Discovering patterns in biological sequences is a crucial problem. For example, the identification of patterns in DNA sequences has resulted in the determination of open reading frames, identification of gene promoter elements, intron/exon splicing sites, and SH RNAs, location of RNA degradation signals, identification of alternative splicing sites, etc. In protein sequences, patterns have led to domain identification, location of protease cleavage sites, identification of signal peptides, protein interactions, determination of protein degradation elements, identification of protein trafficking elements, discovery of short functional motifs, etc. In this paper we focus on the identification of an important class of patterns, namely, motifs. We study the \((\ell, d)\) motif search problem or Planted Motif Search (PMS). PMS receives as input \(n\) strings and two integers \(\ell\) and \(d\). It returns all sequences \(M\) of length \(\ell\) that occur in each input string, where each occurrence differs from \(M\) in at most \(d\) positions. Another formulation is quorum PMS (qPMS), where the motif appears in at least \(q\%\) of the strings. We introduce qPMS9, a parallel exact qPMS algorithm that offers significant runtime improvements on DNA and protein datasets. qPMS9 solves the challenging DNA \((\ell, d)\)-instances \((28, 12)\) and \((30, 13)\). The source code is available at https://code.google.com/p/qpms9/.
Figure 1 | This pseudocode generates tuples of ℓ-mers that can potentially have common neighbors, for the PMS problem.

Algorithm 1: GenerateTuples(T, k, R)

Input: T = (t_1, t_2, ..., t_n), current tuple of ℓ-mers; k, desired size of the tuple; R, array of n - i rows, where R_j contains all alive ℓ-mers from string s_{i+j};

Result: Generates tuples of size k and passes them to the GenerateNeighborhood function;

begin
    if |T| == k then
        GenerateNeighborhood(T, d);
        return;
    outerLoop: for u ∈ R_i do
        T' := T ∪ {u};
        for j ← 1 to n - i - 1 do
            R_j' = {v ∈ R_{j+1} | common d-neighborhood for T' ∪ {v}};
            if |R_j'| == 0 then
                continue outerLoop;
            minAdd := min_{v∈R_j'} Cd(T' ∪ {v}) - Cd(T');
            aliveLmers := |s_{i+j+1} - |R_j'|;
            sortKey[j] := (minAdd, -aliveLmers);
            sort R' decreasingly by sortKey;
        GenerateTuples(T', k, R');
end

Figure 2 | This pseudocode generates tuples of ℓ-mers that can potentially have common neighbors, for the qPMS problem.

Algorithm 2: QGenerateTuples(qTolerance, T, k, R)

Input: qTolerance, number of strings we can afford to skip; T, current tuple of ℓ-mers; i, last string processed; k, desired size of the tuple; R = (R_1, R_{n-i}), where R_j contains all alive ℓ-mers in s_{i+j};

Result: Generate tuples of size k and pass them on to the GenerateNeighborhood function;

begin
    if |T| == k then
        GenerateNeighborhood(T, d);
        return;
    outerLoop: for u ∈ R_i do
        T' := T ∪ {u};
        incompat := 0;
        for j ← 1 to n - i - 1 do
            R_j' = {v ∈ R_{j+1} | common d-neighborhood for T' ∪ {v}};
            if |R_j'| == 0 then
                if incompat ≥ qTolerance then
                    continue outerLoop;
                incompat + +;
            minAdd := min_{v∈R_j'} Cd(T' ∪ {v}) - Cd(T');
            aliveLmers := |s_{i+j+1} - |R_j'|;
            sortKey[j] := (minAdd, -aliveLmers);
            sort R' decreasingly by sortKey;
            QGenerateTuples(qTolerance - incompat, T', k, R');
        if qTolerance > 0 then
            QGenerateTuples(qTolerance - 1, T, k, R \ R_i);
end
Table 1 | Maximum value of $d$ such that the expected number of spurious motifs in random datasets does not exceed 500, for $\ell$ up to 50 and $q$ between 50% and 100%, on DNA data

