Solving Planted Motif Problem Using Modeling Method

S. Desarjau and R. Mukkamala
Computer Science Department, Old Dominion University, Norfolk, Virginia, USA

Abstract - *In this paper we describe a new method for solving the Planted Motif Problem that has applications in computational biology. A number of algorithms to solve this problem have been proposed in the past. The largest problem reported solved in the literature is (21, 8). Using the new method we have solved much larger problems, up to a size of (48, 12). The new method is also much faster, and we compare its performance with the best performances reported in the literature.*

Keywords: Elmers, heuristics, memory-constrained computing, modeling, motif, planted motif problem

1 Introduction

The Planted Motif Problem (PMP) may be abstractly defined as: "Given a set of *n* strings, each of length *L*, over an alphabet Σ, find a string *M* of length $l < L$ over Σ, such that there is at least one *d*-neighbor of *M* in each of the *n* strings, where a *d-neighbor* is a string of length *l* that differs from *M* in at most *d* positions and *M* is the *motif* for the given set of *n* strings."

PMP has applications in molecular biology. Identifying subtle signals in the transcription-factor binding sites of several genes is the primary application of PMP. Finding such regulatory patterns among DNA sequences aids the study of gene regulatory networks [1]. The alphabet of the Planted Motif Problem is usually {*A, C, G, T*}, corresponding to the four nucleotide bases that constitute DNA. In principle, however, the problem can be posed for strings over any finite alphabet.

The Planted Motif Problem can be posed for different values of '*l*' and '*d*'. Larger values of '*l*' and '*d*' constitute larger problems, and typically take more time and/or memory to solve. The pair (*l, d*) is used to express the size of a given problem. Most researchers keep '*n*' and '*L*' fixed at 20 and 600, respectively [2].

Several methods currently exist to solve the PMP. These are: PMS1 [3]; Pattern Branching [4]; WINNOWER [2]; MITRA [5]; Random Projection [6]; Bit-based Multi-core [7]; ExVote [8], Stemming [9], PMSPrune [10], RISOTTO [11], and algorithms for solving the Extended Motif Problem (EMP) [1] [12].

We summarize some of these methods below. All these methods have limitations in the size of the problems that they solve and the running time that they require.

In contrast to these existing algorithms, we approach the motif-finding problem with a method of deriving the motif from clues present in the input strings. If there is a motif of length *l* present for a given set of input strings, the length *l* substrings of the input exhibit certain simple properties. We identify these properties and use them to construct the motif. The method has been used to solve problem sizes much larger than those reported solved in the literature, in times much shorter than the times reported for smaller problems in the literature. The method also holds promise for problems of larger alphabet size than the DNA alphabet size of 4.

The rest of the paper is organized as follows. In section 2, we illustrate the problem and summarize previous work in the area. Section 3 describes 'modeling' and the discovered properties as a set of propositions. Section 4 presents a formal algorithm and analysis of the computational complexity. In Section 5 we provide an overview of the statistical properties of the factors involved in the computation. In section 6, we present the results that show the superiority of the proposed method in terms of ability to solve larger problems with smaller run times. We also state the notable shortcomings of the method. Finally, Section 7 summarizes our contributions and discusses future work.

2 Problem Illustration and Previous work

The Planted Motif Problem has been an area of active research for about the last twenty years, and a number of different approaches have been described.

Before we describe the previous work, let us look at an example to illustrate the problem. Consider a set of 3 DNA sequences, each of length 32. The problem is to find a string (motif) of length 6, for which a 2-neighbor exists in each of the 3 sequences. Here, n=3, L=32, l=6, d=2, and $\sum$={A, C, G, T}. The size of the problem is (6, 2).

0 GTCAGAC**AGATCG**TGTTCTATACGACGACTTC
1 CTATGACCAAGGGATTTCTAACCAC**GGCACT**T
2 ATCAGTCCCAGGGTGTTC**CGCTCG**ACGTGTTC

For this problem, there exists a motif – **GGCTCG**. The sub-sequences of length 6 that are in bold face are *d*-neighbors of the motif. The first substring **AGATCG** differs from the motif at the 1st and 3rd positions. The 2nd substring **GGCACT** differs from the motif at the 4th and 6th positions. The third substring **CGCTCG** differs at only the 1st position. So **GGCTCG** is one of the solutions. There could be more than one motif. But the problem statement calls for finding any one such string.

It is important to note that the motif itself may not literally occur in any of the input sequences. All that is required is that a *d*-neighbor of the motif occurs in each of the input sequences.

We now summarize previous efforts to solve this problem. WINNOWER [2] takes a *graph-based* approach to solve the problem. It finds cliques in a graph constructed by representing each length l substring of the input as a node. Thus the number of nodes will be $n*(L\text{-}l+1)$. WINNOWER is effective up to problem sizes of (18, 5) and requires substantial computational resources (both time and memory).

