

Space And Time Efficient Algorithms For Planted Motif Search

Jaime Davila, Sudha Balla and Sanguthevar Rajasekaran

CSE Department at University of Connecticut, Storrs
{jdavila, ballasudha, rajasek}@engr.uconn.edu

Abstract. We consider the (l, d) Planted Motif Search Problem, a problem that arises from the need to find transcription factor-binding sites in genomic information. We propose the algorithms PMSi and PMSP which are based on ideas considered in PMS1 [6]. These algorithms are exact, make use of less space than the known exact algorithms such as PMS and are able to tackle instances with large values of d . In particular algorithm PMSP is able to solve the challenge instance $(17, 6)$, which has not reported solved before in the literature.

1 Introduction

The Planted Motif Search Problem arises from the need to find *transcription factor-binding sites* in genomic information and has been studied extensively in the biocomputing literature –see [7] for a literature survey.

The problem can be defined in the following formal way.

Definition 1. Given a string s with $|s| = m$ and a string x with $|x| = l$ with $l < m$:

1. We say $x \triangleleft_l s$ if x is a subsequence of s . Equivalently we say that x is an l -mer of s .

Definition 2. Given a set of strings $\{s_i\}_{i=1}^n$ over an alphabet Σ , with $|s_i| = m$ and l, d with $0 \leq d < l < m$ we define the (l, d) motif search problem as that of finding a string x with $|x| = l$ such that s_i has an l -mer x_i with

$$d_H(x, x_i) = d \text{ for } i = 1, \dots, n.$$

We will call x a motif.

Numerous algorithms have been implemented in order to solve instances of this problem. Among them we have Random Projection [1], MITRA [3], Winner [4], Pattern Branching [5], PMS1 [6] and Voting [2].

Out of these algorithms, the ones that are exact are MITRA, PMS and Voting. The last two algorithms are able to work in reasonable time for practical instances where $m = 600$ and $n = 20$. In [1] the notion of a *challenging instance* was defined, as one where if the strings are selected at random, the expected

number of (l, d) motifs is greater than 1. We have that $(9, 2)$, $(11, 3)$, $(13, 4)$, $(15, 5)$ and $(17, 6)$ are challenging instances.

PMS1 is able to solve $(9, 2)$, $(11, 3)$ and $(13, 4)$ in less than a couple of minutes, and Voting is able to solve these instances and $(15, 5)$ –the last one in 22 min–. However as d becomes larger the higher memory requirements get.

In this paper we propose the algorithms PMSi and PMSP that build upon ideas similar to the ones in PMS1, but are able to achieve better time results and less use of memory for practical as well as challenging instances. In particular we notice that PMSP is able to handle challenging instances like $(l, d) = (15, 5)$ and $(17, 6)$. No other exact algorithm has been reported to work on $(17, 6)$.

In section 2 we describe the algorithm PMS1 and our improvements PMSi and PMSP and prove some complexity bounds on them.

In section 3 we describe some of the experimental results we have obtained, and compare the results with the already existing exact algorithms.

2 Improved algorithms based on PMS1

PMS1 is a simple exact algorithm introduced in [6] which works as follows. It considers each l -mer in each input sequence and for each such l -mer q it constructs a list of neighbors (i.e., l -mers) that are at a distance of d from q . Neighbor lists of the input sequences are then intersected using radix sort to identify the planted motif. This algorithm works well in practice for values of $d \leq 3$ however as d increases the memory requirement tends to be large.

We propose two algorithms PMSi and PMSP that improve on the memory requirements of PMS1 at the cost of possibly more computation time. However in practical instances we show how they perform better than PMS1 and are able to tackle instances which could not be solved by PMS1.

2.1 PMS1

Before we describe PMS1 [6] we consider the following definitions.

Definition 3. For any string x , with $|x| = l$, let $B(x, d) := \{y : |y| = l \text{ and } d_H(y, x) = d\}$.

Definition 4. Given s , with $|s| = m$ and $0 \leq d < l \leq m$ let $L_s := \bigcup_{x \triangleleft_l s} B(x, d)$.

PMS1 works by doing the following simple steps.

1. Build L_i for $i = 1, \dots, n$.
2. The set of (l, d) motifs will be $M := \bigcap_{i=1}^n L_i$.

In order to do the intersections of step 2 in an efficient way, we keep the sets L_i sorted in lexicographical order.