| $L$ | $q = 50\%$ | $q = 75\%$ | $q = 100\%$ |
|-----|-------------|-------------|-------------|
| 13  | 3           | 3           | 4           |
| 14  | 3           | 4           | 4           |
| 15  | 4           | 4           | 5           |
| 16  | 4           | 5           | 5           |
| 17  | 5           | 5           | 6           |
| 18  | 6           | 6           | 6           |
| 19  | 6           | 6           | 7           |
| 20  | 6           | 7           | 7           |
| 21  | 6           | 7           | 8           |
| 22  | 7           | 8           | 8           |
| 23  | 7           | 8           | 9           |
| 24  | 8           | 9           | 9           |
| 25  | 8           | 9           | 10          |
| 26  | 9           | 10          | 11          |
| 27  | 9           | 10          | 11          |
| 28  | 9           | 11          | 12          |
| 29  | 10          | 11          | 12          |
| 30  | 11          | 12          | 13          |
| 31  | 11          | 12          | 13          |
| 32  | 12          | 13          | 14          |
| 33  | 12          | 13          | 14          |
| 34  | 13          | 14          | 15          |
| 35  | 14          | 15          | 16          |
| 36  | 14          | 15          | 16          |
| 37  | 14          | 16          | 17          |
| 38  | 15          | 16          | 17          |
| 39  | 15          | 17          | 18          |
| 40  | 16          | 17          | 18          |
| 41  | 16          | 18          | 19          |
| 42  | 17          | 18          | 20          |
| 43  | 17          | 19          | 20          |
| 44  | 18          | 19          | 21          |
| 45  | 18          | 20          | 21          |
| 46  | 19          | 21          | 22          |
| 47  | 19          | 21          | 22          |
| 48  | 20          | 22          | 23          |
| 49  | 20          | 22          | 24          |
| 50  | 21          | 23          | 24          |

Table 2 | Maximum value of $d$ such that the expected number of spurious motifs in random datasets does not exceed 500, for $\ell$ up to 30 and $q$ between 50% and 100%, on protein data

| $L$ | $q = 50\%$ | $q = 75\%$ | $q = 100\%$ |
|-----|-------------|-------------|-------------|
| 9   | 4           | 4           | 5           |
| 10  | 4           | 5           | 5           |
| 11  | 5           | 6           | 6           |
| 12  | 6           | 6           | 7           |
| 13  | 6           | 7           | 8           |
| 14  | 7           | 8           | 8           |
| 15  | 8           | 9           | 9           |
| 16  | 9           | 9           | 10          |
| 17  | 9           | 10          | 11          |
| 18  | 10          | 11          | 11          |
| 19  | 11          | 12          | 12          |
| 20  | 11          | 12          | 13          |
| 21  | 12          | 13          | 14          |
| 22  | 13          | 14          | 15          |
| 23  | 14          | 15          | 15          |
| 24  | 14          | 15          | 15          |
| 25  | 15          | 16          | 17          |
| 26  | 16          | 17          | 18          |
| 27  | 16          | 18          | 19          |
| 28  | 17          | 18          | 19          |
| 29  | 18          | 19          | 20          |
| 30  | 19          | 20          | 21          |
Definition 2. The PMS problem: Given n sequences \( s_1, s_2, \ldots, s_n \) over an alphabet \( \Sigma \), and two integers \( \ell \) and \( d \), identify all \( \ell \)-mers \( M, M \in \Sigma^\ell \), such that \( \forall i, 1 \leq i \leq n, 3j, 1 \leq j \leq \ell \), such that \( dH(M, s_j) = d, \mu \leq \ell \leq s \).

Definition 3. The qPMS problem: same as the PMS problem, however the motif appears in at least \( q \% \) of the strings, instead of all of them. PMS is a special case of qPMS for which \( q = 100 \% \).

Another useful notion is that of a \( d \)-neighborhood. Given a tuple of \( \ell \)-mers \( T = (t_1, t_2, \ldots, t\ell) \), the common \( d \)-neighborhood of \( T \) includes all the \( \ell \)-mers \( r \) such that \( dH(r, t_i) \leq d, 1 \leq i \leq s \).

