, d_2)-condition.*

Validity of the (l, d_1, d_2)-motif model

In this section we demonstrate the validity of the (l, d_1, d_2)-motif model with a simple random model for mutations. Assume that the species under consideration have the same origin. Let M be an original motif of length l. Consider a random model where the number of mutations occurring in the species is uniformly distributed in the range $\left[0, \frac{l}{2}\right]$. Let n be the number of species and let the number of mutations that have occurred in these species be $X_1, X_2, \ldots, X_n$, respectively and let $Y = \min\{X_1, X_2, \ldots, X_n\}$ and $Z = \max\{X_1, X_2, \ldots, X_n\}$. It is easy to show that:

$$E[Y] = \sum_{k=1}^{l/2} k \frac{(l/2 - k + 1)^n - (l/2 - k)^n}{(l/2 + 1)^n}$$

$$E[Z] = \frac{1}{(l/2 + 1)^n} \sum_{k=1}^{l/2} [(k + 1)^n - k^n]$$

Thus the difference between Y and Z could be quite large! As an example consider an input of 20 sequences, each of length 600 and let $l = 10$. Assume that the number of mutations d is uniformly random in the range $[0, 5]$. If we set $d_2 = 1$, the probability that there exists at least one DNA sequence such that the motif occurs with a hamming distance of at most d_2 is:

$$p = 1 - \left(\frac{4}{6}\right)^{20} \approx 0.9997$$

When n is larger than 20, this probability will become even higher. Therefore, it is quite reasonable to add the (l, d_2)-condition into the LDMS model.

It is easy to see that if $d_2 \geqslant d_1$, then the (l, d_2)-condition becomes trivial and the LDDMS problem will become the standard LDMS problem. Thus, the LDMS problem is a special case of the LDDMS problem. If $d_2 = 0$, it means that we want to look for a motif that appears exactly in at least one of the input sequences. In the rest of this paper we assume that $d_2 < d_1$.

Related work

(l, d) motif search is also referred to as Planted Motif Search (PMS) problem in some literature. Since (l, d_1, d_2) motif search is closely related to PMS and we will use a PMS solver in one of the LDDMS algorithms, it is necessary to discuss some of the latest PMS algorithms.

In 2012, Yu, et al., proposed PairMotif to solve PMS problems [5]. They reduced the size of candidate motifs and scanned l-mers by selecting pairs of l-mers from different input sequences and then generate the common neighbors. The authors tested PairMotif algorithm on simulated data as well as on five real data sets from [6], which are preproinsulin, DHFR, c-fos, metallothionein and Yeast ECB. It can solve the weak instance (27, 9) within 10 hours. They also showed that PairMotif is more stable in solving PMS problem in longer input sequences [5].

Sometimes, biologists may also be interested in motifs that occur in a fraction of the input strings. The problem of identifying such motifs is known as quorum Planted Motif Search (qPMS). In this case, in addition to l and d and n strings there is an extra input parameter q. The problem is to identify all the (l, d, q)-motifs, that is, all the (l, d)-motifs that occur in at least q% of the input strings. In 2014, Tanaka proposed TraverStringRef in [7]. This algorithm is based on the PMS8 algorithm of Nicolae and Rajasekaran [8]. This is the first algorithm that solved the challenging DNA instance with $(l, d, q) = (25, 10, 20)$ in a reasonable amount of time.

In 2015, Nicolae and Rajasekaran proposed qPMS9 [9]. It can solve challenging instances up to $(25, 10)$ using a single core machine and up to $(30, 13)$ using a 48-core machine. The algorithm is based on PMS8 proposed by the same authors [8], but it added quorum support and also included better pruning techniques to significantly reduce the size of the search space.

In 2016, Xiao, Pal and Rajasekaran proposed qPMS10 [3, 4]. qPMS10 is a randomized algorithm based on the idea of random sampling. It will first utilize any existing PMS solver on a subset of the input. Then the candidate motifs are filtered to get the correct motifs for the original problem. Probability analysis shows that with high probability, the result is correct. Experimental result shows that this algorithm is competitive especially when the dataset is large.

Not only mutations, but also insertions and deletions are important as they may also play critical roles in divergence of biological sequences [10, 11]. In this case, edit distance instead of hamming distance should be considered

[12, 13]. This corresponding problem is modeled as *Edit-distance-based Motif Search (EMS)* problem. There are also some works in the literature on EMS (see e.g., [1, 12–15], and so on).