MITRA [5] is based on a *trie* (or prefix tree) *traversal* algorithm. A *mismatch tree* data structure is used, in which all the possible patterns are segregated into disjoint subsets, with each subset starting with a given prefix. It is effective up to problem sizes of (18, 6), taking 40 minutes and 650 MB of memory.

PatternBranching [4] starts with a random seed string and searches for the length l neighbors of this string in the input. It scores the neighbors with an appropriate scoring function and selects the best scoring neighbor.

The Random Projection Algorithm [6] finds motifs using random projections. For each length l substring of the input, a length k string is constructed as a subsequence of the original substring, sharing k random positions (i.e., it is a random projection). All the length l substrings are hashed using the length k string of any length l substring as the hash value. If a hashed group has at least a threshold number of substrings in it, then it is likely that the motif will have its length k projection equal to the length k projection of this group. The largest problem the method is reported to have solved is (18,6), in about one hour.

The PMS1 algorithm [3] is based on exhaustive enumeration. The Bit-based algorithm [7] is also based on exhaustive enumeration, and is a multicore (i.e. parallel processing) implementation, with modifications to address memory-sharing issues and enhance performance. The largest problem the method is reported to have solved is (21, 8), in 7.8 hours, using 16 CPU cores [6].

3 Proposed Method

Our method is based on a process that we call modeling. If any sequence of length l on the alphabet $\sum$ is an 'l-mer', and if the number of positions at which any two l-mers differ is the 'distance' between them, then given two l-mers l_1 and l_2 that are at a distance of $2d$ from each other, we can construct another l-mer l_m that is at a distance d from both l_1 and l_2 as follows:

(i) Note the points at which l_1 differs from l_2 (there are $2d$ such points)

(ii) To obtain l_m, choose any d points in l_1 out of the $2d$ points of difference with l_2, and replace them with the corresponding letters in l_2.

For example, consider two l-mers l_1= **A**G**AT**C**G** and l_2 = **GGCACT**. They differ in four positions (bold faced). We can obtain l_m by replacing the letters at any two of these four positions in l_1, say the 1st and 3rd positions, by the corresponding letter in l_2, thus forming l_m=GGCTCG. This differs from both $l_1$and l_2 at two positions.

We define the process of finding l_m by replacing d letters in l_1 with the corresponding letters of l_2 as *modeling* – l_1 is modeled on l_2, and l_2 is a model for l_1.

Based on the above definition, we make the following propositions. Note beforehand that for an input having n sequences of length L, there are $L - l + 1$ number of l-mers in each of the n input sequences, and the input sequences are numbered from 0 to $n - 1$.

Proposition 1: If there exists a motif of length l for the given n sequences, then in each sequence, at least one l-mer of length l is a d-neighbor of the motif. In particular, one of the $L - l + 1$ number of l-mers in sequence 0 is a d-neighbor of the motif.

Proposition 2: If there exists a motif of length l for the given n sequences, then the d-neighbor in every sequence from sequences 1 to $n - 1$ is at a distance of at most $2d$ from the d-neighbor in sequence 0.

Proposition 3: If there exists a motif of length l for the given n sequences, then the motif can be found by modeling the d-neighbor in sequence 0**,** on any d-neighbor in sequences 1 to n – 1 that is at a distance e*xactly* $2d$ from it.

The logic of Proposition 3 is described as follows:

If two d-neighbors of the motif, dn_0 and dn_1, are at a distance of exactly $2d$ from each other, then they are identical to each other at exactly $l - 2d$ points. If dn_0 and dn_1 are identical to each other at exactly $l - 2d$ points, the motif consists of the identical $l - 2$d points. This follows necessarily from the definition of 'd-neighbor' (or, what is the same, from the definition of 'motif').

Therefore, given two d-neighbors of the motif that are at a distance of exactly $2d$ from each other, $l - 2$d points of the motif are readily identified. The task is to identify the remaining $2d$ points of the motif.

The key is that these points are supplied by dn_0 and dn_1 themselves. The remaining $2d$ points in the motif are the same $2d$ points at which dn_0 and dn_1 differ. At each of the $2d$ points, the symbol in the motif is identical to the symbol at that point in either dn_0 or dn_1. Further, the motif is identical to dn_0 at d of the $2d$ points, and to dn_1 at the remaining d of the $2d$ points. Again, this follows necessarily from the definition of d-neighbor.

So the task becomes one of constructing the motif by choosing d points from dn_0 out of its $2d$ points of difference with dn_1, and choosing d points from dn_1 out of its $2d$ points of difference with dn_0, and inserting the chosen $2d$ points into the corresponding $2d$ points in the motif.

