SMF: Approximate Algorithm for the Planted (*l, d*) Motif Finding Problem in DNA Sequences

Hasnaa Al-Shaikhli[1,2] **and Elise de Doncker**[1]
[1]Department of Computer Science, Western Michigan University, Kalamazoo, Michigan, USA
[2]Computer Science Department, Al-Nahrain University, Baghdad, Iraq
{hasnaaimadabd.alshaikhli, elise.dedoncker}@wmich.edu

Abstract—*Motif discovery is the problem of finding common substrings within a set of biological strings. Therefore, it can be applied to find Transcription Factors Binding Sites (TFBS) that have common patterns (motifs). The Planted (l, d) Motif Problem (PMP) is a classic version of motif discovery where l is the motif length and d represents the maximum allowed mutation distance. In this paper we present an approximate algorithm, Strong Motif Finder (SMF), to return the k highest ranked (strongest) motifs. We propose a scoring function, Motif Strength Score (MSS), which ranks the motifs discovered in the course of the SMF algorithm. We compare the performance of SMF with the APMotif and MEME algorithms with respect to execution time and prediction accuracy. We use several performance metrics at both the nucleotide and the site level. The algorithms are tested on simulated datasets, and results of SMF are also given for real DNA datasets of orthologous regularity regions from multiple species, without using their related phylogenetic tree.*

Keywords: Approximate motif discovery, motif scoring, algorithm evaluation and comparisons, performance metrics

1. Introduction

The DNA motif discovery problem reveals short conserved sites in genome DNA sequences [1], which is considered a major problem in computer science and molecular biology. Tracking and exploring motifs not just in DNA but in other biological sequences such as RNA and proteins help biologists to understand, discover and learn about the functions of these sequences. Despite the research efforts, this problem is still challenging for computer scientists since the general version is considered NP-hard [2].

Motifs are short, recurring substrings that are presumed to have a biological function. Usually, they indicate sequence specific binding sites for proteins such as Transcription Factors (TF). In other cases, they may be engaged in vital processes at the RNA level, such as ribosome binding, mRNA processing (splicing, editing, polyadenylation), and transcription termination [3]. In more detail, the gene expression process starts when a transcription factor molecule begins to bind at a short substring in the promoter region of the gene. This substring is the Transcription Factor Binding Site (TFBS). A transcription factor can bind to multiple binding sites in the promoter region of different genes to make these genes co-regulating. These binding sites generally have common patterns (motifs), and the goal of the motif discovery problem is to find these motifs [4].

Several versions of the motif finding problem can be found in the literature [5]: Planted (l, d) Motif Problem (PMP), Edited Motif Problem (EdMP), Simple Motif Problem (SMP) and Extended (l, d) Motif Problem (ExMP) [2]. In bioinformatics, the classic version is PMP due to its importance in identifying meaningful patterns in biological sequences [6].

The planted (l, d) motif finding problem was first introduced by Sagot [7]. Pevzner and Sze [8] presented a specific challenge instance for $(l, d) = (15, 4)$. The planted (l, d)-motif search problem was described formally in [1] [4] [5] [9]. As input it takes a set of t sequences $\{S_1, S_2, ..., S_t\}$ of length n over the finite alphabet $\{A, C, G, T\}$, and the length l of the motif to search for with allowed mutation distance d where $0 \leq d < l << n$. The problem aims to uncover a consensus motif M of length l (original motif without mutations); target output contains a list of motifs of length l (and their locations) such that, for each sequence S_i, $1 \leq i \leq t$, there is at least one substring of length l at Hamming distance $\leq d$ from M.

Planted (l, d) motif search algorithms can be categorized into approximate and exact classes. Exact algorithms always return the correct solution (implanted motif), whereas approximate algorithms do not guarantee to find the correct solution [5]. Some approximate algorithms, such as WEEDER [10], VINE [11], Pattern Branching [12] and Random Projection [1], apply greedy or heuristic search techniques. Many algorithms apply statistical techniques such as hidden Markov models [13], Gibbs Sampling [14], and expectation maximization such as in the MEME algorithm [15] [16] [17]. Some algorithms combine expectation maximization with other techniques such as the APMotif algorithm [18], which uses affinity propagation clustering to produce informative motif candidates, then refines these using expectation maximization.

2. Proposed Algorithm

If multiple DNA sequences (from the same species or from different species) share certain characteristics, i.e., have common binding sites, then it is reasonable to assume that they share one or more conserved motifs. Furthermore, if multiple DNA sequences have a common conserved motif, then one or more sequences will likely have a motif instance with at most a small number of mutations.

The proposed algorithm attempts to select a sequence with a binding site without or with minimal mutations, which will lead the algorithm to the discovery of binding sites in other sequences.

