Fast and Practical Algorithms for Planted (l, d) Motif Search

Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran

Abstract—We consider the planted (l, d) motif search problem, which consists of finding a substring of length l that occurs in a set of input sequences$\{s_1, \ldots, s_n\}$ with up to d errors, a problem that arises from the need to find transcription factor-binding sites in genomic information. We propose a sequence of practical algorithms, which start based on the ideas considered in PMS1. These algorithms are exact, have little space requirements, and are able to tackle challenging instances with bigger d, taking less time in the instances reported solved by exact algorithms. In particular, one of the proposed algorithms, PMSprune, is able to solve the challenging instances, such as (17, 6) and (19, 7), which were not previously reported as solved in the literature.

Index Terms—Planted motif search problem, challenging instances, exact algorithms, branch and bound algorithms.

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 [18] for a literature survey.

The problem consists of finding a substring of length l —which will be called the *motif*—that occurs in a set of input sequences $\{s_1, \ldots, s_n\}$ with up to d errors. The problem can be formally defined as follows (following [2] and [14]).

Definition 1. *Given a set of strings $\mathcal{S} := \{s_i\}_{i=1}^n$ 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 a substring x_i of length l such that x differs from x_i in at most d places for $i = 1, \ldots, n$.*

We will call x an (l, d) motif for $\mathcal{S}$.

This problem is known to be NP-complete [9] and a Polynomial Time Approximation Scheme (PTAS) exists for variants of the problem (for example, for the Common Approximate Substring and the Common Approximate String problems [11]). However, the high degree in the polynomial complexity of the PTAS makes it of little practical use.

Numerous algorithms have been implemented in order to solve the practical instances of this problem. Examples include Random Projection [2], MITRA [7], Winnower [14], Pattern Branching [16], Hybrid Sample and Pattern Driven Approaches [21], PMS1 [17], PMSP [6], SPELLER [19], SMILE [12], RISO [3], RISOTTO [15] CENSUS [8], and Voting [4]. Out of these algorithms, CENSUS, MITRA, PMS1, PMSP, SMILE, RISO, RISOTO, and Voting are among the exact category. (A motif search algorithm is exact if it always comes up with the correct answer(s); otherwise, it is *approximate*).

Many of these algorithms are able to work in a reasonable amount of time for practical instances, where $m = 600$ and $n = 20$. When we fix these values of m and n, we can define the concept of a challenging instance [2] as one where, if the strings are selected at random, the expected number of (l, d) motifs is more than 1. According to this definition, (11, 3), (13, 4), (15, 5), (17, 6), and (19, 7) are challenging instances. It should also be pointed out that, when this expected value is much more than 1, our algorithm will output all of the (l, d) motifs, which will not differentiate between planted and random motifs. In this case, one can follow the strategies in [20] and [5] where they consider different versions of the problem, based on the structure of the transcription factor-binding sites or making use of scoring schemes, which will rank the different motifs. In the implementation of our algorithm, we use a simplified version of a scoring system used by the algorithm Weeder [13], which is sometimes referred to as sequence specificity.

MITRA solves this problem and the more general one of dyads by considering a data structure called the Mismatch Tree and is able to tackle instances such as (15, 4). However, its theoretical time and space complexity are not discussed in [7].

The family of algorithms SPELLER, SMILE, RISO, and RISOTTO solve a more general problem and are based on the use of suffix trees. RISOTTO is the latest and the most efficient implementation among these [15]. In the case of the planted (l, d) motif search problem, their theoretical space complexity is $O(n^2 m)$ and their time complexity is $O(n^2 m \mathcal{N}(l, d))$, where $\mathcal{N}(l, d) := \sum_{i=1}^{d} \binom{l}{i} (|\Sigma| - 1)^d$. They are able to solve instances such as (15, 5) in 100 minutes.