We use modeling to achieve this effect. We take dn_0 and model it on dn_1 at d points out of the $2d$ points of difference. As there are $\binom{2d}{d}$ ways in which d points can be chosen out of $2d$ points, there are $\binom{2d}{d}$ ways in which dn_0 can be modeled on dn_1. The motif can be found by taking each variant of dn_0 by turns, and testing to see if it is the motif.

As it is not known *a priori* which l-mer in sequence 0 is a d-neighbor of the motif, it is required to choose each l-mer one-by-one for processing. Similarly, as it is not known *a priori* which l-mers in sequences 1 to $n - 1$ are d-neighbors of the motif, all the l-mers in sequences 1 to $n - 1$ that are at a

distance of exactly 2*d* from the chosen *l*-mer in sequence 0, have to be found and considered as models.

When the *d*-neighbor of the motif in sequence 0 comes up for processing, all the *d*-neighbors of the motif in sequences 1 to $n - 1$ that are at a distance of exactly 2*d* from it will be found, during the search for all the *l*-mers that are at a distance of exactly 2*d* from it (along with other *l*-mers that happen to satisfy the property). Thereby sooner or later the *d*-neighbor in sequence 0 will be modeled on a *d*-neighbor of the motif that is at a distance of exactly 2*d* from it, and the motif will be found.

Accordingly, we construct the following method, consisting of twelve steps numbered 1 thru 12, to find the motif:

Step 1: Take the first *l-mer* of length *l* in sequence 0; call this the 'root'.

Step 2: Check whether the root is the motif by finding its distance from all the *l*-mers in sequences 1 to $n - 1$.

Step 3: If the root is within distance *d* of at least one *l*-mer in each of the sequences 1 to $n - 1$, then the root is the required motif. Return the root. Otherwise, continue with the next step.

Step 4: For the root, find all the 2*d*-neighbors in sequences 1 to $n - 1$. Call them the 'candidates'.

Step 5: From the set of candidates, take the first candidate that is at a distance of *exactly* 2*d* from the root. Call it the 'model-candidate'. If no such model-candidate exists, repeat the steps from Step 1, taking as the root the next *l*-mer in sequence 0.

Step 6: Model the root on the model-candidate. There are $\binom{2d}{d}$ possible combinations for modeling the root on the model-candidate. Take the first of the $\binom{2d}{d}$ possible combinations, and model the root according to it.

Step 7: Check the distance of the modeled root from all the candidates (i.e., all the 2*d*-neighbors of the root.)

Step 8: If the modeled root is within distance *d* of at least one candidate from each of the input sequences, it is the required motif. Return the modeled root. Otherwise, continue with the next step.

Step 9: Repeat Step 6 by taking the next of the possible $\binom{2d}{d}$ combinations and repeat Step 7. If all the $\binom{2d}{d}$ combinations are exhausted and the motif is not found, repeat the steps from Step 5, by taking as the model-candidate the next candidate that is at a distance of *exactly* 2*d* from the root.

Step 10: If all the candidates found in Step 4 are exhausted and the motif is not found, repeat the steps from Step 1, taking as the root the next *l*-mer in sequence 0.

Step 11: If all the *l*-mers in sequence 0 are exhausted and the motif is not found, relocate sequence 0 to the bottom of the input, such that it becomes sequence $n - 1$ and all the other sequences are promoted in the order by one step. In particular, sequence 1 becomes the new sequence 0. Then repeat the entire process from Step 1, with the new sequence 0.

Step 12. If $n - 1$ input sequences have been promoted to sequence 0 and the motif is not found, then stop and return an exception.

Explanation of Step 11: If all the *l*-mers in sequence 0 are exhausted and the motif is not found (but assumed to exist), it means that either:

(i) the *d*-neighbor of the motif in sequence 0 is not at a distance of exactly *d* from the motif, or

(ii) no *d*-neighbor is found in sequences 1 to n – 1 that is at a distance of exactly 2*d* from the *d*-neighbor in sequence 0

In either case, the fundamental requirement of the method, given under Proposition 3, is not met. Hence the method starts over with a different input sequence taken as sequence 0.

It should be noted that the occurrences of condition (i) and condition (ii) have a computable probability, which will be dealt with in Section 6.