Theorem 1. PMS1 can be implemented in $O(nm \binom{l}{d} |\Sigma|^{d \frac{l}{w}})$ time and $O(m \binom{l}{d} |\Sigma|^{d \frac{l}{w}})$ space, where w is the word length of the computer.

2.2 PMSi

PMSi works by generating $L_{s_{2i-1}} \cap L_{s_{2i}}$ for $i = 1, \dots, \frac{n}{2}$ and then taking the intersection of these lists. The advantage of such an approach is that $L_{s_{2i-1}} \cap L_{s_{2i}} \subset L_{s_{2i}}$, so it uses less memory than PMS1. In practice, the size of these sets can be substantially less than the size of $L_{s_{2i}}$.

In a more formal way we have the following.

Definition 5. Let $1 \leq i \leq \lceil \frac{n}{2} \rceil$ we define $\bar{L}_i := L_{s_{2i-1}} \cap L_{s_{2i}}$.

Algorithm PMSi

1. Let $M := \bar{L}_1$.
2. Sort M by lexicographical order using radix sort.
3. For $i = 2, \dots, \frac{n}{2}$
 - (a) Construct \bar{L}_i .
 - (b) Sort \bar{L}_i by lexicographical order using radix sort.
 - (c) Construct $M \cap \bar{L}_i$ by merging \bar{L}_i with M and considering the l -mers which appear twice.
4. Output M .

In order to generate \bar{L}_i we need to introduce the following definitions and results.

Definition 6. Given x , with $|x| = l$, s with $|s| = m$ and $0 \leq h \leq l$, we define $\mathcal{N}(x, s, h) := \{y \triangleleft_l s : d_H(x, y) \leq h\}$.

Lemma 1. $\bar{L}_i = \bigcup_{x \triangleleft_l s_{2i-1}} \left(\bigcup_{y \triangleleft_l s_{2i}} B(x, d) \cap B(y, d) \right)$.

Proof. It follows by using De Morgan's theorem.

Lemma 2. $\bar{L}_i = \bigcup_{x \triangleleft_l s_{2i-1}} \left(\bigcup_{y \in \mathcal{N}(x, s_{2i}, 2d)} B(x, d) \cap B(y, d) \right)$.

Proof. It follows from fact that if $d_H(x, y) > 2d$ then $B(x, d) \cap B(y, d) = \emptyset$ and Lemma 2.

Taking into account Lemma 2 we define the procedure GenInter(i) which outputs \bar{L}_i .

Procedure GenInter(i)

1. Let $M = \emptyset$.
2. For each l -mer x of s_{2i-1} :
 - (a) Let $N = \mathcal{N}(x, s_{2i}, 2d)$.
 - (b) For each l -mer $y \in B(x, d)$
 - i. For each $x' \in N$, if $d_H(y, x') = d$ add y to M .

3. Output M .

It is useful when generating $\mathcal{N}(x, s_{2i}, 2d)$, to order the elements of it in ascending order of distance towards x , in this way we calculate the distance against the l -mers which are closer, which have greater probability of being in the intersection.

The following two results are straightforward.

Theorem 2. *PMSi can be implemented in $O(nm^2 + \binom{l}{d}|\Sigma|^d\mathcal{S}\frac{l}{w})$ time and in $O(\max_{i=1, \dots, \frac{n}{2}} |\bar{L}_i|)$ space, where $\mathcal{S} = \sum_{i=1}^{\frac{n}{2}} \sum_{x \triangleleft_l s_{2i-1}} \mathcal{N}(x, s_{2i}, 2d)$.*

Corollary 1. *PMSi can be implemented in $O(nm^2 \binom{l}{d}|\Sigma|^d \frac{l}{w})$ time using $O(m \binom{l}{d}|\Sigma|^d \frac{l}{w})$ space.*

2.3 PMSP

PMSP follows the following simple idea. For every l -mer x in s_1 it generates the set of neighbors of x and tries to guess if an l -mer y in that neighborhood is a motif by checking whether there are l -mers in s_i for $i = 2, \dots, n$ that are at distance d from it. This algorithm has been given in [6]. But we modify this algorithm in a critical way. The key observation is to notice that we do not need to evaluate the distance in all l -mers of s_i , but rather in the l -mers of s_i which are at distance $\leq 2d$ from x .

This can be said in a formal way in the straightforward lemmas 3 and 4.