CENSUS solves the problem by the use of n TRIES representing the l-mers of each sequence. Its space complexity is $O(nml)$ and the time complexity is $O(lnm\mathcal{N}(l, d))$, improving the theoretical bounds of the previous family of algorithms. Its implementation is able to tackle instances such as (15, 4). In [8], an algorithm with a claimed space

• *The authors are with the Department of Computer Science and Engineering, 371 Fairfield Rd., University of Connecticut, Storrs, CT 06269-3155.*
E-mail: {jdavila, rajasek}@engr.uconn.edu, sudha.balla@uconn.edu.

Manuscript received 14 July 2006; revised 27 Jan. 2007; accepted 30 Mar. 2007; published online 31 Oct. 2007.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0143-0706.

complexity of $O(m\mathcal{N}(l,d))$ and time complexity of $O(mn\mathcal{N}(l,d))$ is described, but no implementation is known.

PMS [17] solves the problem by making use of radix sorting; its space complexity is $O(\frac{l}{w}m\mathcal{N}(l,d))$ and its time complexity is $O(\frac{l}{w}nm\mathcal{N}(l,d))$, improving the time complexity of the first version of CENSUS. Modifications of this algorithm, such as PMS2, are able to solve instances such as (15, 4) but at the cost of using nearly 1 Gbyte of memory.

Voting is based on the idea of hashing and its space complexity is $O(m\mathcal{N}(l,d))$, with expected time complexity of $O(nm\mathcal{N}(l,d))$. Voting is able to solve challenging instances as big as (15, 5) in 22 minutes, but has high memory requirements that increase with d. However, it is not able to solve instances such as (17, 6) because of high memory requirements.

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.

PMSP is based on the idea of exploring the neighborhoods of the l-mers of the first sequence and checking whether the elements of such neighborhoods are (l,d) motifs. It uses $O(nm^2)$ memory and its time complexity is $O(\frac{l}{w}nm^2\mathcal{N}(l,d))$. Even though the theoretical bound is worse than that of PMS, it is able to solve challenging instances such as (15, 5) in 35 minutes and (17, 6) in 12 hours with small requirements of memory.

We also propose the algorithm PMSprune, which significantly extends and improves the ideas used in PMSP. PMSprune is proven to have a time complexity of $O(nm^2\mathcal{N}(l,d))$ and a space complexity of $O(nm^2)$, improving the theoretical bound of PMSP. From a practical perspective, PMSprune solves the challenging instance (17, 6) in 70 minutes and solves the instance (19, 7) in less than 10 hours. The latter instance was not previously reported solved in the literature.

The paper is structured as follows: In Section 2, we describe briefly the algorithm PMS1 and the new algorithms PMSi and PMSP, which improve upon the ideas considered in PMS1, and we prove their complexity bounds. In Section 3, we describe PMSprune incrementally and prove its complexity bounds. Finally, in Section 4.1, we describe some of the experimental results we have obtained and compare the results with some of the already existing exact algorithms.

2 Improved Algorithms Based on PMS1

PMS1 is a simple exact algorithm, introduced in [17], 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 (that is, 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 [17], we consider the following definitions.

Definition 2. *Given a string s with $|s| = m$ and a string x with $|x| = l$ with $l < m$. We say $x \lhd_l s$ if x is a subsequence of s. Equivalently, we say that x is an l-mer of s.*

Definition 3. *For any string x, with $|x| = l$, let $B_d(x) := \{y : |y| = l \text{ and } d_H(y,x) \leq d\}$, where d_H is the "Hamming" distance, that is, the number of places where the two strings differ.*

Notice that $\mathcal{N}(l,d) = \sum_{i=1}^{d} \binom{l}{i}(|\Sigma|-1)^d = |B_d(x)|$.

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

PMS1 works by doing the following simple steps:

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

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

Theorem 1. *PMS1 can be implemented in $O(\frac{l}{w}mn\mathcal{N}(l,d))$ time and $O(\frac{l}{w}m\mathcal{N}(l,d))$ 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 $i = 1, \ldots, \frac{n}{2}$ and then taking the intersection of these lists. The advantage 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 smaller than the size of $L_{s_{2i}}$. Independently, in [5], a similar idea, that is, of taking the intersection of pairs of vicinities of l-mers and hashing, is used in solving the so-called Extended Motif with Control Set.