4 Algorithms and Complexity Analysis

The algorithm that encapsulates the 12 steps is given below in Algorithm 1. We analyze the computational complexity of the algorithm as follows:

The algorithm halts when it finds the first motif. In the worst case, statement 1 is executed *n* times. For each execution of statement 1, statement 2 is executed at most $L - l + 1$ times. Statement 3 requires comparing the current root R_{ij} with all possible *l*-mers in $n - 1$ input strings for determining whether it is the motif. This requires at most $(n - 1)*(L - l + 1)$ *l*-mer comparisons. Each comparison involves at most *l* equality checks. If R_{ij} is not the motif, the control comes to statement 5. From here on, we look for a motif using modeling. In statement 5, the set *C* is constructed. This requires $(n - 1)*(L - l + 1)$ *l*-mer comparisons. Since the root is a string of size *l* over an alphabet of size 4, and a candidate is a 2*d*-neighbor of the root, the probability that any *l*-mer is a candidate is given by the ratio of the total number of 2*d*-neighbors that any *l*-mer can have, to the total number of *l*-mers possible. This ratio is:

$$P_C = \frac{\sum_{k=0}^{2d} 3^k \binom{l}{k}}{4^l} \quad (1)$$

The probable number of candidates in *C* is given by multiplying the probability P_C with the total number of *l*-mers in the field of search, which is $(n - 1)*(L - l + 1)$:

$$|C| = (n - 1) * (L - l + 1) * P_C \quad (2)$$

Among the candidates in *C*, those that are at a distance of exactly 2*d* are in the set C_m. These are the model-candidates. By statement 6, the root is modeled on at most $|C_m|$ model-candidates. As $C_m \subseteq C$, $|C_m| \leq |C|$ and therefore the root is modeled on at most $|C|$ model-candidates. In statement 7, the 2*d* points of difference between R_{ij} and one model-candidate are identified. This involves at most *l* equality checks. In statement 8, there are $\binom{2d}{d}$ possible combinations of *d* points among the 2*d* points of difference. In statement 9, R_{ij} is modeled according to one combination to get R_{ijm}, which takes at most *d* operations. For each R_{ijm}, statement 10 is executed to determine whether it is the motif, which involves at most $|C|$ *l*-mer comparisons. Each comparison involves at most *l* equality checks.

In summary, the upper-bound on the number of computations is given by:

$|N| = \boldsymbol{O}(n * (L - l + 1) * (|C|*(l + \binom{2d}{d}) * (d + |C|*l)) + (n - 1)*(L - l + 1)*2*l))$ (3)

The values of n and L are usually constant (20 and 600 respectively), and as L >> l in all practical PMPs, $(L - l + 1) \cong L$. Omitting the constant factors, we have:

$|N| = \boldsymbol{O} (|C| * (l + \binom{2d}{d}) * (d + |C|*l)))$ (4)

Thus the significant factors affecting the running time are the square of the number of candidates per root $|C|^2$, the number of combinations per candidate ($\binom{2d}{d}$), and l and d.

Algorithm 1 FindMotif

Input: *n, L, l , d*
Output: *M (motif)*

```
1:  for i = 0 to n - 1 do
2:     for j = 0 to L - l  do
3:        check whether  root R_ij (an l-mer in
          sequence i starting at position j) is the motif
4:        if R_ij is not the motif then
5:        generate C, the set of all candidates, of which
             C_m is the subset containing the model-candidates
6:        for each model-candidate c in C_m do
7:           identify the 2d points of difference
                between R_ij and c
8:           for each combination of d points of
                   difference between R_ij and c do
9:              model R_ij on c to get R_ijm
10:          check whether R_ijm is the motif using C
11:          if R_ijm is the motif then
                      output R_ijm as M
                      HALT
             end if
12:          end for
13:       end for
14:    else
15:       output R_ij as M
             HALT
16: end if
17:    end for
18: end for
```

5 Overview of Statistical Properties

We have performed a detailed statistical analysis of various factors involved in the computation. Owing to space constraints, we discuss here the salient statistical properties revealed by the analysis, omitting the details.

As noted in Section 4, a major contribution to the computational workload of the method comes from the square of the number of candidates per root, $|C|^2$. An increase in this factor increases the computational workload. The value of $|C|$ depends on the value of l and d (Equations 1 and 2) such that:

(i) increasing l keeping d fixed decreases $|C|$, and
(ii) increasing d keeping l fixed increases $|C|$.

As the computational workload is proportional to $|C|^2$, it is highly sensitive to the ratio l/d. Increasing d keeping l fixed results in massive increase of workload for every step of increment of d. Our analysis shows that, for a broad range of values of l (from 12 to at least 50), a massive increase of $|C|$ occurs when d is increased from $0.25l$ to $0.25l + 1$, rendering problem sizes in which d is greater than $0.25l$ challenging for this method. Conversely, decreasing d keeping l fixed results in a massive drop in $|C|$ for every step of decrement of d. Problem sizes in which d is lesser than $0.25l$ are solved extremely fast.