However, as far as the authors know, no such generalizations of PMS model exist in the published literature. Therefore, we propose LDDMS model and the corresponding algorithms.

Methods

Since the LDMS problem is NP-hard, the LDDMS problem is also NP-hard. All the known exact algorithms for solving the LDMS problem take time that is exponential in some of the underlying parameters. In this paper, we present three efficient algorithms for solving the LDDMS problem. These algorithms are referred to as LDDMS1, LDDMS2 and LDDMS3. Time complexities of these three algorithms are analysed. Experimental results on simulated dataset and real datasets both demonstrate that our algorithms are efficient.

Description of LDDMS algorithms

For any l-mer u we define its d-friendhood as the set of l-mers v whose hamming distance is exactly d from u; define its d-neighborhood as the set of l-mers v whose hamming distance is at most d from u.

For all the LDDMS algorithms, the input is a database S containing n sequences, each of length m, and integers l, d_1 and d_2; the output is all the strings of length l that meet both (l, d_1)-condition and (l, d_2)-condition.

A straight-forward solution is the pattern driven approach. If Σ is the alphabet under concern, there are $|\Sigma|^l$ possible l-mers. For every such l-mer, check if it meets both the (l, d_1)-condition and the (l, d_2)-condition. If so, output this l-mer. Obviously, this algorithm takes too much time.

In addition to pattern driven approaches, we also have sample driven approaches. We could employ the following two step algorithm: 1) First find all the motifs that satisfy the (l, d_1)-condition. This can be done using any of the LDMS algorithms. Let C_1 be the set of these motifs; and 2) For every motif $x \in C_1$, check if x satisfies the (l, d_2)-condition and if so output x. We call this algorithm LDDMS1. Since qPMS9 is currently the most efficient LDMS algorithm [9], we will take advantage of it in LDDMS1 (See Algorithm 1).

Algorithm 1 LDDMS1

```
1: Run any LDMS algorithm to solve the $(l, d_1)$-motif search problem on the input $S$. Let $C_1$ be the set of motifs found;
2: $O_1 = \emptyset$. For every $x \in C_1$, check if $x$ is a $(l, d_2)$-motif in $S$. If so, add $x$ to $O_1$;
3: return $O_1$;
```

Equivalently, we can also find (l, d_2)-motifs in the first step, and then for every such motif check if it satisfies the (l, d_1)-condition. We refer to this algorithm as LDDMS2 (See Algorithm 2).

Note that each valid motif has at least one d_2-neighbor in at least one of the input sequences. We generate $n(m - l + 1)$ l-mers from each of the input sequences. d_2-neighborhood of an l-mer u can be found by constructing the neighborhood tree. With u being the root and the height of the tree being d_2, the level of a node is the hamming distance between u and this node. All the nodes of this tree, including the root and the leaves, will constitute the d_2-neighborhood of u. In Step 3 of LDDMS2, we can employ radix sort and eliminate duplicates. In Step 4 the output O_2 of valid motifs found will be in sorted order.

Algorithm 2 LDDMS2

```
1: Generate all possible $l$-mers from out of each of the $n$ input sequences. Let the collection of these $l$-mers be $L$;
2: For every $u \in L$, generate all the $l$-mers $v$ such that $v$ is a $d_2$-neighborhood of $u$. Let the union of $v$ be $N$;
3: Sort all the elements in $N$, in lexicographic order, and remove the duplicates. Let the sorted and reduced set be $C_2$;
4: $O_2 = \emptyset$. For every $x \in C_2$, check if $x$ is a $(l, d_1)$-motif in $S$. If so, add $x$ to $O_2$;
5: return $O_2$;
```

If d_2 is very small (for example, d_2 = 0 or 1), we can expect LDDMS2 to run faster than LDDMS1. This is because the d_2-neighborhhod for any l-mer will be small. However, when d_2 is large, the neighborhood tree will be large and so will be the number of candidate motifs. Therefore, LDDMS2 takes much more time and memory when d_2 is large. To save time, one idea is to check the candidate motifs concurrently while constructing the neighborhood tree. During the checking process, some pruning conditions can be developed such that once certain conditions hold, a node is not explored deeper. The stronger the pruning condition is, the faster the algorithm will be. Inspired by similar pruning ideas proposed for the LDMS model [16], we develop LDDMS3 (See Algorithm 3).