In a more formal way, we have the following:

Algorithm PMSi.
1) Let $M := L_{s_1} \cap L_{s_2}$.
2) Sort M in a lexicographical order using radix sort.
3) For $i = 2, \ldots, \frac{n}{2}$
   a) Construct $\bar{L}_i := L_{s_{2i-1}} \cap L_{s_{2i}}$.
   b) Sort $\bar{L}_i$ in a lexicographical order using radix sort.
   c) Construct $M \cap \bar{L}_i$ by merging $\bar{L}_i$ with M and considering the l-mers that appear twice.
4) Output M.

In order to generate $\bar{L}_i$, we need to consider the following simple lemmas:

Lemma 1.

$$L_{s_i} \cap L_{s_{i+1}} = \bigcup_{x \lhd_l s_i} \left(\bigcup_{y \lhd_l s_{i+1}} B_d(x) \cap B_d(y) \right).$$

Proof. It follows by using the definition of L_{s_i} and De Morgan's theorem. □

Lemma 2.

$$L_{s_i} \cap L_{s_{i+1}} = \bigcup_{x \lhd_l s_i} \left(\bigcup_{x \lhd_l s_{i+1}} \{B_d(x) \cap B_d(y) : d_H(x,y) \leq 2d\} \right). \quad (1)$$

Proof. It follows from fact that if $d_H(x,y) > 2d$, then $B_d(x) \cap B_d(y) = \emptyset$ and from Lemma 1. □

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

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

It is useful when generating N to order the elements of it in ascending order of distance toward x; in this way, we calculate the distance against the l-mers which are closer, which have greater probability of being in the intersection.

It should also be noted that we represent every l-mer as a sequence of integers, as in [17]. In Step 2.b.i, the distance is calculated by using an array in which the distances of the sequences represented by integers are cached.

Taking these into account, the following two results are straightforward.

Theorem 2. *PMSi can be implemented in* $O(nm^2 + \frac{l}{w}\mathcal{S}\mathcal{N}(l,d))$ *time and in* $O(\max_{i=1,\ldots,\frac{n}{2}} |L_{s_{i-1}} \cap L_{s_i}|)$ *space, where* $\mathcal{S} = \sum_{i=1}^{\frac{n}{2}} \sum_{x \lhd_l s_{2i-1}} \{y \lhd_l s_{2i} : d_H(x,y) \leq 2d\}|$.

Corollary 1. *PMSi can be implemented in* $O(nm^2 \frac{l}{w}\mathcal{N}(l,d))$ *time using* $O(m\frac{l}{w}\mathcal{N}(l,d))$ *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 $i = 2, \ldots, n$ that are at distance $\leq d$ from it. This algorithm has been given in [17]. However, 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*

$$M = \bigcup_{x \lhd s_1} \left(B_d(x) \cap \bigcap_{i=2}^{n} L_{s_i} \right).$$

Proof. It follows by definition of L_{s_i} and M. □

Lemma 4. *Let* $x \lhd_l s_1$ *and* $x' \in B_d(x)$. *We have that* $x' \in \bigcap_{i=2}^{n} L_{s_i}$ *iff* $\forall i = 2, \ldots, n$,

$$\exists y_i \lhd_l s_i : (d_H(x,y_i) \leq 2d) \wedge (d_H(x',y_i) = d).$$

Proof. It follows by Lemma 3 and the use of Lemma 2 iteratively. □

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

Algorithm PMSP.
1) Let $M = \emptyset$.
2) For each $x \lhd_l s_1$:
   a) Let $N(i) = \{y \lhd s_i : d_H(x,y) \leq 2d\}$ for $i = 2, \ldots, n$.
   b) For each $x' \in B_d(x)$:
      i) Check if for **every** $i (2 \leq i \leq n)$, there exists a $y_i \in N(i)$ such that $d_H(x', y_i) \leq d$.
      ii) In the affirmative, add x' to M.
3) Output M.