2.1 Algorithm Input and Output

Input: The input for the algorithm includes $t \times n$ DNA sequences over the alphabet $\{A, C, G, T\}$ where t is the number of sequences and n is the length of each sequence. Also given are the length l of the motif, the maximum allowed mutation distance d, and the number of consensus motifs to be returned by the algorithm (i.e., the top k).

Output: The algorithm returns up to k consensus motifs sorted by their scores from higher to lower. Each consensus motif M is presented with its score, found binding site(s) (called neighbor(s)) in each sequence, and starting position(s).

2.2 Algorithm Description

The pseudo-code of the algorithm is listed as Algorithm 1. The algorithm starts by initializing parameter r in **step 1**. This value is the allowed distance between each substring of length l that the algorithm is checking and its collected neighbors. Specifying its value is important in narrowing or widening the search space. The r value can range between 1 and $2d$. **Step 2** initializes the list *Tried Sequences*. This list contains the sequences processed by the algorithm that failed to return a solution.

The algorithm picks one of the t sequences randomly as the *Selected Sequence* SS in **step 3**. Possible membership of SS in the *Tried Sequences* list is checked in **step 4**, and SS is added to the list in **step 5** if not already present. The core work of the algorithm is performed through **steps 6-21**, where each substring s_i of length l in SS, $1 \leq i \leq n - l + 1$, is checked in **step 9** to see whether it is found in the other $t-1$ sequences at Hamming distance $\leq r$. If the condition is not satisfied, this means that the substring s_i is not common among these sequences, so it is discarded at **step 17**. If the condition is satisfied, then s_i is considered a strong candidate to be a true motif and kept with its found neighbors in the *Neighbors* list at **step 10**.

The *Neighbors* list contains the substring s_i with all similar substrings (neighbors) found in other sequences within a distance of at most r. The Hamming distance d_H is used as the distance measure, which is the total number of mismatches between two substrings. The total number of substrings in the *Neighbors* list is denoted by N.

The collected neighbors may vary from the substring s_i by the allowed distance r. If the substring s_i is the original motif or a motif with small number of mutations, then the retrieved neighbors are also close to the motif. This leads to a highly scored consensus motif and an accurate prediction of binding site locations. If the substring s_i is a motif with a larger number of mutations, then the neighbors are also farther from the motif. This generates a consensus motif with a lower score and weaker prediction of binding sites. Smaller r values tend to reduce the number of collected neighbors and the running time, but lead the algorithm in a direction where no solution may be found if r is too small. Larger r values allow for more sequences, substrings, and neighbors to participate in solutions. This increases the running time and multiple solutions may arise due to sequences that have motif instances with large mutations.

After collecting neighbors, they are refined in **step 11** by keeping the nearest neighbors in each sequence. For example, if a sequence has four neighbors for substring s_i, two of which are at distance $d_H = 2$ and two at $d_H = 1$, the algorithm will ignore the two more distant neighbors and keep the nearest ones.

The algorithm calculates the *Profile matrix* P for the collected neighbors in **step 12**. P is a $4 \times l$ matrix that represents the frequencies of each letter $\{A, C, G, T\}$ at each location for the *Neighbors* list. A consensus motif M is generated in **step 13** from P by choosing the highest frequency at each letter position. In **step 14**, the consensus motif M is scored using the Motif Strength Score (MSS), as proposed in the next subsection. The algorithm adds M to the *Nominated Motifs* list with its score, collected neighbors and starting locations in **step 15**. This list contains all the consensus motifs that are nominated to be binding sites.

When all $(n - l + 1)$ substrings of SS are considered, the algorithm checks whether the *Nominated Motifs* list is empty in **step 20**. If it is empty, then the current SS did not return a solution and the algorithm selects another sequence in **step 21**, until a solution is reached or no more sequences are available. If it is not empty **(step 22)** then the algorithm has a solution.

Finally, if the algorithm found a solution, then the *Nominated Motifs* list is sorted in **step 27** in descending order of the MSS score values. SMF returns up to k of the top-ranked nominated motifs in a final output list named *Final Motifs* at **step 29**. If all sequences fail to return a solution, then the algorithm is unable to reach a solution. Thus SMF does not guarantee to find a solution. The best scenario occurs when SMF selects a sequence SS where a substring s_i has no mutations at all, which helps the algorithm to collect true neighbors at distance r.