We now define the consensus \( \ell \)-mer and the consensus total distance for a tuple of \( \ell \)-mers. The consensus \( \ell \)-mer for a tuple of \( \ell \)-mers \( T = (t_1, t_2, \ldots, t\ell) \) is an \( \ell \)-mer \( u \) where \( u[i] \) is the most common character among \( t_1[i], t_2[i], \ldots, t\ell[i] \) for each \( 1 \leq i \leq \ell \). If \( u \) is the consensus \( \ell \)-mer for \( T \) then the consensus total distance of \( T \) is defined as \( Csl(T) = \sum_{i=1}^{\ell} dH(u, t^i) \).

Tupple Generation. In the sample driven part of PMS88, tuples \( T = (t_1, t_2, \ldots, t\ell) \), where \( t_i \) is an \( \ell \)-mer from string \( s_i, 1 \leq i \leq k \), are generated based on the following principles. First, if \( T \) has a common \( d \)-neighborhood, then every subset of \( T \) has a common \( d \)-neighborhood. Second, for a motif to exist, there has to be at least one \( \ell \)-mer in \( u \) in the remaining strings \( s_i = 1, 2, \ldots, k \) such that \( T \cup \{u\} \) has a common \( d \)-neighborhood. We call such \( \ell \)-mers \( u \) "alive" with respect to tuple \( T \). As we add \( \ell \)-mers to \( T \) we update the alive \( \ell \)-mers and reorder the strings in increasing order of the number of alive \( \ell \)-mers. This reordering reduces the running time because it leads to generating fewer tuples overall.

In qPMS9 we change the criteria by which the strings are reordered, as follows. Let \( T \) be the current tuple of \( \ell \)-mers and let \( u \) be an alive \( \ell \)-mer with respect to \( T \). If we add \( u \) to \( T \), then the consensus total distance of \( T \) increases. We compute this additional distance \( Csl(TU) - Csl(T) \). For each of the remaining strings, we compute the minimum additional distance for any alive \( \ell \)-mer in that string. Then we sort the strings decreasingly by the minimum additional distance. Therefore, we give priority to the string with the minimum additional distance, we give priority to the string with fewer alive \( \ell \)-mers. The intuition is that larger additional distance could indicate more "diversity" among the \( \ell \)-mers in the tuple, which means smaller common \( d \)-neighborhoods. The pseudocode for generating tuples is given in Figure 1. We invoke the algorithm as \( GenTuples((i, k), R) \) where the matrix \( R \) contains all the \( \ell \)-mers in all the input strings, grouped as one row per string.

Neighborhood Generation. For every tuple \( T \), obtained as described in the previous section, we generate the common \( d \)-neighbors of the \( \ell \)-mers in the tuple. In qPMS9, the neighbor generation uses the same process as in PMS88. For the sake of completeness, we briefly review the process.

Given a tuple \( T = (t_1, t_2, \ldots, t\ell) \) of \( \ell \)-mers, we want to generate all \( \ell \)-mers \( M \) such that \( dH(t_r, M) \leq d, 1 \leq r \leq \ell \). We traverse the tree of all possible \( \ell \)-mers. A node at depth \( d \) which represents an \( \ell \)-mer, is not explored deeper if certain pruning conditions are met. Necessary and sufficient conditions for 2 and 3 \( \ell \)-mers to have a common neighbor are given in Ref. 7. The same paper gives necessary conditions for more than 3 \( \ell \)-mers to have a common neighbor. The interested reader is referred to the PMS8 paper for a more in depth description of neighborhood generation.

Adding Quorum Support. We extend the algorithm to solve the qPMS problem. In the qPMS problem, when we generate tuples, we may "skip" some of the strings entirely. This translates to the implementation as follows: in the PMS version we successively try every alive \( \ell \)-mer in a given string by adding it to the tuple \( T \) and recursively calling the algorithm for the remaining strings. For the qPMS version we have an additional step where, if the value of \( q \) permits, we skip the current string and try \( \ell \)-mers from the next string. At all times we keep track of how many strings we have skipped. The pseudocode for this algorithm is given in Figure 2. We invoke the algorithm as \( QGenerateTuples(n = Q + 1, 1), 0, R \) where \( Q = \frac{\| \ell \|}{100} \) and \( R \) contains all the \( \ell \)-mers in all the strings.