Lemma 3. *The set of motifs M can be written as*

$$M = \bigcup_{x \triangleleft_l s_1} \left(B(x, d) \cap \bigcap_{i=2}^n L_i \right).$$

Lemma 4. *Let $x \triangleleft_l s_1$ and $x' \in B(x, d)$. We have that $x' \in \bigcap_{i=2}^n L_i$ iff*

$$\forall i = 2, \dots, n : \exists y_i \in \mathcal{N}(x, s_i, 2d) : d_H(x', y_i) = d.$$

Taking these considerations into account, we can write PMSP in the following way.

Algorithm PMSP

1. For each $x \triangleleft_l s_1$:
 - (a) Construct $N(i) = \mathcal{N}(x, s_i, 2d)$ for $i = 2, \dots, n$.
 - (b) For each $x' \in B(x, d)$:
 - i. Check if for **every** i ($2 \leq i \leq n$) there exists a $y_i \in N(i)$ such that $d_H(x', y_i) = d$.
 - ii. In the affirmative case output x' .

In PMSP we are using only $O(nm^2)$ space due to the fact that we need to calculate the distance from all l -mers in s_1 to all l -mers in s_i for $i = 2, \dots, n$. This can be done in $O(nm^2)$ time and space by using the strategy described in [4]. Hence the following results hold.

Theorem 3. *PMSP can be implemented in $O(nm^2 + \binom{l}{d}|\Sigma|^d\mathcal{S})$ time and in $O(nm^2)$ space, where $\mathcal{S} = \sum_{i=1}^{\frac{n}{2}} \sum_{x \triangleleft_i s_{2i-1}} \mathcal{N}(x, s_{2i}, 2d)$.*

Corollary 2. *PMSP can be implemented in $O(nm^2 \binom{l}{d}|\Sigma|^d)$ time and in $O(nm^2)$ space.*

2.4 Complexity Bounds In The Expected Case

If we assume that the set of strings s_i is constructed by picking each character from every sequence with equal probability we can prove sharper estimates in the expected case. In doing so we follow the approach used in [1], which basically assumes that the probability of occurrence of two different l -mers at different positions is independent from each other –which is not the case when the l -mers are in close proximity–. Furthermore we assume $\Sigma = \{A, C, G, T\}$.

The following results can be found in [1].

Lemma 5. *For a fixed $0 \leq d \leq l$ we can estimate the probability of two random l -mers being at a Hamming distance of $\leq d$ as*

$$p_d := \sum_{i=0}^d \binom{l}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{l-i}.$$

Lemma 6. *The expected number of (l, d) motifs for a set of strings s_i with $i = 1, \dots, n$, $|s_i| = m$ and such that s_i is picked at random using the previous model can be estimated by*

$$E(l, d, n) = 4^l (1 - (1 - p_d)^{m-l+1})^n.$$

By applying the previous lemma we can estimate in a better way the time and space complexity in the expected case.

Lemma 7. *The expected value of \mathcal{S} in the previous model can be estimated as $p_{2d}m^{\frac{n}{2}}$.*

Proof. It follows from the definition of $\mathcal{N}(x, s, h)$ and \mathcal{S} , linearity of expectation and Lemma 5.

Lemma 8. *The expected value of $\max_{i=1, \dots, \frac{n}{2}} |\bar{L}_i|$ in the previous model can be estimated as $E(l, d, 2) = 4^l (1 - (1 - p_d)^{m-l+1})^2$.*

Proof. It follows from Lemma 6.

Theorem 4. *The expected time complexity of PMSi is $O(p_{2d}nm^2\binom{l}{d}|\Sigma|^d\frac{l}{w})$ and the expected space complexity is $O(\frac{l}{w}E(l, d, 2))$. The expected time complexity of PMSP is $O(p_{2d}nm^2\binom{l}{d}|\Sigma|^d)$.*

Notice that the results of Theorem 4 imply that if we fix d , in the expected case we are going to get better results in PMSi and PMSP if we increase the value of l . Notice that it also implies the worst time and space complexities are attained for values of l and d such that $E(l, d, n)$ is greater than 1.

3 Experimental Results

As described before we follow the experimental setting described in [4] and [1]. That is, we fix $n = 20$, $m = 600$ and consider different values of l and d . In this model the strings are generated uniformly at random, and an l -mer is planted in random positions of these strings, mutating it in exactly d places.

We implemented versions of PMS1, PMSi and PMSP in C, and run our programs on a Pentium4 2.40 GHz machine with a core memory size of 1GB.

Our version of PMS1 is coded in C as well –the original version of PMS1 is implemented in Java–. Our version of PMS1 is slower than the original one but it is a good framework to test our results.