Algorithm 1 Strong Motif Finder Algorithm

```
Input: $DNA, t, n, l, d, k$
Output: Final Motifs
 1: Set bound for Hamming distance ($r$)
 2: Tried Sequences $\leftarrow$ [ ]
 3: Select $SS$ randomly
 4: while $SS$ not in Tried Sequences and not all sequences are exhausted do
 5:     Add $SS$ to Tried Sequences
 6:     for each substring $s_i$ in $SS$, $1 \leq i \leq n-l+1$ do
 7:         Neighbors $\leftarrow$ [ ]
 8:         Add $s_i$ to Neighbors
 9:         if $s_i$ found in the other $t-1$ sequences at $d_H \leq r$ then
10:             Add all found similar substrings to $Neighbors$
11:             Refine Neighbors
12:             Calculate Profile Matrix $P$ for Neighbors
13:             Generate $M$ (consensus motif)
14:             Compute Motif Strength Score $MSS$ of $M$
15:             Add $M$ to the Nominated Motifs
16:         else
17:             Discard $s_i$
18:         end if
19:     end for
20:     if Nominated Motifs is empty then
21:         Select $SS$ randomly
22:     else
23:         break                              ▷ out of while loop
24:     end if
25: end while
26: if Nominated Motifs is not empty then
27:     Sort Nominated Motifs list
28:     Final Motifs $\leftarrow$ Top motifs (up to $k$) of the Nominated Motifs
29:     return Final Motifs
30: else
31:     print No Solution Found
32: end if
```

2.3 Algorithm Scoring Function

The consensus motifs M derived by the algorithm are scored in order to determine their relative strength. Various statistical scoring functions have been proposed in the literature, based on information content, p-value, z-score, sequence specificity, etc. In [19], Tompa et al. performed an assessment of 13 computational tools for the discovery of TFBS, followed by further analysis in [20]. Using three of the most used objective functions it was concluded that none of the available scoring functions were satisfactory to retrieve the true binding sites from the background, and there appears to be a lack of knowledge of binding sites characteristics for the scoring functions to work well in general. Here we will utilize a simple statistical measure, *Motif Strength Score (MSS)*, which measures the consensus motif strength based on the generated *Profile Matrix* P according to

$$MSS = \frac{\sum_{i=1}^{l} P[M[i]][i]}{l \times N} \tag{1}$$

where l is the length of the motif, N is the number of neighbors, and M is the consensus string.

MSS sums the frequencies of the *Profile Matrix* P for the corresponding letters of M, and divides the sum by $l \times N$. This is a modification of the scoring function of Jones and Pevzner [21], which only uses the sum in the numerator. In the SMF algorithm the value of $l \times N$ is the maximum frequency sum if all neighbors are exact (so $MSS = 1$). If the MSS value is near 1, that means that the substrings in the *Neighbors* list have high similarity. As such, strong motifs are motifs that are common through all sequences, but not necessarily with the most occurrences. On the contrary, they may be the common ones with least occurrences (as pointed out by Sagot [7]). By the $l \times N$ scaling, the MSS measure can be used to evaluate motifs of different lengths and different numbers of neighbors.

2.4 Algorithm Time Complexity

Within an iteration of the **while** loop started at line 4, the SMF algorithm (Algorithm 1) consumes most of its time through lines 6-19 (the **for** loop). Each substring from the *Selected Sequence SS* is compared to all $(n-l+1)$ substrings of the other sequences. Searching $O(t)$ sequences for a string s_i requires $O((n-l+1)tl)$ time for line 9, and lines 10-15 take $O(lN)$ time leading to $O((n-l+1)tl) + O(lN)$ where N is the maximum number of neighbors retrieved and its bound is $O((n-l+1)t)$. Thus through the **for** and **while** loops, $O((n-l+1)t\ [(n-l+1)tl + O(lN)])$ time may be needed, which is $O((n-l+1)^2t^2l) + O((n-l+1)tlN) = O((n-l+1)^2t^2l)$.

The sort in line 23 consumes $O(N'logN')$ time, where N' is the number of motifs in the *Nominated Motifs* list. Since each s_i can lead to one consensus motif, this can yield $O(n-l+1)$ nominated motifs per *Selected Sequence SS*. Thus, $N' = O((n-l+1)t)$ over $O(t)$ selected sequences. The final algorithm complexity is $O((n-l+1)^2t^2l) + O(N'logN') = O(n^2t^2l)$.

3. Experimental Results

Experimental results were obtained on a DELL laptop with Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz/2.90 GHz and 12.0 GB RAM. SMF is compared with the recent approximate APMotif algorithm [18] (2015), and with the three distribution choices of the MEME algorithm [15]. The latter are: One Occurrence Per Sequence (OOPS), Zero or One Occurrence Per Sequence (ZOOPS), and Any Number of Repetition (ANR). OOPS assumes that each sequence has exactly one occurrence of a motif, ZOOPS assumes that each sequence has at most one motif, and ANR searches for any number of motif occurrences. Therefore, ANR is considered the most practical choice since it does not take prior knowledge about the data into account. The comparison focuses on algorithm execution time and prediction accuracy.