Parallel Algorithm. In PMS88 the search space is split into \( m = \| s \| - \ell + 1 \) independent subproblems \( P_1, P_2, \ldots, P_m \), where \( P_i \) explores the \( \ell \)-neighborhood of \( \ell \)-mer \( s_i[(i-1)\ell + 1 \ldots \ell] \). In the parallel implementation, processor \( 0 \) acts as both a master and a worker, the other processors are workers. Each worker requests a subproblem from the master, solves it, then repeats until all subproblems have been solved.

Communication between processors is done using the Message Passing Interface (MPI).

In qPMS9, we extend the previous idea to the \( q \) version. We split the problem into subproblems \( P_1, P_2, \ldots, P_1, P_{\ell} \ldots, P_{\ell} \), where \( r = n - Q + 1 \) and \( Q = \frac{\| \ell \|}{100} \). Problem \( P_i \) explores the \( \ell \)-neighborhood of the \( i \)-th \( \ell \)-mer in string \( s_i \) and searches for \( \ell \)-mers \( M \) such that there are \( Q - 1 \) instances of \( M \) in strings \( s_1, ..., s_{\ell} \). Notice that \( Q \) is fixed, therefore subproblems \( P_i \) get progressively easier as \( i \) increases.

Test Data Generation. As mentioned in the introduction, PMS algorithms are typically tested on datasets generated as follows. 20 strings of length 600 each are generated from the i.i.d. We choose an \( \ell \)-mer \( M \) as a motif and plant modified versions of it in \( q\% \) of the \( n \) strings. Each planted instance is modified in \( d \) random positions.

It is useful to estimate how many "spurious" motifs (motifs expected by random chance) will be found in a random sample. For that, we make the following observations. The probability that a random \( \ell \)-mer \( u \) is within distance at most \( d \) from another \( \ell \)-mer \( v \) is

\[
P(d, \Sigma) = \frac{\sum_{i=0}^{d} \binom{|\Sigma| - 1}{i}}{|\Sigma|}
\]

The probability that an \( \ell \)-mer is within distance \( d \) from any of the \( \ell \)-mers in a string \( S \) of length \( m \) is:

Table 4 | Runtimes for protein data when \( q = 100 \% \). The time is given in hours (h), minutes (m) or seconds (s), averaged over 5 datasets. TL means that the program runs for more than 24 h

| \( \ell, d \) | TraverStringRef | PMS8 | qPMS9 |
|---|---|---|---|
| (10,5) | 2.6 m | 42 s | 37 s |
| (11,6) | 1.67 h | 11 m | 6.1 m |
| (13,7) | 58.2 m | 2.6 m | 19 s |
| (14,8) | TL | 1.03 h | 29.6 m |
| (15,8) | 28.5 m | 1.2 m | 1.1 m |
| (17,9) | 16.6 m | 45 s | 43 s |
| (19,10) | 5.9 m | 32 s | 32 s |
| (19,11) | TL | 1.23 h | 36 m |
| (22,12) | 3.73 h | 1.2 m | 1.1 m |
| (24,13) | 1.84 h | 48 s | 47 s |
| (26,14) | 30.7 m | 31 s | 32 s |
| (26,15) | TL | 1.19 h | 12.5 m |

Table 5 | Runtimes for DNA data when \( q = 50 \% \). The time is given in hours (h), minutes (m) or seconds (s), averaged over 5 datasets

| Instance | TraverStringRef | PMS8 | qPMS9 |
|---|---|---|---|
| (20,6) | 3 m | 1.5 m |
| (22,7) | 12.9 m | 6.3 m |
| (23,7) | 2.6 m | 48 s |
| (24,8) | 56 m | 26.3 m |
| (25,8) | 9 m | 3.1 m |
| (26,9) | 4.31 h | 1.55 m |
| (27,9) | 39.9 m | 10.6 m |
| (28,10) | 20.86 h | 5.15 h |
| (29,10) | 2.89 h | 34.5 m |
The probability that an $\ell$-mer is within distance $d$ from at least $q$ out of $n$ strings of length $m$ each is:

$$P(m,\ell,d,\Sigma) = 1 - (1 - p(\ell,d,\Sigma))^n - \ell + 1$$  \hspace{1cm} (2)

where $p(\ell,d,\Sigma)$ is the probability that a random $\ell$-mer is within distance $d$ from a single string.