Another major contributor to the computational workload, as noted in Section 4, is the number of modeling combinations per model-candidate, given by ($\binom{2d}{d}$). This number has a sharply increasing trend for every step of increment in d. Combined with the property that a massive increase of $|C|$ occurs when d is increased from $0.25l$ to $0.25l + 1$, a steep barrier exists at the boundary between those problem sizes in which $d \leq 0.25l$, and those in which $d > 0.25l$ (for all values of l ranging from 12 to at least 50).

Table I presents the values of $|C|$, $|C|^2$ and ($\binom{2d}{d}$) for a few selected problem sizes at the $d = 0.25l$ boundary. The notable feature is that as the problem sizes increase, $|C|$ (and $|C|^2$) decrease sharply at every step, and ($\binom{2d}{d}$) increases sharply. As the computational load is proportional to $|C|^2$ and ($\binom{2d}{d}$), the opposing trends of $|C|^2$ and ($\binom{2d}{d}$) mean that the trend of the computational load is essentially U-shaped, with a minima occurring in the mid-range of problem sizes. (The opposing trends of $|C|^2$ and ($\binom{2d}{d}$) do not perfectly balance each other as their rates of change are not the same, and the proportions of their contribution to the workload are not the same. Therefore we should not expect a flat trend of the workload.)

TABLE I
NUMBER OF CANDIDATES PER ROOT AND NUMBER OF MODEL COMBINATIONS PER CANDIDATE FOR SELECTED PROBLEM SIZES

l	12	16	20	24	28	32	36	40	44	48
d	3	4	5	6	7	8	9	10	11	12
$\|C\|$	609	302	153	79	41	22	11	6	3	2
$\|C\|^2$	370881	91204	23409	6241	1681	484	121	36	9	4
($\binom{2d}{d}$)	20	70	252	924	3432	12870	48620	184756	705432	2704156

Note: All values of d are equal to $0.25l$.

We now turn to the statistical properties of d-neighbors. For modeling to successfully find the motif, the d-neighbor of the motif in sequence 0 has to be at a distance of exactly d from the motif. Those d-neighbors that are at a distance of less than d from the motif are valid d-neighbors, but do not contain enough information to find the motif. As such, the d-neighbor in sequence 0 may or may not be at a distance of exactly d from the motif. The statistics show that the probability of the d-neighbor of the motif in sequence 0 being at a distance of exactly d from the motif is 90% or better, for all problems sizes in which l is in the range of 12 to 50 and d is $d \leq 0.25l$. (Uniform random distribution of d-neighbors is assumed.)

In the 10% of the cases in which the d-neighbor in sequence 0 is at a distance of less than d from the motif, after processing the entire sequence 0 the motif will not be found and method will enter Step 11. Input sequence 1 will become the new sequence 0. The probability that the d-neighbors of

the motif in the first two input sequences are both at a distance less than *d* from the motif is ~1% (by multiplying the 10% probability of each sequence, as they are mutually independent.) Therefore probability that the *d*-neighbor of the motif in the new sequence 0 is at a distance of exactly *d* from the motif is about 99%, and the method can be expected to enter Step 11 for a second time only in 1% of the cases.

The second condition for modeling to successfully find the motif is that at least one *d*-neighbor in input sequences 1 to n – 1 should be at a distance of exactly 2*d* from the *d*-neighbor in sequence 0. The probability of such a *d*-neighbor existing has been found to depend on the ratio *l* / *d*. If *d* is increased keeping *l* fixed, the probability decreases, and if *l* is increased keeping *d* fixed, the probability increases. If the probability is too low and therefore such a *d*-neighbor does not exist, a different input sequence has to be taken as sequence 0. The *d*-neighbor in the new sequence 0 may be such that there is at least one *d*-neighbor in input sequences 1 to n – 1 that is at a distance of exactly 2*d* from it. That is, Step 11 has to be executed.

For problem sizes that have higher values of *d* relative to *l*, the method enters Step 11 more number of times. The number of times that the method enters Step 11 is called the Swap factor (S), and it can be probabilistically calculated for every problem size, from the statistical properties of *d*-neighbors through the values of *l* and *d*. The problem of swapping, however, has been found to become acute only for PMP sizes of (36, 9) and higher (when *d* is restricted to 0.25*l* or less). Table II shows the calculated values of the Swap factor S for selected problem sizes having *l* in the range of 36 to 50.