In PMSP, we are using $O(m^2 n)$ 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 $i = 2, \ldots, n$. This can be done in $O(m^2 n)$ time and space by using the strategy described in [14]. Hence, the following results hold.

Theorem 3. *PMSP can be implemented in* $O(nm^2 + \frac{l}{w}\mathcal{N}(l,d)\mathcal{S})$ *time and in* $O(nm^2)$ *space.*

Corollary 2. *PMSP can be implemented in* $O(nm^2 \frac{l}{w}N(l,d))$ *time and in* $O(nm^2)$ *space.*

2.4 Complexity Bounds in the Expected Case

According to Corollaries 1 and 2, we have that the newly designed algorithms are slower than PMS1 by a factor of m in the worst case. In this section, we will argue that, in the expected case, the behavior is better.

If we assume that the set of strings s_i is constructed by picking each character of every sequence randomly with equal probability, we can prove sharper estimates in the expected case. In doing so, we follow the approach used in [2], which basically assumes that the probability of the 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 that $\Sigma = \{A, C, G, T\}$.

The following results can be found in [2]:

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*

$$p_d := \sum_{i=0}^{d} \binom{l}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{l-i}. \quad (2)$$

It should be pointed out that (2) holds only when the probability of the four letters is the same. However, in biological applications, this is rarely the case, so, if we call p_A the probability of occurrence of A, p_C the probability of occurrence of C, and so on, we have (3):

$$p_d := \sum_{i=0}^{d} \binom{l}{i} (1-q)^i q^{l-i} \text{ where } q = p_A^2 + p_C^2 + p_G^2 + p_T^2. \quad (3)$$

Lemma 6. *Let x be an l-mer in the model previously described. Let $\mathcal{S} = \{s_i\}_{i=1}^n$ be a set of strings s_i constructed in the same manner. We have that*

$$\Pr\{\bar{d}_H(x,\mathcal{S}) \geq d\} = \left(1-(1-p_d)^{m-l+1}\right)^n. \quad (4)$$

Lemma 7. *The expected number of (l,d) motifs for a set of strings s_i with $i = 1,\ldots,n$, $|s_i| = m$ and such that s_i is picked at random using the previous model can be estimated by*

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

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

Lemma 8. *The expected value of $\mathcal{S}$ can be estimated as $p_{2d}m^2\frac{n}{2}$.*

Proof. It follows from the definition of $\mathcal{L}_{i,d}(x)$ and $\mathcal{S}$, linearity of expectation, and Lemma 5. □

Lemma 9. *The expected value of $|\bar{L}_i|$ for $i = 1,\ldots,\frac{n}{2}$ can be estimated as $E(l,d,2) = 4^l(1-(1-p_d)^{m-l+1})^2$.*

Proof. It follows from Lemma 7. □

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

One consequence of Theorem 4 is that the proposed algorithms will be better if $p_{2d} < \frac{1}{m}$. Also notice that they 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 that 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 PMSprune: A Branch and Bound Algorithm

Before proceeding, we will introduce some concepts that will allow us to present the algorithms in a more precise and "geometrical" way.

Definition 5. *Given strings $s := s[1]\ldots s[m]$ and $x := x[1]\ldots x[l]$ with $l < m$, we denote by*

$$\bar{d}_H(x,s) = \min_{x' \lhd_l s} d_H(x,x').$$

Definition 6. *Given a string $x = x[1]\ldots x[l]$ and a set of strings $\mathcal{S} := \{s_1\ldots s_n\}$ with $|s_i| = m$ for $i = 1,\ldots,n$ and $l < m$, we denote by*

$$\bar{d}_H(x,\mathcal{S}) := \max_{i=1}^n \bar{d}_H(x,s_i) = \max_{i=1}^n \min_{r \lhd_l s_i} d_H(x,r).$$

The advantage of this notation is that it allows us to write Definition 1 in a more concise way. The following are equivalent ways of stating that x is an (l,d) motif for $\mathcal{S} := \{s_1,\ldots,s_n\}$:

1. $\bar{d}_H(x,\mathcal{S}) \leq d$.
2. $\exists y \lhd_l s_1$: $x \in B_d(y) \wedge \bar{d}_H(x,\{s_2,\ldots,s_n\}) \leq d$.