$$Q(q,n,\ell,d,\Sigma) = \sum_{i=q}^{n} \binom{n}{i} P(m,\ell,d,\Sigma)^i (1 - P(m,\ell,d,\Sigma))^{n-i}$$  \hspace{1cm} (3)

Therefore, the expected number of motifs for a given qPMS instance is: $|\Sigma|^q Q(q,n,m,\ell,\Sigma)$. Based on these formulas, we compute for every $\ell$ the largest value of $d$ such that the number of spurious motifs does not exceed 500. These values are presented in Table 1 for DNA and Table 2 for protein.

### Results

In this section we analyze the running times of PMS87, TraverStringRef11 and qPMS9, on several synthetic DNA and protein datasets.

| Instance | TraverStringRef | qPMS9 |
|----------|-----------------|-------|
| (9,4)    | 11.3 m          | 3.7 m |
| (11,5)   | 14 m            | 4.1 m |
| (12,6)   | 6.22 h          | 57.5 m|
| (13,6)   | 17.4 m          | 4.9 m |
| (14,7)   | 5.09 h          | 41.3 m|
| (15,8)   | TL              | 4.62 h|
| (17,9)   | 1.79 h          | 33.1 m|
| (18,9)   | 2.71 h          | 33.3 m|
| (20,10)  | 2.33 h          | 50.9 m|

Table 6 | Runtimes for protein data when $q = 50\%$. The time is given in hours (h), minutes (m) or seconds (s), averaged over 5 datasets. TL means that the program runs for more than 24 h.

| Instance | TraverStringRef | qPMS9 |
|----------|-----------------|-------|
| (9,4)    | 11.3 m          | 3.7 m |
| (11,5)   | 14 m            | 4.1 m |
| (12,6)   | 6.22 h          | 57.5 m|
| (13,6)   | 17.4 m          | 4.9 m |
| (14,7)   | 5.09 h          | 41.3 m|
| (15,8)   | TL              | 4.62 h|
| (17,9)   | 1.79 h          | 33.1 m|
| (18,9)   | 2.71 h          | 33.3 m|
| (20,10)  | 2.33 h          | 50.9 m|

Results in this section analyze the running times of PMS87, TraverStringRef11 and qPMS9, on several synthetic DNA and protein datasets.
instances. For every instance of the problem we generated 5 datasets as described in the Methods section. For \( q = 100\% \) we compare all three algorithms, for \( q = 50\% \) we compare only the algorithms that solve the quorum PMS problem: TraverStringRef and qPMS9. All programs were executed on the Hornet cluster at the University of Connecticut, which is a high-end, 104-node, 1408-core High Performance Computing cluster. For our experiments we used Intel Xeon X5650 Westmere cores. Most results refer to single core execution, unless specified otherwise.

In table 3 we compare the three algorithms on DNA data when \( q = 100\% \). In table 4 we show a similar comparison on protein data. In table 5 we compare TraverStringRef and qPMS9 on DNA data when \( q = 50\% \). In table 6 we compare TraverStringRef and qPMS9 on protein data when \( q = 50\% \).

### Discussion

We have presented qPMS9, an efficient algorithm for Quorum Planted Motif Search. The algorithm is based on the PMS8 algorithm. qPMS9 includes a new procedure for exploring the search space and adds support for the quorum version of PMS. We compared qPMS9 with two state of the art algorithms and showed that qPMS9 is very competitive. qPMS9 is the first algorithm to solve the challenging DNA instances (28, 12) and (30, 13). qPMS9 can also efficiently solve instances with larger \( \ell \) and \( d \) such as (50, 21) for DNA data or (30, 18) for protein data.

For future work, one of our reviewers kindly pointed out that our approach of filtering \( \ell \)-mers for Hamming Distances could benefit for the work in Ref. 16.

---

**Figure 4** | qPMS9 runtimes on protein datasets for multiple combinations of \( \ell \) and \( d \) where \( q = 100\% \). The runtimes are averages over 5 random datasets. The times are given in hours (h) minutes (m) or seconds (s). Grey cells indicate instances that are expected to have more than 500 motifs by random chance (spurious motifs). Blue cells indicate that the program used 48 cores whereas white cells indicate single core execution. Instances in orange could not be solved efficiently.