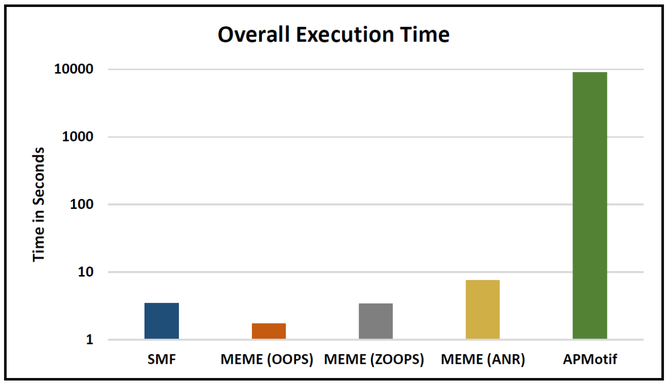

Fig. 1: Overall averaged execution time

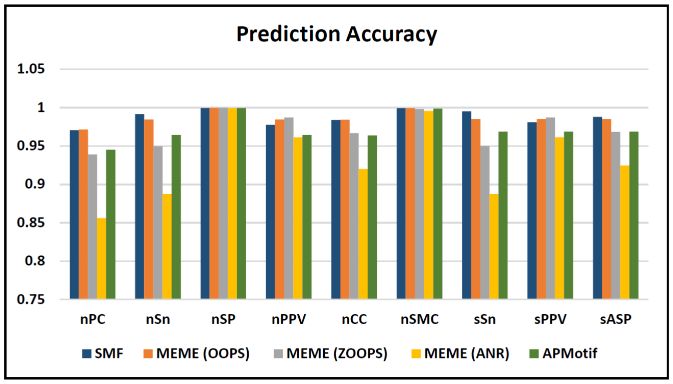

Fig. 2: Overall averaged prediction accuracy

3.1 Experimental Results on Simulated Data

We generate test data for 23 challenging problem instances (l, d), where a problem is called *challenging* according to the classification by Buhler and Tompa [1] into solvable and unsolvable problems, while considering the expected number of spurious motifs (that can occur by chance) in random DNA sequences. We accept the last solvable problem instance as a challenge problem. The 23 problem instances are: *(8,1), (9,1), (10,2), (11,2), (12,3), (13,3), (14,4), (15,4), (16,5), (17,5), (18,6), (19,6), (20,7), (21,7), (22,8), (23,8), (24,9), (25,10), (26,10), (27,11), (28,11), (29,12)* and *(30,12)*. For each of these, we generate a random dataset of 10 sample files over the DNA characters $\{A, C, G, T\}$ with equal probability. A sample file contains $t = 20$ sequences each of length $n = 600$. For each file, a consensus motif M of length l is also generated at random, and twenty instances of M are created by randomly mutating it with variations $\leq d$ at random positions. The instances of M are implanted in the different sequences at random locations. The main goal of the algorithms is to return the consensus motif M from the implanted neighbors with their locations. We average the results over these files for every problem instance. This dataset creation configuration has been used by many authors when testing their algorithms.

3.1.1 Execution Time

Figure 1 depicts the average execution times for the MEME, APMotif and SMF algorithms. The execution time is averaged over all 23 problem instances for MEME and SMF. The APMotif algorithm average time is 2.5 hours over 14 problem instances. It is expected to be higher if the remaining problem instances are timed. The MEME algorithm average execution times for OOPS, ZOOPS, and ANR are 1.7, 3.4, and 7.6 seconds, respectively. The SMF algorithm achieves 3.5 seconds.

3.1.2 Prediction Accuracy

We use nine metrics to measure the prediction accuracy. Two levels of prediction are tested, the nucleotide level and the site level. The metrics definitions and other pertinent definitions related to their computation are given in [19] [22]. In the list below, n and s indicate nucleotide and site respectively. Using T (True), F (False), P (Positive) and N (Negative), the following notations are adopted:

nTP: is the number of nucleotide positions present in both the known and the predicted sites.
nFN: is the number of nucleotide positions in known sites but not in predicted sites.
nFP: is the number of nucleotide positions not in known sites but in predicted sites.
nTN: is the number of nucleotide positions in neither known sites nor predicted sites.
sTP: is the number of known sites overlapped by predicted sites.
sFN: is the number of known sites not overlapped by predicted sites.
sFP: is the number of predicted sites not overlapped by known sites.