TABLE II
SWAP FACTOR S FOR SELECTED PROBLEM SIZES

l = 36					
d	8	9	10	11	12
S	0	1	2	7	25
l = 40					
d	9	10	11	12	13
S	0	1	4	11	38
l = 44					
d	10	11	12	13	14
S	1	2	5	15	51
l = 48					
d	11	12	13	14	15
S	1	3	8	21	67
l = 50					
d	11	12	13	14	15
S	1	2	6	15	44

6 Experimental Results

We implemented the modeling method in a single-threaded C++ program and executed it for 11 selected problem sizes on a system with 2.2GHz Intel Core2 Duo Processor T6600, 800 MHz FSB and 4 GB RAM.

Although the algorithm terminates when the first motif is found, in the implementation we processed all the roots so as to observe the processing time for the entire sequence 0. This is required because the 'correct' root (i.e. the *d*-neighbor of the motif) in sequence 0 can occur anywhere in the sequence from position 0 to position L – *l*, which means the motif may be found at any stage in the processing of sequence 0. The time taken to find the motif is therefore not a meaningful indicator of performance. The meaningful indicator is the time taken to process the entire sequence 0.

Also in the implementation, 20 trials were conducted for each problem size, using each of the 20 input sequences as sequence 0, by turns. The rotation was done to observe the variation in processing time when different input sequences are taken as sequence 0. (This rotation of input sequences is unrelated to Step 11 of the method, by which if the motif is not found after processing sequence 0, another input sequence is used as sequence 0, till all the 20 input sequences are used up. It has the same effect as Step 11, however, and therefore, Step 11 of the method was omitted in the test runs as redundant.)

The running time, has to be subjected to certain considerations. Firstly, because the *d*-neighbor of the motif in sequence 0 is at a distance of exactly *d* from the motif in only 90% of the cases, the extra time taken when the method enters Step 11 in 10% of the cases has to be accounted for. Secondly, when the Swap factor S is ≥ 1, the method enters Step 11 S times, and processes a new sequence 0 each time. Therefore the running time has to be multiplied by S. (Only problem sizes (36, 9) and above are affected by this, however.) Thirdly, the time taken to process sequence 0 is different when a different input sequence is taken as sequence 0. This is because all the roots are different and exactly the same number of candidates will not be found for the roots (see Equations 1 and 2). As the complexity is proportional to $|C|^2$, the running time is sensitive to fluctuations in $|C|$.

For problem size (36, 9), the lowest time among the 20 trials, to process all the roots in sequence 0 (=565 in number), was 21 seconds. The motif was found in 9 seconds by modeling root # 318 on *l*-mer # 185 of sequence 4. (This means that the *d*-neighbor of the motif in sequence 0 was at position 318, and there was a *d*-neighbor of the motif in sequence 4 at position 119, that was its '2d' neighbor.)

The highest time among the 20 trials was 244 seconds. The motif was found in 129 seconds by modeling root # 185 on *l*-mer # 318 of sequence 16.

The average time over the 20 trials, for problem size (36, 9), was 117 seconds. The motif was found on 14 of the 20 trials and not found on 6.

We term the average time over the 20 trials as t_{AVG20}, and deem the indicator of the time taken to find the motif in sequence 0 to be 0.5 * t_{AVG20}. This is the intermediate case, between the two extremes of the 'correct' root occurring at position 0 (in which case it takes ~ 0 time to find the motif), and occurring at position $L - l$ (in which case it takes the full average time of t_{AVG20}).

To account for the extra time taken when the method enters Step 11, in 10% of the cases that the d-neighbor of the motif in sequence 0 is not at a distance of exactly d from the motif, an amortized amount of 10% is added to t_{AVG20}.

The time to process sequence 0, obtained from these two considerations, is:

$t_{CORR} = 0.5 * t_{AVG20} + 0.1 * t_{AVG20} = 0.6 * t_{AVG20}$ (5)

For problem size (36, 9), t_{CORR} is 0.6 * 117) = 71 sec. We now consider the extra time taken on account of the Swap factor S. If S swaps are expected, an amount of time equal to S * t_{AVG20} has to be added to t_{CORR} to get the expected time taken to find the motif. Thereby, the expected time taken to find the motif t_{EXP} is: $t_{EXP} = (S + 0.6) * t_{AVG20}$ (6)

Note that the full t_{AVG20} rather than half has to be considered for swap time, because the method always runs through the entire sequence 0 before making a swap.

For problem size (36, 9), 1 swap is expected (see Table II). Therefore, the expected time taken to find the motif t_{EXP} for problem size (36, 9) comes to (1 + 0.6) * 117 = 187 sec.

Table III shows the values of the expected time taken to solve problem sizes in the range of l = 12 to 50 having d = 0.25l. For each problem size, the amount of time added on account of swaps is indicated, as is the 10% correction amount to account for the 'correct' root not occurring in sequence # 0 of the input 10% of the time.