---

1. Pevzner, P. A. & Sze, S.-H. Combinatorial approaches to finding subtle signals in dna sequences. In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, La Jolla / San Diego, CA, USA, vol. 8, 269–278 (AAAI Press 2000).
2. Buhler, J. & Tompa, M. Finding motifs using random projections. *J. Comp. Biol.* 9, 225–242 (2002).
3. Eskin, E. & Pevzner, P. A. Finding composite regulatory patterns in dna sequences. *Bioinformatics* 18, 354–363 (2002).
4. Price, A., Ramabadran, S. & Pevzner, P. A. Finding subtle motifs by branching from sample strings. *Bioinformatics* 18, 354–363 (2002).
5. Kevin Lancot, J., Li, M., Ma, B., Wang, S. & Zhang, L. Distinguishing string selection problems. *Inform. Comput.* 185, 41–55 (2003).
6. Davila, J., Balla, S. & Rajasekaran, S. Fast and practical algorithms for planted (\( l, d \)) motif search. *IEEE/ACM Trans. Comput. Biol. Bioinf.* 4, 544–552 (2007).
7. Nicolae, M. & Rajasekaran, S. Efficient sequential and parallel algorithms for planted motif search. *BMC Bioinformatics* 15, 34 (2014).
8. Rajasekaran, S., Balla, S. & Huang, C.-H. Exact algorithms for planted motif problems. *J. Comp. Biol.* 12, 1117–1128 (2005).
9. Rajasekaran, S. & Dinh, H. A speedup technique for (\( l, d \))-motif finding algorithms. *BMC Res Notes* 4, 54 (2011).
10. Yu, Q., Huo, H., Zhang, Y. & Guo, H. Pairmotif: A new pattern-driven algorithm for planted (\( l, d \)) dna motif search. *PLoS ONE* 7, e48442 (2012).
11. Tanaka, S. Improved exact enumerative algorithms for the planted (\( l, d \))-motif search problem. *IEEE/ACM Trans. Comput. Biol. Bioinf.* 11, 361–374 (2014).
12. Dinh, H., Rajasekaran, S. & Kundeti, V. Pms5: an efficient exact algorithm for the $(l, d)$-motif finding problem. *BMC bioinformatics* **12**, 410 (2011).

13. Bandyopadhyay, S., Sahni, S. & Rajasekaran, S. Pms6: A fast algorithm for motif discovery. In *IEEE 2nd International Conference on Computational Advances in Bio and Medical Sciences, ICCABS 2012, Las Vegas, NV, USA, February 23–25, 2012* 1–6 (IEEE, 2012).

14. Dinh, H., Rajasekaran, S. & Davila, J. qpms7: A fast algorithm for finding $(l, d)$-motifs in dna and protein sequences. *PLoS ONE* **7**, e41425 (2012).

15. Roy, I. & Aluru, S. Finding motifs in biological sequences using the micron automata processor. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium, IPDPS 14, Washington, DC, USA* 415–424 (IEEE, 2014).

16. Peterlongo, P., Pisanti, N., Boyer, F., do Lago, A. P. & Sagot, M.-F. Lossless filter for multiple repetitions with hamming distance. *JDA* **6**, 497–509 (2008).

**Author contributions**

M.N. and S.R. designed and analyzed the algorithms. M.N. implemented the algorithms and carried out the empirical experiments. M.N. and S.R. analyzed the empirical results and drafted the manuscript. All authors read and approved the final manuscript.

**Additional information**

**Competing financial interests:** The authors declare no competing financial interests.

**How to cite this article:** Nicolae, M. & Rajasekaran, S. qPMS9: An Efficient Algorithm for Quorum Planted Motif Search. *Sci. Rep.* **5**, 7813; DOI:10.1038/srep07813 (2015).

This work is licensed under a Creative Commons Attribution 4.0 International License. The images or other third party material in this article are included in the article’s Creative Commons license, unless indicated otherwise in the credit line; if the material is not included under the Creative Commons license, users will need to obtain permission from the license holder in order to reproduce the material. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.