Definition 2 *Given an l-mer u from Sequence i ($i \in [1, n]$), construct its d_2-neighborhood tree. Let x be any node in this tree, denote $\delta(x, i, q)$ as the smallest hamming distance between x and any l-mer out of Sequence q. Denote $\delta(x, i, I)$ to be the maximum of $\delta(x, i, q)$ where*

Algorithm 3 LDDMS3

```
1: For $i = 1, 2, ...n$, generate all possible $l$-mers from out of the input sequence $S_i$. Let the collection of these $l$-mers be from the sequence $S_i$ be $L_i$, for $1 \le i \le n$;
2: For every $u \in L_i$, construct a $d_2$-neighborhood tree;
3: $O_3 = \emptyset$. Traverse the neighborhood tree and for each node $x$ compute $\delta(x, i, I)$ and based on Theorem 1, add $x$ or $x\prime$ into $O_3$;
4: Sort all the elements in $O_3$, in lexicographic order, and remove the duplicates;
5: return $O_3$;
```

$q = 1, 2, ..., n$ *and* $q \neq i$.

$$\delta(x, i, I) = \max_{q=1, q\neq i}^{n} \delta(x, i, q) = \max_{q=1, q\neq i}^{n} \min_{v \lhd_l s_q} Hd(v, x)$$

If v is an l-mer in the sequence S_q, we denote it as: $v \lhd_l s_q$. Also, $Hd(v, x)$ is the hamming distance between v and x. By computing $\delta(x, i, I)$, we have the following pruning conditions [16].

Theorem 1 *Traverse the d_2-neighborhood tree of u in a depth-first manner and compute $\delta(x, i, I)$ where x is a node in the tree, h is the level of x (root is at level 0);*

1 *If $\delta(x, i, I) \leqslant d_1$, output x;*
2 *If $\delta(x, i, I) - d_1 > d_2 - h$, prune all the descendants from x;*
3 *If $\delta(x, i, I) - d_1 = d_2 - h$, consider only $x\prime$ such that $x\prime$ is a child of x and $\delta(x\prime, i, I) = \delta(x, i, I) - 1$;*
4 *If $\delta(x, i, I) - d_1 = d_2 - h - 1$, consider only $x\prime$ such that $x\prime$ is a child of x and $\delta(x\prime, i, I) \leqslant \delta(x, i, I)$.*

Analysis of LDDMS algorithms

Candidate size and expected number of motifs

In this section, we estimate candidate sizes of LDDMS1 and LDDMS2, i.e., $|C_1|$ and $|C_2|$, and also the expected number of motifs that would be found. Such estimation is useful in computing the time complexities of these two algorithms.

Recall that in the benchmark dataset all the characters are generated from i.i.d. and there are n sequences with length m each. Given an l-mer M, the number of l-mers that have a hamming distance of $\leqslant d_1$ from M is:

$$N(\Sigma, l, d_1) = \sum_{i=0}^{d_1} \binom{l}{i} (|\Sigma| - 1)^i \tag{1}$$

where Σ is the alphabet under concern.

The probability that a randomly chosen l-mer has a hamming distance of at most d_1 from M is:

$$p_1 = \frac{N(\Sigma, l, d_1)}{|\Sigma|^l} \tag{2}$$

The probability that in a sequence of length m, there is at least one string u such that u and M are within a hamming distance of d_1 is:

$$p_2 = 1 - (1 - p_1)^{m-l+1} \tag{3}$$

The probability that a randomly chosen l-mer occurs within a hamming distance of d_1 in each of the n input sequences, each of length m is:

$$p_3 = p_2^n \tag{4}$$

Therefore, the expected number of (l, d_1)-motifs is:

$$|C_1| = |\Sigma|^l p_3 \tag{5}$$

Similarly, the probability that a randomly chosen l-mer has a hamming distance of at most d_2 from M is:

$$p_4 = \frac{\sum_{i=0}^{d_2} \binom{l}{i} (|\Sigma| - 1)^i}{|\Sigma|^l} \tag{6}$$