We use the following performance metrics to evaluate algorithm performance at the nucleotide and site levels. In the definitions below, x is n (nucleotide level) or s (site level):

Performance Coefficient nPC: Pevzner and Sze [8] define the nucleotide level performance coefficient, which is equivalent to [19],

$$nPC = \frac{nTP}{(nTP + nFN + nFP)} \tag{2}$$

Sensitivity xSn: gives the fraction of known sites or site nucleotides that are predicted,

$$xSn = \frac{xTP}{(xTP + xFN)} \tag{3}$$

Positive Predictive Value xPPV: gives the fraction of predicted sites or site nucleotides that are known,

$$xPPV = \frac{xTP}{(xTP + xFP)} \tag{4}$$

Specificity nSP: gives the fraction of non-site nucleotides that are predicted as non-site nucleotides. However, the number of non-coding nucleotides in DNA sequences is much larger than the number of coding nucleotides. For this reason, TN tends to be much larger than FP,

$$nSP = \frac{nTN}{(nTN + nFP)} \tag{5}$$

Correlation Coefficient nCC: is defined by Burset and Guigó in [22]. The value of nCC ranges from -1 (indicating perfect anti-correlation) to +1 (indicating perfect correlation),

$$nCC = \frac{(nTP)(nTN) - (nFN)(nFP)}{\sqrt{(nTP + nFN)(nTN + nFP)(nTP + nFP)(nTN + nFN)}} \tag{6}$$

Average Site Performance sASP:

$$sASP = \frac{sSn + sPPV}{2} \tag{7}$$

Simple Matching Coefficient nSMC: is the probability of a correct prediction,

$$nSMC = \frac{nTP + nTN}{(nTP + nFN + nFP + nTN)} \tag{8}$$

In Figure 2, the average over all problem instances for each performance metric is given for the three algorithms. The MEME algorithm with the OOPS choice achieves higher prediction accuracy than ZOOPS, and ZOOPS scores higher than ANR. The SMF algorithm achieves performance results at the level of MEME (OOPS), whereas APMotif scores close to MEME (ZOOPS). The APMotif performance tests cover 14 problem instances.

3.2 Experimental Results on Real Data

The SMF algorithm is tested on real datasets that are used by Blanchette and Tompa in [23], where they tested their FootPrinter algorithm. We used seven datasets: *c-fos*, *c-myc*, *growth hormone*, *histone H1*, *insulin*, *interleukin-3*, and *metallothionein*. These are freely available at `http://bio.cs.washington.edu/supplements/FootPrinter`. The website contains nine datasets. We did not consider *c-myc second intron* and *c-fos first intron* because their sequences are of different lengths. While the proposed SMF algorithm can currently only process DNA sequences of the same length, a modification is planned to address this.

The purpose of using the same datasets is to compare the top k motifs returned by SMF with published motifs. The FootPrinter algorithm also returns multiple motifs instead of just one. Other papers that used some of these datasets are [24], [1] and [11]. In [24], the detected motifs are compared with only one motif that was listed in [23]. In [11], only insulin, metallothionein, and c-fos are used.

Figures 3-9 show the common substrings in each dataset when searching for the (l, d) instance. For most datasets we find a large number of common substrings, especially for

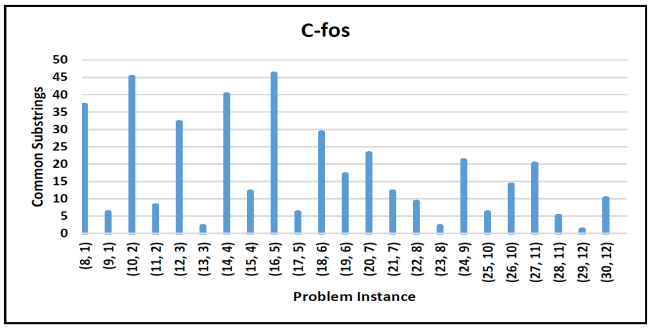

Fig. 3: Number of common substrings for c-fos dataset

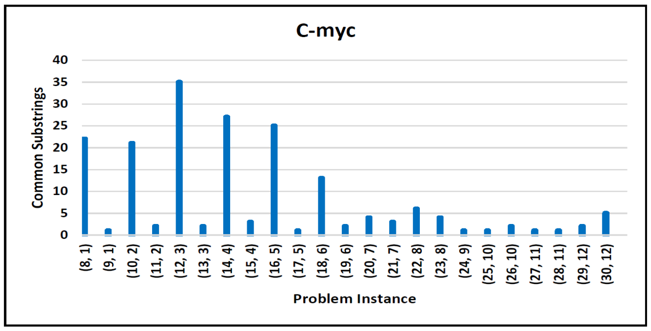

Fig. 4: Number of common substrings for c-myc dataset

interleukin-3, whereas no common substrings are found for the *growth hormone* and *metallothionein* datasets using the default SMF settings of $r = d$. If a dataset does not contain any common DNA pieces, then it is advised to increase the value of r, which will allow more neighbors to be collected and more sequences to contribute, although less powerful motifs may be returned. The results reported for *growth hormone* and *metallothionein* are obtained with $r = d + 1$.