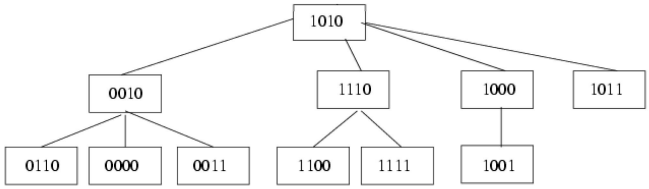

Fig. 1. $\mathcal{T}(1010)$ with $d = 2$ and $\Sigma = \{0,1\}$.

PMSprune follows the same strategy as PMSP: For every l-mer y in s_1, it generates the set of neighbors of y and, for each of them, checks whether this is a motif or not. However, it extends and improves in a significant way these ideas by using several refinements, which can be briefly described as follows:

1. It generates the vicinity of every l-mer in a branch and bound manner. In this method, these l-mers will correspond to nodes in a tree of height at most d.
2. For an l-mer x, which corresponds to a node in the tree of height h, it uses the value of $\bar{d}_H(x,\mathcal{S})$ and h to prune the descendants of x.
3. It dynamically prunes the l-mers in s_i for $i = 2,\ldots,n$ that are considered for the calculation of $\bar{d}_H(\cdot,\mathcal{S})$.
4. It calculates the value of $\bar{d}_H(\cdot,\mathcal{S})$ in an incremental way, taking into account the way in which the neighborhood is generated.

The details of the above refinements will be described in the following sections.

We also present a variant of the problem that allows us to solve a more general problem, which can be potentially more useful in the biological realm.

3.1 Branch and Bound Strategy

We introduce the following notations in order to introduce the ideas in a precise way.

Definition 7. *Given $y = y[1],\ldots,y[l]$ and $0 \leq d < l$, we define a tree $\mathcal{T}(y)$ by the following rules:*

1. *A node at level $0 \leq h \leq d$ is of the form $(x,h,\{i_1,\ldots,i_h\})$, where $1 \leq i_1 < i_2 < \ldots < i_h \leq l$.*
2. *The root level is $(y,0,\emptyset)$.*
3. *Given a node $(x,n,\{i_1,\ldots,i_n\})$, where $0 \leq n < d$, the set of children of x will be nodes of the form $(x',n+1,\{i_1,\ldots,i_n,i_{n+1}\})$, where $i_n < i_{n+1} \leq l$, and $x[j] = x'[j]$ for $j \neq i_{n+1}$ and $x'[i_{n+1}] \neq x[i_{n+1}]$.*

Notice that $\mathcal{T}(y)$ represents, in a very natural way, the set of l-mers which are at distance $\leq d$ from a given l-mer y. Furthermore, it is easy to see that, if x is at level h, then $d_H(x,y) = h$ and that, if x' is a child of x, then $d_H(x,x') = 1$. In Fig. 1, we show an example of such a tree.

We can describe in precise terms our first version of PMSprune as follows:

Algorithm PMSprune—first version.

1) For each $y \lhd_l s_1$:
   a) Traverse $\mathcal{T}(y)$ in a depth-first search way, evaluating $\bar{d}_H(x,\mathcal{S})(x \in \mathcal{T}(y))$. If $\bar{d}_H(x,\mathcal{S}) \leq d$ output x.