The probability that in a sequence of length m, there is at least one string u that has a hamming distance of at most d_2 from M is:

$$p_5 = 1 - (1 - p_4)^{m-l+1} \tag{7}$$

Therefore, the expected number of (l, d_2)-motifs is:

$$|C_2| = |\Sigma|^l \left(1 - (1 - p_5)^n\right) \tag{8}$$

In all of the above assertions we have assumed that the l-mers of a sequence are independent. Clearly, this is incorrect. However, such analyses have proven useful in estimating the number of motifs in practice (see e.g., [17]). Along these lines, let us look at the expected number of motifs that will be found, i.e., $|O_1|$ or $|O_2|$. Let M be a random l-mer, A_i be the event that M has a neighbor that is within a hamming distance of d_2 in exactly i of the input sequences. Similarly, let B_j be the event that M has a neighbor that is within a hamming distance of $(d_2, d_1]$ in exactly j of the input sequences. It should be noted here that if M has a neighbor whose hamming distance is at most d_2 in an input sequence, then it automatically will also have a neighbor that is within a hamming distance of d_1 in such sequence since we assume $d_2 < d_1$.

We want to know the probability that events A_i and B_{n-i} both happen, which means in each of the n input sequences, there is an l-mer that is within a hamming distance of d_2 from M and also, in each of the remaining $n - i$ input sequences, there will be an l-mer that is within a hamming distance of $(d_2, d_1]$ from M.

Given an l-mer M, the probability that a random string u of length l has a hamming distance in the range of $(d_2, d_1]$ from M is:

$$p_6 = \frac{\sum_{i=d_2+1}^{d_1} \binom{l}{i} (|\Sigma| - 1)^i}{|\Sigma|^l} \tag{9}$$

In one sequence, there are $m-l+1$ such l-mers. The probability that in such a sequence, there is at least one l-mer that is within a hamming distance of d_1 but no l-mer that is within a hamming distance of d_2 from M is:

$$p_7 = \sum_{k=1}^{m-l+1} \binom{m-l+1}{k} p_6^k (1-p_4-p_6)^{(m-l+1-k)} \quad (10)$$

Therefore, the probability that a random l-mer out of such dataset meets both (l, d_1) and (l, d_2)-condition is:

$$p_8 = \sum_{i=1}^{n} p(A_i \cap B_{n-i}) = \sum_{i=1}^{n} \binom{n}{i} p_5^i p_7^{(n-i)} \quad (11)$$

In conclusion, the expected number of spurious motifs we can find in the LDDMS model is:

$$|O_1| = |O_2| = |O_3| = |\Sigma|^l p_8 \quad (12)$$

Time complexity of the algorithms

Note that all the three algorithms (LDDMS1, LDDMS2, and LDDMS3) can be nicely parallelized. For LDDMS1, there are parallel versions of LDMS solvers, such as PMS9. For every candidate motif, the checking process is independent and can also be parallelized. For LDDMS2 and LDDMS3, we need to generate the neighnorhood tree for $n(m-l+1)$ l-mers out of the input sequences. There are $n(m-l+1)$ independent subproblems and can be assigned to different processors. However, in this paper, we only implement these algorithms sequentially and analyze the time complexity of the sequential versions of these algorithms.

Given a candidate motif of length l, checking if it meets (l, d_1) and (l, d_2)-condition in an input of n sequences, each of length m, will take $O((m-l+1)nl) = O(mnl)$ time. It is easy to see that the brute-force algorithm takes time $O(|\Sigma|^l mnl)$.

For LDDMS1, qPMS9 can be implemented in $O(m^k mnN(\Sigma, l, d_1))$ time. $N(\Sigma, l, d_1)$ has the same definition as in Eq. 1. k is a dynamic variable between 1 and n. We get the following:

Theorem 2 *The time complexity of LDDMS1 algorithm is*

$$T_{LDDMS1} = O(m^k mnN(\Sigma, l, d_1) + |C_1|mnl)$$

where $|C_1|$ is the candidate size of (l, d_1)-motif. An expected number can be obtained from Eq. 5.

For LDDMS2, in Step 1 and Step 2, generating the neighborhoods from all l-mers out of each of the input sequences will take time $O((m-l+1)nN(\Sigma, l, d_2))$. In Step 3, radix sort and removing the duplicates will take time $O((m-l+1)nlN(\Sigma, l, d_2))$. Thus we arrive at:

Theorem 3 *LDDMS2 can be implemented in time*

$$\begin{aligned} T_{LDDMS2} &= O((m-l+1)nlN(\Sigma, l, d_2)) + O(|C_2|mnl) \\ &= O(mnlN(\Sigma, l, d_2) + |C_2|mnl) \end{aligned}$$

where $|C_2|$ is the candidate size of (l, d_2)-motif. An expected number is given in Eq. 8.