Table 1 lists the final SMF motifs for each dataset, their MSS score and number of neighbors (N) (left column of Table 1). The performance metrics cannot be computed since we do not know the exact locations of the real motifs. SMF discovers multiple motifs for each set, most of which are similar to the published motifs (see the right column of

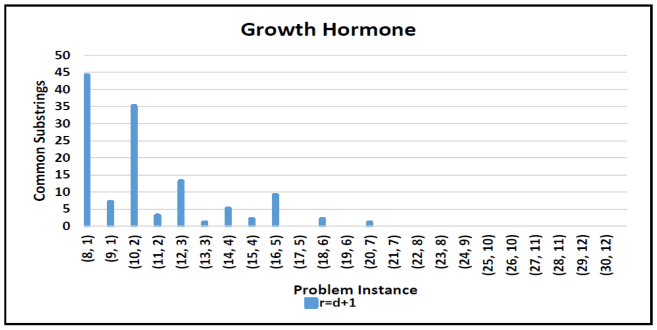

Fig. 5: Number of common substrings for growth hormone dataset

Table 1: Top motifs returned by SMF when applied on real datasets

Dataset$_{t\times n}$	Final SMF Motifs $(l)(N)(MSS)$	Motif $(l)(id$ in [23])
c-fos$_{6\times 702}$	ACAGGATG (8)(6)(1)	CACAGGATGTcc (10)(5)
	CCATATTAGGA (11)(6)(0.97)	
	CCGCGGCCCC (10)(6)(0.95)	
c-myc$_{7\times 1000}$	CCCTCCCC (8)(8)(0.97)	ccaCCCTCCCC (8)(6)
	GTTTATTC (8)(9)(0.93)	aGTTTATTC (8)(1)
	CAGCAAAT (8)(9)(0.92)	
	GCAGCAAA (8)(8)(0.92)	
growth hormone$_{16\times 380}$	TATAAAAA (8)(25)(0.89)$_{(r=d+1)}$	cagggTATAAAAAGggc (9)(7)
	ATGCATTA (8)(31)(0.81)$_{(r=d+1)}$	
histon H1$_{4\times 650}$	ACAAAAGT (8)(4)(1)	gAAACAAAAGTtt (10)(2)
	CAATCACC (8)(4)(0.97)	CAATCACCAC (10)(1)
	GCGGCTCCTCTC (12)(4)(0.9)	
insulin$_{8\times 500}$	CTATAAAG (8)(8)(1)	CTATAAAGcc (8)(3)
	TCAGCCCC (8)(8)(1)	tcagccccaGCCATCTGCC (10)(2)
	GCCATCTGC (9)(8)(1)	tcagccccaGCCATCTGCC (10)(2)
	TTAAGACTCTAAT (13)(8)(0.97)	gttAAGACTCTAAtgacc (10)(1)
interleukin-3$_{6\times 490}$	ACTAGAAA (8)(6)(1)	TTGAGTACTagaaagt (8)(1)
	CTATGGAGGTTCCATGTCAGATAAAG (26)(6)(1)	GTCTGTGGTTTtCTATGGAGGTTCCATGT CAGATAAAG (8)(3)
	TGTGGTTTGCTATGGAGGTTCCATGTCAGA(30)(6)(0.99)	GTCTGTGGTTTtCTATGGAGGTTCCATGT CAGATAAAG (8)(3)
metallothionein$_{26\times 590}$	GTGTGCAG (8)(52)(0.84)$_{(r=d+1)}$	CGTGTGCAggc (8)(3)
	TGTGTGCA (8)(46)(0.84)$_{(r=d+1)}$	CGTGTGCAggc (8)(3)
	TTGCACCC (8)(50)(0.82)$_{(r=d+1)}$	tGCGCCCGG (8)(5)