Table 1 shows the running time in seconds as well as the memory usage in percentage for our implementations of PMS1, PMSi and PMSP for $d = 3$ and $l = 13, 14, 15$. When we compare PMSi with PMS1 we get a speedup between 5 and 8 times and between 10 and 20 times less memory. In the case of PMSP we get a speedup between 20 and 50 times and the amount of memory used is very low. Let us also notice that the speedup in time as well as in memory use gets higher for higher values of l which is consistent with Theorem 4.

Algorithm	$l = 13$		$l = 14$		$l = 15$	
	Time (sec)	Mem (%)	Time(sec)	Mem (%)	Time(sec)	Mem (%)
PMSP	3	< 1%	2	< 1 %	1.5	< 1%
PMSi	28	1.2%	15	< 1 %	10	< 1%
PMS1	70	20.0%	79	21.0%	94	22.0%

Table 1. Comparison of PMS1 and PMSi for $d = 3$

Table 2 shows the running time and memory usage of PMSi and PMSP for instances that PMS1 could not solve because of large memory requirements. We have to notice that already existing extensions of PMS1 such as PMS2 are able to handle the case of $d = 4$ as reported in [6]. In this table we see that PMSi takes more time than PMS2 and for bigger values of l it gets faster and uses less memory. PMSP outperforms the two previous algorithms, using significantly less memory and the time it expends decreases as the value of l increases.

Algorithm	$l = 13$		$l = 14$		$l = 15$	
	Time (min)	Mem (%)	Time(min)	Mem (%)	Time(min)	Mem (%)
PMSP	2:12	< 1%	1:36	< 1%	1:34	< 1%
PMSi	28:30	35%	17:54	20%	10:23	12%
PMS2	3:48	95%	3:46	95%	3:37	95%

Table 2. Comparison of PMS2, PMSi, PMSP for $d = 4$

In Table 3 we get to see the behavior of PMSP, PMSi, PMS1 and Voting Algorithm in different challenging instances. A ‘-’ in the table means that the algorithm uses too much memory in that instance or its time is not reported. PMSi takes 3 times as much time as PMS1 in some cases, but it uses less memory, which allows it to solve the (13, 4) instance which cannot be solved by PMS1. PMSP clearly outperforms PMSi and PMS1 and uses less memory which allows it to solve bigger challenging instances. PMSP behaves well when compared with Voting, sometimes performing better and sometimes worse by no more than twice in time. However the main advantage of PMSP is its low use of memory which allows it to solve the (17, 6) instance, which was not reported solved before.

Algorithm	(9,2)	(11,3)	(13,4)	(15,5)	(17,6)
PMSP	0.6s	6.9s	152s	35m	12h
Voting	0.4s	8.6s	108s	22m	-
PMSi	4.6s	111.0s	18m	-	-
PMS1	3.0s	44.9s	-	-	-

Table 3. Time Comparison In Challenging Problems Instances

4 Conclusions

We presented algorithms PMSi and PMSP that are based upon ideas used in PMS1. These algorithms are more space efficient than PMS1 and improve the running time of PMS1 in several cases. PMSP clearly outperforms PMS1 and PMSi in challenging instances and compares well with already existing exact algorithms while making use of significantly less memory. This property allows it to solve the challenging instance (17, 6) which was not reported solved before.

References

1. J. Buhler and M. Tompa, Finding motifs using random projections, *Proceedings Fifth Annual International Conference on Computational Molecular Biology (RECOMB)*, April 2001.

2. Francis Y.L. Chin and Henry C.M. Leung, Voting Algorithms for Discovering Long Motifs, *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC2005)*, January 2005, pp. 261-271.
3. E. Eskin and P. Pevzner, Finding composite regulatory patterns in DNA sequences, *Bioinformatics* S1, 2002, pp. 354-363.
4. 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.
5. Price A., Ramabhadran S., Pevzner P.A. 2003. Finding subtle motifs by branching from sample strings, *Bioinformatics supplementary edition, Proceedings of the Second European Conference on Computational Biology (ECCB-2003)*.
6. S. Rajasekaran, S. Balla, and C.-H. Huang, Exact algorithms for the planted motif problem, *Journal of Computational Biology*, Oct 2005, Vol. 12, No. 8, pp. 1117-1128.
7. S. Rajasekaran, Motif search algorithms, in *Handbook of Computational Molecular Biology*, CRC Press, 2005.