The following lemma from [16] is useful in computing the time complexity of LDDMS3.

Lemma 1 *For a node x in the neighborhood tree, $\delta(x, i, I)$ can be updated in $O(mn)$ time.*

Theorem 4 *LDDMS3 can be implemented in time*

$$T_{LDDMS3} = O\left(n^2 m^2 N(\Sigma, l, d_2)\right)$$

Note this is only the worst-case time complexity and d_1 does not appear in this expression. The actual run time could be much less because a lot of branches can be "pruned".

Results and discussion

LDDMS1, LDDMS2 and LDDMS3 are tested on synthetic datasets as well as real datasets. We evaluated our algorithms on a Dell Precisions Workstation T7910 running RHEL 7.0 on two sockets each containing 8 Dual Intel Xeon Processors E5-2667 (8C HT, 20MB Cache, 3.2GHz) and 256GB RAM.

Synthetic datasets

Following the tradition, we employ combinations of (l, d_1) that are challenging [3]. We vary d_2 from 0 to $\lfloor d_1/2 \rfloor$. The challenging instances of $n = 20$, $m = 600$ for DNA sequences and the values of d_2 for carrying out the test are listed in Table 1.

The challenging instances correspond to a small number of spurious motifs. This will make the candidate size in LDDMS1 very small and hence the time spent in Step 2 in LDDMS1 is trivial. To avoid such problems, we slightly change the way we plant the motifs. We will randomly generate two l-mers, M_1 and M_2. The hamming distance of M_1 and M_2 is q. Then we insert M_1 into each of the first $\lceil n/2 \rceil$ input sequences and M_2 into each of the rest $\lfloor n/2 \rfloor$ input sequences. A detailed algorithm for generating the test cases is given in Algorithm 4.

In this way, the common neighbors that are within d_2 hamming distance of M_1 and M_2 are (l, d_1, d_2)-motifs we plant. Generally, when q is small, there will be more common neighbors between M_1 and M_2. Conversely, when q is large, there are fewer common neighbors between M_1 and M_2. By varying q, we can control the output motif size. There is a theory proposed in [8] which proves to be useful here.

Table 1 Challenging instances and value of d_2 for test ($n = 20, m = 600$)

l	d_1	d_2
7	1	0
8	1	0
9	2	[0, 1]
10	2	[0, 1]
11	3	[0, 1]
12	3	[0, 1]
13	4	[0, 2]
14	4	[0, 2]
15	5	[0, 2]
16	5	[0, 2]
17	6	[0, 3]
18	6	[0, 3]
19	7	[0, 3]

Table 2 Running time of LDDMS1

d_2	0	1	2	3
(l, d_1)				
(7, 1)	0.24 s	NA	NA	NA
(8, 1)	0.19 s	NA	NA	NA
(9, 2)	0.44 s	0.42 s	NA	NA
(10, 2)	0.34 s	0.31 s	NA	NA
(11, 3)	1.91 s	1.24 s	NA	NA
(12, 3)	1.83 s	0.81 s	NA	NA
(13, 4)	20.19 s	9.75 s	7.36 s	NA
(14, 4)	23.03 s	8.11 s	5.18 s	NA
(15, 5)	4.75 m	2.51 m	1.51 m	NA
(16, 5)	6.18 m	2.32 m	1.17 m	NA
(17, 6)	1.12 h	29.36 m	20.63 m	12.02 m
(18, 6)	1.57 h	36.55 m	24.51 m	13.44 m
(19, 7)	10.68 h	7.74 h	6.13 h	4.02 h

Algorithm 4 generateTestCases

```
1: Generate $n$ sequences, each of length $m$ from the alphabet $\Sigma$;
2: Randomly generate two $l$-mers, denoted as $M_1$ and $M_2$. The hamming distance between $M_1$ and $M_2$ is $q$;
3: Insert $M_1$ into each of the first $\lceil n/2 \rceil$ input sequences. Insert $M_2$ into each of the remaining $\lfloor n/2 \rfloor$ input sequences. The starting position of $M_i$ $(i = 1, 2)$ is randomly chosen from 1 to $m - l + 1$.
```