One interesting and very useful observation is the fact that if x and x' are two l-mers with $d_H(x,x') = 1$, then

$\bar{d}(x', \mathcal{S}) = \bar{d}(x, \mathcal{S}) + \delta$, where $\delta \in \{-1, 0, 1\}$. This implies that the value of $d(\cdot, \mathcal{S})$ does not differ too much between an l-mer x and its children x' in $\mathcal{T}(y)$. This property in particular will allow us to state the following pruning strategy.

Theorem 5. *Let $x \in \mathcal{T}(y)$ and let $\mathcal{S} = \{s_1, \ldots, s_n\}$. Assume that $\bar{d}_H(x, \mathcal{S}) = d + \Delta$ and that x appears at level h, that is, $d_H(y, x) = h$). Then, the following statements hold:*

1. *If $\Delta > d - h$, no descendant of x in $\mathcal{T}(y)$ will be a motif.*
2. *If $\Delta = d - h$, the motifs that are descendants of x will also be descendants of x', where x' is a child of x and $\bar{d}_H(x', \mathcal{S}) = \bar{d}_H(x, \mathcal{S}) - 1$.*
3. *If $\Delta = d - h - 1$, the motifs that are descendants of x will also be descendants of x', where x' is a child of x and $\bar{d}_H(x', \mathcal{S}) \leq \bar{d}_H(x, \mathcal{S})$.*

Proof. Let us illustrate the validity of the result by showing the proof for the first statement. Suppose that x is a motif and that x' is a descendant of x in $\mathcal{T}(y)$, that is, there exists $\{y_i\}_{i=1}^{n}$, where $n \leq d - h$, $y_1 = x$, $y_n = x'$, and with y_{i+1} being a child of y_i. By the property described before the theorem, we have that

$$d + \Delta = \bar{d}_H(x, \mathcal{S}) \leq \bar{d}_H(x', \mathcal{S}) + (d - h) \leq 2d - h.$$

Hence, $\Delta \leq d - h$ and the result follows by contradiction. □

Employing Theorem 5, we can construct a second and improved version of Algorithm PMSprune.

Algorithm PMSprune—second version.

1) For each $y \lhd_l s_1$:
   a) Traverse $\mathcal{T}(y)$ in a depth-first search way evaluating $\bar{d}_H(x, \mathcal{S})(x \in \mathcal{T}(y))$.
      i) If $\bar{d}_H(x, \mathcal{S}) \leq d$, output x.
      ii) If $\bar{d}_H(x, \mathcal{S}) - d > d - h$, prune all of the descendants from x.
      iii) If $\bar{d}_H(x, \mathcal{S}) - d = d - h$, consider only x' such that x' is a child of x, and $\bar{d}_H(x', \mathcal{S}) = \bar{d}_H(x, \mathcal{S}) - 1$.
      iv) If $\bar{d}_H(x, \mathcal{S}) - d = d - h - 1$, consider only x' such that x' is a child of x and $\bar{d}_H(x', \mathcal{S}) \leq \bar{d}_H(x, \mathcal{S})$.

3.2 Incremental Calculation of Distance

We are interested in the problem of incrementally calculating $\bar{d}_H(x, \mathcal{S})$ as we progress in exploring the tree $\mathcal{T}(y)$ in an efficient way.

In the rest of the discussion, we let x and x' be strings of length l. We assume that x and x' are nodes in $\mathcal{T}(y)$ and x' is a child of x with $x[i] \neq x'[i]$. We first observe that the following formula holds for z, an l-mer:

$$d_H(x', z) = \begin{cases} d_H(x, z) - 1 & \text{if}(x'[i] = z[i]) \\ d_H(x, z) & \text{if}(x'[i] \neq z[i]) \wedge (x[i] \neq z[i]) \\ d_H(x, z) + 1 & \text{otherwise.} \end{cases} \quad (5)$$

Based on this observation, we can design the following algorithm:

Algorithm UpdateDistance—first version.