It can be observed from Table III that the best-case performance in the test runs was for problem size (32, 8), with an expected time of 48 seconds, and the worst-case performance was for problem size (48, 12), with an expected time of 6892 seconds, or about 1.9 hours.

The trend in Table III of the expected running time is more or less flat in the range (12, 3) to (24, 6). In the range (28, 7) to (48, 12), there is a clear U-shaped trend with a minima occurring in the mid-range at (32, 8). In this range, the trend is in line with what was expected for the entire range from the statistical analysis in Section 5 (the value for (50, 12) is irrelevant for the trend, as it is an anomalous problem size in the table.) The other notable feature in Table III is the variation over 20 trials, of the range of time taken to process sequence 0. The ratio of the maximum time taken to the minimum time taken increases from about 1 at (24, 6) to about 28 at (44, 11), and then drops to being about 10 for (48, 12) and 5 for (50, 12). The reason for this trend remains to be investigated.

For problem sizes in which d is less than 25% of l, the method is expected to perform much faster than for problem sizes in which d is exactly 25% of l (see Section 5). This has been observed to be the case in practice, and as a ready indicator of the increase in speed for problem sizes in which d is less than 25% of l, the time taken for problem size (50, 12) is included in Table III. This can be compared with the time taken for problem size (48, 12). Although l is larger in the (50, 12) problem, it is solved in less than a third of the time as (48, 12), because d is slightly less than 25% of l in it. The consequence of a slightly smaller d is a significantly reduced computational workload, and also a smaller swap factor S. (It can be observed from Table II that the swap factor decreases with a decrease in d relative to l.) These factors combine to greatly reduce the time taken to solve the (50, 12) problem relative to the (48, 12) problem. Other problem sizes in which d is < 0.25l have been omitted due to space constraints.

TABLE III
TIME TAKEN BY MODELING METHOD FOR SELECTED PROBLEM SIZES

(1) Problem size .	(2) Time for Seq. 0 Min (a)	Max (b)	t_{AVG20} (c)	(3) t_{CORR} 0.6 x (2c)	(4) Swap factor S	(5) S * t_{AVG20} (4)x(2c)	(6) t_{EXP} . (3)+(5)
(12, 3)	1216	1335	1259	756	0	0	756
(16, 4)	1236	1326	1277	767	0	0	767
(20, 5)	1372	1643	1477	887	0	0	887
(24, 6)	1195	1679	1408	846	0	0	846
(28, 7)	288	519	381	229	0	0	229
(32, 8)	25	150	80	48	0	0	48
(36, 9)	21	244	117	71	1	117	187
(40, 10)	33	522	163	98	1	163	262
(44, 11)	24	666	367	221	2	735	955
(48, 12)	645	6625	1939	1164	3	5818	6982
(50, 12)	389	2126	869	522	2	1738	2260

Note: All times are in seconds.
Min, Max and Average times are from 20 trials.

It should be noted that t_{EXP} reported in Table III is derived from practically observed values, and can vary either way, when working with different input sets generated of the same problem size. A different set of n input sequences would have a different distribution of l-mers, affecting the values of $|C|$ and also possibly the number of actual swaps that happen. However the overall trend over the different problem sizes will be more or less the same.

Further, as with any computer program, t_{EXP} depends heavily on the platform used (including the hardware and the operating system) and also the implementation (for example, using the bitset data structure rather than character or string formats for the input sequences and l-mers results in a speed-up of about 2x, as comparison operations run much faster with the bitset data structure).

Coming to the memory requirements, the method uses very little memory. We have calculated that the worst-case memory requirement is well under 1 MB, which is negligible.

From these facts, it is established that the method is very effective for solving PMPs as large as (48, 12). Thus it solves problems much larger than those reported solved in the literature, in running times much shorter than the times reported for smaller problems in the literature. For comparison, Table IV contains representative samples of the time taken by various other methods as reported in the literature.

7 Summary and Future Work

An efficient method of solving the Planted Motif Problem has been developed that uses a technique called modeling. The method is very fast over a broad range of problem sizes, and

takes up very little memory. Using the method, PMPs having problem sizes up to (48, 12) have been solved, with a single-threaded program executed on a system having one 2.2GHz Intel Core2 Duo Processor T6600, 800 MHz FSB and 4 GB RAM.

The high speed of the method, combined with low memory requirement brings motif-finding problems of the order of (48, 12) within easy reach of ordinary desktop/laptop computers. The program can be run comfortably along with the other applications that are typically found in a desktop environment. (In other words, high-end / dedicated systems are not required.)

In conclusion, we note that modeling is independent of the radix of the alphabet, as it works by one-to-one substitution of characters. The same amount of time is taken to model *l*-mers over an alphabet of size 20, say, as it takes to model *l*-mers over an alphabet of size 4. As the method is not restricted to the A, C, G, T alphabet of the Planted Motif Problem, it can have applications in other areas of pattern-finding, which is to be investigated.