Theorem 5 *Two l-mers a and b have a common neighbor M such that $Hd(a, M) \leqslant d_a$ and $Hd(b, M) \leqslant d_b$ if and only if $Hd(a, b) \leqslant d_a + d_b$.*

Applying the above theorem, q has to be at a distance of at most $2d_2$ for M_1 and M_2 to have common neighbors that are within a d_2 hamming distance. When $d_2 = 0$, we set $q = 0$, then there will be at least $N(\Sigma, l, d_2)$ (l, d_1, d_2)-motifs that can be found. When $d_2 \neq 0$, $q = 2d_2$, there will be at least $\binom{2d_2}{d_2}$ (l, d_1, d_2)-motifs that can be found. However, the number of planted (l, d_1)-motifs, i.e., common neighbors that are within a d_1 hamming distance between both M_1 and M_2, is much larger.

We have tested our algorithms on challenging instances of (l, d_1) from $(7, 1)$ upto $(19, 7)$, where d_2 varies from 0 to $\lfloor d_1/2 \rfloor$. Tables 2, 3 and 4 show the running times of LDDMS1, LDDMS2 and LDDMS3 on different (l, d_1, d_2) values. For small instances such as $(l, d_1) = (7, 1)$, $(8, 1)$, $(9, 2)$, $(10, 2)$, LDDMS1 runs faster than LDDMS2 and LDDMS3. This is because qPMS9 is fast and there are only a few (l, d_1)-motifs to check. However, for moderate and relatively large instances, a small value of d_2 will make LDDMS2 run much faster than LDDMS1. For example, for $(l, d_1, d_2) = (17, 6, 1)$, LDDMS1 takes 29.36 minutes while LDDMS2 only takes 9.19 minutes to solve. However, for large values of d_2, LDDMS2 is slow. Compared to LDDMS2, LDDMS3 performs much better for large instances although it will take more time when d_2 is small. For example, it can solve instances which LDDMS2 cannot solve, such as $(l, d_1, d_2) = (18, 6, 3)$, $(19, 7, 3)$.

It is obvious that as (l, d_1) instances become larger, all the LDDMS algorithms will take more time. However, an interesting observation is that for a fixed (l, d_1) instance, increasing the value of d_2 will make LDDMS1 run faster but LDDMS2 and LDDMS3 slower. This is because of the

Table 3 Running time of LDDMS2

d_2	0	1	2	3
(l, d_1)				
(7, 1)	2.76 s	NA	NA	NA
(8, 1)	5.26 s	NA	NA	NA
(9, 2)	5.34 s	1.45 m	NA	NA
(10, 2)	7.86 s	3.47 m	NA	NA
(11, 3)	7.14 s	3.86 m	NA	NA
(12, 3)	9.46 s	5.76 m	NA	NA
(13, 4)	8.66 s	5.77 m	1.68 h	NA
(14, 4)	10.74 s	7.73 m	2.54 h	NA
(15, 5)	10.04 s	7.65 m	2.72 h	NA
(16, 5)	11.25 s	9.25 m	3.57 h	NA
(17, 6)	10.62 s	9.19 m	3.84 h	47.27 h
(18, 6)	12.38 s	11.25 m	4.96 h	-
(19, 7)	12.13 s	11.54 m	5.52 h	-

Table 4 Running time of LDDMS3

d_2	0	1	2	3
(l, d_1)				
(7, 1)	5.59 s	NA	NA	NA
(8, 1)	6.66 s	NA	NA	NA
(9, 2)	8.78 s	8.79 s	NA	NA
(10, 2)	9.32 s	9.17 s	NA	NA
(11, 3)	12.09 s	12.22 s	NA	NA
(12, 3)	11.49 s	11.97 s	NA	NA
(13, 4)	16.09 s	16.44 s	25.55 s	NA
(14, 4)	14.41 s	14.71 s	19.68 s	NA
(15, 5)	20.71 s	21.25 s	35.73 s	NA
(16, 5)	18.35 s	18.55 s	28.36 s	NA
(17, 6)	25.86 s	26.05 s	53.12 s	8.30 m
(18, 6)	24.27 s	23.23 s	40.34 s	6.44 m
(19, 7)	32.01 s	33.66 s	1.08 m	7.55 m

Time is in seconds (s), minutes (m) or hours (h). Cells with 'NA' indicate instances that are not defined. Cells with '-' imply the algorithm did not complete in the stipulated 48 h

way we generate the test cases. If d_2 is very small, then the two l-mers we plant will be almost identical. In this case, we will find a lot of (l, d_1)-motifs in the end of Step 2 in LDDMS1. However, small values of d_2 will make the neighborhood tree small, thus LDDMS2 and LDDMS3 will run faster.