- **Input:** x and x' with $x[i] \neq x'[i]$, $\mathcal{S} = \{s_1, \ldots, s_n\}$ and $\{\mathcal{D}_i\}_{i=1}^{n}$ with $\mathcal{D}_i = \{d_{i,1}, \ldots, d_{i,m-l+1}\}$, where $d_{i,j}$ is the distance between the jth l-mer of s_i and x.
- **Output:** $\bar{d}_H(x', \mathcal{S})$ and $\{\mathcal{D}'_i\}_{i=1}^{n}$ with $\mathcal{D}'_i = \{d'_{i,1}, \ldots, d'_{i,m-l+1}\}$, where $d'_{i,j}$ is the distance between the jth l-mer of s_i and x'.

1) for $i = 1, \ldots, n$
   a) for $j = 1, \ldots, m - l + 1$ calculate $d'_{i,j}$ using (5) knowing $x'[i]$, $d_{i,j}$ and $x[i]$.
   b) Calculate the minimum of $\mathcal{D}'_i$ and call it D'_i.
2) Output the maximum of D'_i with $i = 1, \ldots, n$.

The following is a straightforward result.

Theorem 6. *Algorithm UpdateDistance can be implemented in $O(mn)$ time and $O(mn)$ space.*

Another important observation is that, when we are exploring a node x at level h in the tree $\mathcal{T}(y)$, we just need to keep track of the l-mers in s_i whose distance is $\leq 2d - h$ from x.

Based on this, we introduce our last version of UpdateDistance.

Algorithm UpdateDistance—second version.

- **Input:** x and x' with $x[i] \neq x'[i]$, $\mathcal{S} = \{s_1, \ldots, s_n\}$, $0 \leq h \leq d$ and $\{\mathcal{D}_i\}_{i=1}^{n}$ with $\mathcal{D}_i = \{(r, d_{i,j}) : (r \lhd_l s_i) \wedge (d_{i,j} := d_H(x, r) \leq 2d - h)\}$
- **Output:** $\bar{d}_H(x', \mathcal{S})$ and $\{\mathcal{D}'_i\}_{i=1}^{n}$ with $\mathcal{D}'_i = \{(r', d'_{i,j}) : (r' \lhd_l s_i) \wedge (d'_{i,j} := d_H(x', r') \leq 2d - h - 1)\}$.

1) for $i = 1, \ldots, n$
   a) for $(r, d_{i,j}) \in \mathcal{D}_i$ calculate $d'_{i,j}$ using (5).
   b) If $d'_{i,j} \leq 2d - h - 1$ include $(r, d'_{i,j})$in $\mathcal{D}'_i$.
   c) Calculate the minimum of $\mathcal{D}'_i$ and call it D'_i.
2) Output the maximum of D'_i with $i = 1, \ldots, n$.

Theorem 7. *Algorithm UpdateDistance can be implemented in $O(\sum_{i=1}^{n} |\mathcal{D}_i|)$ time and $O(\sum_{i=1}^{n} |\mathcal{D}_i|)$ space.*

In the context of the second version of PMSprune, we are interested in implementing Steps iii) and iv) of the pruning strategy in an efficient way.

In particular, given x, we are interested in generating all x', children of x such that $\bar{d}_H(x', \mathcal{S}) = \bar{d}_H(x, \mathcal{S}) - 1$ or such that $\bar{d}_H(x', \mathcal{S}) \leq \bar{d}_H(x, \mathcal{S})$.

The following list of results and definitions will allow us to do that:

Lemma 10. *Let $\mathcal{R} = \{r_1, \ldots, r_n\}$ with $|r_j| = l$ and $d_H(x, r_j) = d$, for $j = 1, \ldots, n$. Let $d' = \min_{i=1}^{n} d_H(x', r_i)$. Then, the following holds:*

$$d' = \begin{cases} d - 1 & \text{iff } \exists j : r_j[i] = x'[i]. \\ d + 1 & \text{iff } \forall j : r_j[i] \neq x'[i] \wedge \forall j : r_j[i] = x'[i]. \\ d & \text{otherwise.} \end{cases}$$