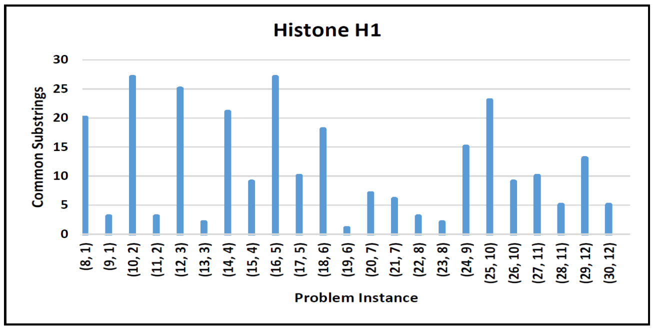

Fig. 6: Number of common substrings for histone H1 dataset

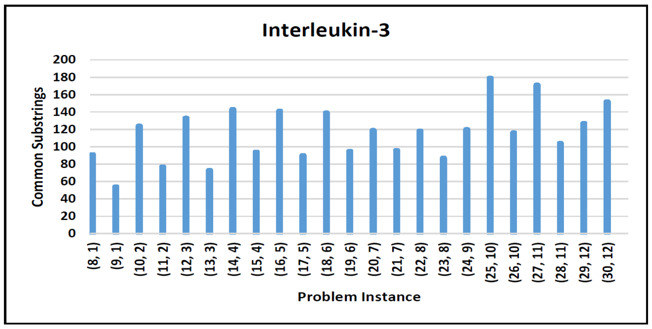

Fig. 8: Number of common substrings for interleukin-3 dataset

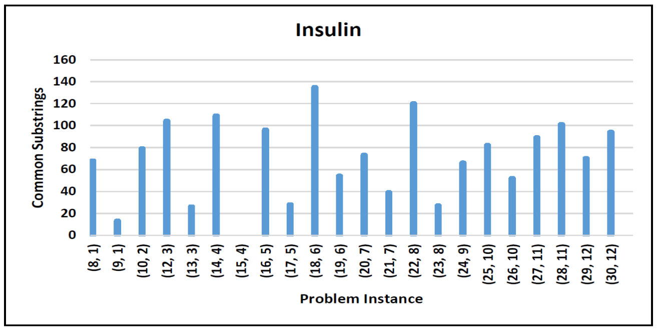

Fig. 7: Number of common substrings for insulin dataset

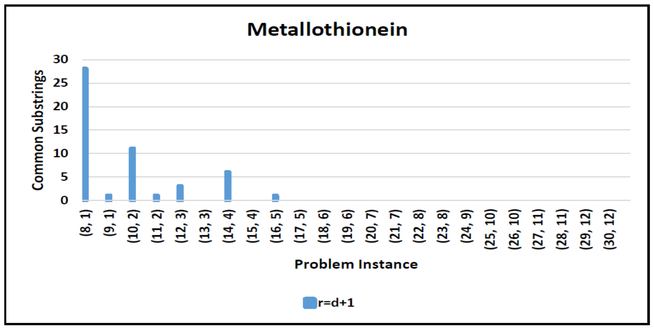

Fig. 9: Number of common substrings for metallothionein dataset

Table 1). The (id) references the motif number in [23]).

4. Conclusions and Future Work

In this paper we propose an algorithm (SMF) to solve the planted (l, d) motif problem. The algorithm implements a strategy based on collecting neighbors of a substring. We further propose a simple statistical measure, Motif Strength Score (MSS) to rank the nominated motifs by the SMF algorithm. The time complexity of SMF is quadratic with respect to the length of the sequences and to the number of sequences. In comparison, the MEME algorithm is quadratic with respect to the number of characters and cubic with respect to the number of sequences [25].

We test the SMF algorithm on simulated data with respect to execution time and prediction accuracy, in comparison with MEME and APMotif. The time comparisons show that APMotif is not practical due to its long execution time even with its high prediction accuracy. SMF is faster than MEME (ANR) and similar in speed to MEME (ZOOPS). The MEME algorithm with choice OOPS is the fastest but is not practical if no prior knowledge is available. Using various performance metrics, the prediction accuracy results show that SMF outperforms APMotif, and performs at the level of the best prediction accuracy of MEME (with OOPS choice), notwithstanding that SMF is not given a-priori information. Results are also reported for SMF using real datasets for multiple species.

With respect to future work, modifications will be studied for adjusting the r parameter in the algorithm, and for specifying a fraction q of the number of sequences where a motif may be found. In addition, whereas the top k motifs returned by the current version of SMF can contain strings that are substrings of each other, further filtering of the results will be addressed.

Acknowledgment. The authors would like to thank Dr. C. Elkan for help with the MEME software installation, and Dr. T. Baily for further information regarding MEME execution times. Many thanks also to Dr. H. Huo for sending the APMotif algorithm code.

References

[1] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," *Journal of Computational Biology*, vol. 9, no. 2, 2002.

[2] T. El Falah, M. Elloumi, and T. Lecroq, "Motif finding algorithms in biological sequences," in *Algorithms in Computational Molecular Biology : Techniques, Approaches, and Applications*, M. Elloumi and A. Y. Zomaya, Eds. Hoboken, NJ: John Wiley & Sons, 2011, ch. 18, pp. 385–396.