Real datasets

We also used the datesets discussed in [18] to test our algorithms. We chose a group of real datasets. We excluded datasets with only one input sequence because such datasets are not meaningful for our test.

We chose two relatively large number, 18 and 19 for the motif length. Then we re-computed d_1 which will make (l, d_1) challanging instances since each dataset has different number of input sequences and different length for each sequence. However, as we noted before, the challenging instances will make the candidate size in LDDMS1 very small. In this case, we cannot manually plant a motif to avoid such a problem. Therefore, we will increment d_1 by 2. We tested the minimum and maximum number of d_2, i.e., 0 and $\lfloor d_2/2 \rfloor$. Table 5 shows the datasets information and the (l, d_1, d_2) instances we have tested.

Table 6 shows the running time of LDDMS1, LDDMS2 and LDDMS3 on real datasets. On the real dataset, for fixed (l, d_1), changing d_2 does not affect the running time of LDDMS1 very much. This is because for a real dataset, the candidate size, i.e., the number of (l, d_1) motifs is unchanged. This is also true for the number of (l, d_2) motifs for LDDMS2. Moreover, as one can find, for a fixed d_1, increasing l will make LDDMS1 run faster because

Table 5 Real datasets from [18]

Datasets	n	m	l	d	$d_1 = d + 2$	d_2
dm01r	4	1500	18	3	5	0, 2
			19	3	5	0, 2
dm03r	3	2000	18	2	4	0, 2
			19	2	4	0, 2
dm04r	4	2000	18	3	5	0, 2
			19	3	5	0, 2
dm05r	3	2500	18	2	4	0, 2
			19	2	4	0, 2

it will be less challenging. Generally, when d_2 is large, LDDMS2 takes much more time. However, it is hard to say for LDDMS1 and LDDMS3, which one performs better. For example, on real dataset dm05r, when $(l, d_1, d_2) = (18, 4, 2)$, LDDMS3 (4.07 s) overperforms LDDMS1 (10.79 s). However, on the same dataset, when $(l, d_1, d_2) = (19, 4, 2)$, LDDMS1 (2.31 s) overperforms LDDMS3 (4.55 s). The actual running time of these algorithms is highly dependent on the dataset and (l, d_1, d_2) values.

Conclusions

Efficient motif search algorithms are crucial in solving many bioinformatics problems effectively. In this paper, we have presented the (l, d_1, d_2) motif model, a more general model for the motif search problem. We also have proposed LDDMS1, LDDMS2 and LDDMS3, three

Table 6 Running time of LDDMS1, LDDMS2, LDDMS3 on real data

Datasets	l	d_1	d_2	LDDMS1	LDDMS2	LDDMS3
dm01r	18	5	0	2.04 m	4.15 s	3.42 s
			2	2.03 m	1.61 h	6.67 s
	19	5	0	17.46 s	4.43 s	3.33 s
			2	17.35 s	1.92 h	5.40 s
dm03r	18	4	0	14.80 s	4.64 s	2.73 s
			2	14.73 s	1.78 h	3.49 s
	19	4	0	3.30 s	4.86 s	2.83 s
			2	3.29 s	2.07 h	3.22 s
dm04r	18	5	0	4.37 m	7.16 s	5.89 s
			2	4.37 m	2.83 h	11.17 s
	19	5	0	42.32 s	7.74 s	5.80 s
			2	42.48 s	3.35 h	9.03 s
dm05r	18	4	0	10.70 s	7.03 s	4.07 s
			2	10.79 s	2.68 h	4.77 s
	19	4	0	2.32 s	6.90 s	4.21 s
			2	2.31 s	3.15 h	4.55 s

Time is in seconds (s), minutes (m) or hours (h) exact efficient algorithms to solve the LDDMS problem. Theoretical analysis shows that our algorithms are very competitive. Experimental results also reveal that our algorithms perform well in practice.

In future we will focus on solving harder LDDMS instances, including those involving protein strings. We also plan to implement our algorithms in parallel.

Acknowledgements
Not applicable.

Funding
This work has been supported in part by the NSF grants 1447711, 1743418 and 1843025. Publication costs have been funded by these grants as well.