TABLE IV
REPRESENTATIVE SAMPLES OF TIME TAKEN BY VARIOUS OTHER METHODS

A	Algorithm			
(*l*, *d*)	Time	Time	Time	Time
	PROJECTION	Styczynski et al.'s	ExVote	
(10,2)	(161.1s)	(8 min)	(0.1 s)	
(11,2)	(12.5 s)	(< 1 min)	(0.7 s)	
(12,3)	(8.7 min)	(10.5 h)	(9.8 s)	
(13,3)	(46.0 s)	(10 min)	(17.4 s)	
(14,4)	(15.4 min)	(> 3 months)	(197.5 s)	
(15,4)	(129.0 s)	(6 h)	(206.1 s)	
(17,5)	(273.2 s)	(3 weeks)	(27 min)	

Source: An Efficient Algorithm for Extended (*l*, *d*)-Motif Problem With Unknown Number of Binding Sites, by Leung and Chin [1]

B	Algorithm			
(*l*, *d*)	Stemming	MITRA	PMSPrune	RISOTTO
(9,2)	0.95s	0.89s	0.99s	1.64s
(11,3)	8.8s	17.9s	10.4s	24.6s
(13,4)	31s	203s	103s	291s
(15,5)	187s	1835s	858s	2974s
(17,6)	1462s	4012s	7743s	29792s
(19,7)	8397s	n/a	81010s	n/a

Source: Efficient Discovery of Common Patterns in Sequences Over Large Alphabets, by Kuksa And Pavlovic [9]

C	Algorithm			
	BitBased			
(*l*, *d*)	16 CPU	8 CPU	4 CPU	
(11,3)	1s	1s	2s	
(13,4)	2s	2s	4s	
(15,5)	15s	24s	47s	
(17,6)	2.8m	5m	9.2m	
(19,7)	35m	63m	112m	
(21,8)	7.8h	-	-	

Source: An Efficient Multicore Implementation of Planted Motif Problem, by Ranjan et al [7]

Note on Table IV: Problems larger than (21,8) have not been reported solved to the best of our knowledge.

8 References

[1] H.C.M. Leung and F.Y.L. Chin, "An efficient algorithm for the extended (l,d)-motif problem with unknown number of binding sites", *Proc. 5th IEEE Symposium on Bioinformatics and Bioengineering (BIBE'05)*, 2004.

[2] P. Pevzner and S.H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences", *Proc. Eighth International Conference on Intelligent Systems for Molecular Biology*, 2000, pp. 269-278.

[3] S. Rajasekaran, S. Balla, and C.-H. Huang, "Exact algorithms for planted motif challenge problems", *Proc. Third Asia-Pacific Bioinformatics Conference*, Singapore, 2005.

[4] A. Price, S. Ramabhadran, and P. A. Pevzner, "Finding subtle motifs by branching from sample strings", *Bioinformatics*, 1 (1), 2003, pp. 1-7.

[5] E. Eskin and P. Pevzner, "Finding composite regulatory patterns in DNA sequences", *Bioinformatics S1*, 2002, pp. 354-363.

[6] J. Buhler and M. Tompa, "Finding motifs using random projections", *Proc. Fifth Annual International Conference on Computational Molecular Biology (RECOMB)*, April 2001.

[7] N. S. Dasari, R. Desh, and M. Zubair, "An efficient multicore implementation of planted motif problem", *Proc. 2010 International Conference on High Performance Computing & Simulation (HPCS 2010)*, France, 2010, pp. 9-15.

[8] F.Y.L. Chin and H.C.M. Leung, "Voting algorithms for discovering long motifs", in *Proc. 3rd Asia-Pacific Bioinformatics Conference (APBC)*, Singapore, 2005. pp. 261–271.

[9] P.P. Kuksa and V. Pavlovic, "Efficient discovery of common patterns in sequences over large alphabets", in *DIMACS Technical Report*, 2009.

[10] J. Davila, S. Balla, and S. Rajasekaran, "Fast and practical algorithms for planted (l, d) motif search", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4), 2007, pp. 544–552.

[11] N. Pisanti, A. Carvalho, L. Marsan, and M.-F. Sagot, "RISOTTO: Fast extraction of motifs with mismatches", *Proc. Latin American Theoretical Informatics Symposium (LATIN)*, Chile, 2006, pp. 757–768.

[12] M.P. Styczynski, K.L. Jensen, I. Rigoutsos, and G.N. Stephanopoulos, "An extension and novel solution to the (l,d)-motif challenge problem", *Genome Informatics*, 15, 2004, pp 63-71.