[3] P. D'Haeseleer, "What are DNA sequence motifs?" *Nat Biotech*, vol. 24, no. 4, pp. 423–425, 4 2006.

[4] F. Y. Chin and H. C. Leung, "Voting Algorithms for Discovering Long Motifs," *Proceedings of the 3rd Asia-Pacific Bioinformatics Conference*, pp. 261–271, 2005.

[5] S. Rajasekaran, "Algorithms for motif search," in *Handbook of Computational Molecular Biology*, 1st ed., S. Aluru, Ed. Boca Raton: Chapman and Hall/CRC, 2005, ch. 37, pp. 37–1.

[6] Y. Xu, J. Yang, Y. Zhao, and Y. Shang, "An improved voting algorithm for planted (l,d) motif search," *Information Sciences*, vol. 237, pp. 305–312, 7 2013.

[7] M. F. Sagot, "Spelling approximate repeated or common motifs using a suffix tree," *Lecture Notes in Computer Science*, pp. 374–390, 1998.

[8] P. A. Pevzner and S. H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, vol. 8, pp. 269–278, 2000.

[9] H. C. Leung and F. Y. Chin, "Generalized planted (l,d)-motif problem with negative set," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3692 LNBI, 2005, pp. 264–275.

[10] G. Pavesi, G. Mauri, and G. Pesole, "An algorithm for finding signals of unknown length in DNA sequences," *Bioinformatics*, vol. 17, no. 1, pp. 207–214, 2001. [Online]. Available: http://www.cs.duke.edu/courses/fall07/cps296.3/pavesi.2001.pdf

[11] Chao-Wen Huang, Wun-Shiun Lee, and Sun-Yuan Hsieh, "An Improved Heuristic Algorithm for Finding Motif Signals in DNA Sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 4, pp. 959–975, 7 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5582077/

[12] A. Price, S. Ramabhadran, and P. A. Pevzner, "Finding subtle motifs by branching from sample strings," *Bioinformatics*, vol. 19, no. 2, pp. 149–155, 2003. [Online]. Available: http://www-cse.ucsd.edu/groups/

[13] M. M. Yin and J. T. L. Wang, "Effective hidden Markov models for detecting splicing junction sites in DNA sequences," in *Information Sciences*, vol. 139, no. 1-2, 2001, pp. 139–163.

[14] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment," *Science, New Series*, vol. 262, no. 5131, pp. 208–214, 1993.

[15] T. L. Bailey and C. Elkan, "Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Bipolymers," *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pp. 28–36, 1994.

[16] ——, "Unsupervised Learning of Multiple Motifs in Biopolymers Using Expectation Maximization," *Machine Learning*, vol. 21, no. 1/2, pp. 51–80, 1995.

[17] C. E. Lawrence and A. A. Reilly, "An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences," *Proteins: Structure, Function, and Bioinformatics*, vol. 7, no. 1, pp. 41–51, 1990.

[18] C. Sun, H. Huo, Q. Yu, H. Guo, and Z. Sun, "An Affinity Propagation-Based DNA Motif Discovery Algorithm," 2015.

[19] M. Tompa *et al.*, "Assessing computational tools for the discovery of transcription factor binding sites," *Nat.Biotechnol.*, vol. 23, no. 1, pp. 137–144, 2005.

[20] N. Li and M. Tompa, "Analysis of computational approaches for motif discovery." *Algorithms for molecular biology : AMB*, vol. 1, p. 8, 2006.

[21] N. C. Jones and P. A. Pevzner, "Exhaustive search," in *An Introduction to Bioinformatics Algorithms*. Cambridge, Massachusetts: Massachusetts Institute of Technology, 2004, pp. 83–123.

[22] M. Burset and R. Guigó, "Evaluation of Gene Structure Prediction Programs," *Genomics*, vol. 34, no. 3, pp. 353–367, 6 1996.

[23] M. Blanchette and M. Tompa, "Discovery of Regulatory Elements by a Computational Method for Phylogenetic Footprinting," *Genome Research*, vol. 12, no. 5, pp. 739–748, 5 2002.

[24] M. M. Abbas, Q. M. Malluhi, and P. Balakrishnan, "Scalable multi-core implementation for motif finding problem," in *Proceedings - IEEE 13th International Symposium on Parallel and Distributed Computing, ISPDC 2014*. IEEE, 6 2014, pp. 178–183.

[25] "The MEME Suite," Running time on large inputs. [Online]. Available: http://web.mit.edu/meme_v4.11.4/share/doc/meme.html