Availability of data and materials
The real DNA sequence data can be downloaded from [18]: http://bio.cs.washington.edu/assessment/download.html

About this supplement
This article has been published as part of *BMC Genomics Volume 20 Supplement 5, 2019: Selected articles from the 7th IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS 2017): genomics*. The full contents of the supplement are available online at https://bmcgenomics.biomedcentral.com/articles/supplements/volume-20-supplement-5.

Authors' contributions
PX, MS and SR conceived the study. SR and PX designed the algorithms. PX implemented the algorithms and carried out the experiments. SR, PX, and MS analyzed the results and wrote the paper. All authors reviewed the manuscript.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]Department of Computer Science and Engineering, University of Connecticut, 371 Fairfield Road, 06269 Storrs, CT, USA. [2]School of Life Sciences, University of Nevada, Las Vegas, NV, USA.

Published: 6 June 2019

References

1. Pal S, Xiao P, Rajasekaran S. Efficient sequential and parallel algorithms for finding edit distance based motifs. BMC Genom. 2016;17(4):465.
2. Xiao P, Rajasekaran S. Efficient exact algorithms for LDD motif search. In: 2017 IEEE 7th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS). IEEE; 2017. p. 1–1.
3. Xiao P, Pal S, Rajasekaran S. qPMS10: A randomized algorithm for efficiently solving quorum planted motif search problem. In: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). IEEE; 2016. p. 670–5.
4. Xiao P, Pal S, Rajasekaran S. Randomised sequential and parallel algorithms for efficient quorum planted motif search. Int J Data Min Bioinforma. 2017;18(2):105–24.
5. Yu Q, Huo H, Zhang Y, Guo H. Pairmotif: a new pattern-driven algorithm for planted (l, d) DNA motif search. PLoS ONE. 2012;7(10):48442.
6. Blanchette M, Schwikowski B, Tompa M. Algorithms for phylogenetic footprinting. J Comput Biol. 2002;9(2):211–23.
7. Tanaka S. Improved exact enumerative algorithms for the planted (l, d)-motif search problem. IEEE/ACM Trans Comput Biol Bioinforma. 2014;11(2):361–74.
8. Nicolae M, Rajasekaran S. Efficient sequential and parallel algorithms for planted motif search. BMC Bioinforma. 2014;15(1):1.
9. Nicolae M, Rajasekaran S. qPMS9: An efficient algorithm for quorum planted motif search. Sci Rep. 2015;5:7813. Nature Publishing Group.
10. Pevzner PA, Sze S-H. Combinatorial approaches to finding subtle signals in DNA sequences. In: ISMB, vol. 8; 2000. p. 269–78.
11. Karlin S, Ost F, Blaisdell BE. Patterns in DNA and Amino Acid Sequences and Their Statistical Significance. In: Waterman MS, editor. Mathematical Methods for DNA Sequences. Boca Raton: CRC Press Inc; 1989.
12. Rocke E, Tompa M. On finding novel gapped motifs in DNA sequences. In: In RECOMB98: Proceedings of the Second Annual International Conference on Computational Molecular Biology. ACM; 1998. p. 228–33.
13. Sagot M-F. Spelling Approximate Repeated or Common Motifs using a Suffix Tree. In: LATIN'98: Theoretical Informatics. Brazil: Springer; 1998. p. 374–90.
14. Pathak S, Rajasekaran S, Nicolae M. EMS1: An Elegant Algorithm for Edit Distance Based Motif Search. Int J Found Comput Sci. 2013;24(04):473–86.
15. Wang X, Miao Y. GAEM: A Hybrid Algorithm Incorporating GA with EM for Planted Edited Motif Finding Problem. Curr Bioinforma. 2014;9(5):463–9.
16. Davila J, Balla S, Rajasekaran S. Fast and practical algorithms for planted (l, d) motif search. IEEE/ACM Trans Comput Biol Bioinforma. 2007;4(4):544–52.
17. Rajasekaran S, Nicolae M. An elegant algorithm for the construction of suffix arrays. J Discret Algorithm. 2014;27:21–8.
18. Tompa M, Li N, Bailey TL, Church GM, De Moor B, Eskin E, Favorov AV, Frith MC, Fu Y, Kent WJ, et al. Assessing computational tools for the discovery of transcription factor binding sites. Nat Biotechnol. 2005;23(1):137–44.