Proof. The proof follows easily from (5). □

Let us introduce some notation:

Definition 8. *Given $\mathcal{S} = \{s_1, \ldots, s_n\}$ with $|s_j| = m$, we define the following:*

1. $\mathcal{I}_d(x) = \{i : \bar{d}_H(s, s_i) = d\}$.
2. $\mathcal{L}_{i,d}(x) = \{r \lhd_l s_i : d_H(s, r) = d\}$.
3. $\mathcal{M}_{d,k}(x) = \bigcup_{i \in \mathcal{I}_d(x)} \mathcal{L}_{i,k}(x)$.

Lemma 11. *Let $d = \bar{d}_H(x,\mathcal{S})$ and $d' = \bar{d}_H(x',\mathcal{S})$. The following statement is true:*

$$d' = \begin{cases} d-1 & \text{iff } \forall j \in \mathcal{I}_d(x): \\ & \exists r \in \mathcal{L}_{j,d} : d_H(x',r) = d-1) \wedge \\ & (\forall j \in \mathcal{I}_{d-1}(x) : \exists r \in \mathcal{L}_{j,d-1}(x) : d_H(x',r) \leq d-1). \\ d+1 & \text{iff } \exists j \in \mathcal{I}_d(x): \\ & \forall r \in \mathcal{L}_{j,d}(x) : d_H(x',r) = \\ & d+1 \wedge \forall t \in \mathcal{L}_{j,d+1}(x) : d_H(x',t) \geq d+1. \\ d & \text{otherwise.} \end{cases}$$

Proof. Follows from Definition 11. □

Theorem 8. *Let $d = \bar{d}_H(x,\mathcal{S})$ and $d' = \bar{d}_H(x',\mathcal{S})$. The following statements are true:*

- *$d' = d-1$ iff (6) and (7) hold.*

$$\forall j \in \mathcal{I}_d(x) : \exists r \in \mathcal{L}_{j,d} : r[i] = x'[i], \tag{6}$$

$$\begin{aligned} &\forall j \in \mathcal{I}_{d-1}(x) : (\exists r \in \mathcal{L}_{j,d-1} : r[i] = x'[i]) \vee \\ &(\exists r \in \mathcal{L}_{j,d-1} : r[i] \neq x[i]). \end{aligned} \tag{7}$$

- *$d' = d+1$ iff (8) holds.*

$$\begin{aligned} &\exists j \in \mathcal{I}_d(x) : (\forall r \in \mathcal{L}_{j,d}(x) : r[i] = x[i]) \wedge \\ &(\forall r \in \mathcal{L}_{j,d}(x) : r[i] \neq x'[i]) \wedge \\ &(\forall r \in \mathcal{L}_{j,d+1}(x) : r[i] \neq x'[i]). \end{aligned} \tag{8}$$

- *$d' = d$ otherwise.*

Proof. Follows from Lemmas 10 and 11. □

Notice that, as a consequence of Theorem 8, the value of d' depends only on $x'[i]$, $\mathcal{M}_{d,d}(x)$, $\mathcal{M}_{d,d+1}(x)$, and $\mathcal{M}_{d-1,d-1}(x)$. Furthermore, d' will depend only on a particular set of predicates defined over i and σ.

Definition 9. *Given $1 \leq i \leq l$ and $\sigma \in \Sigma$, we define the following predicates:*

1. $\varphi_1^{i,j}(\sigma) := \exists r \in \mathcal{L}_{j,d} : r[i] = \sigma.$
2. $\varphi_2^{i,j}(\sigma) := \exists r \in \mathcal{L}_{j,d} : r[i] \neq \sigma.$
3. $\varphi_3^{i,j}(\sigma) := \exists r \in \mathcal{L}_{j,d+1} : r[i] = \sigma.$
4. $\varphi_4^{i,j}(\sigma) := \exists r \in \mathcal{L}_{j,d-1} : r[i] \neq \sigma.$

Theorem 9. *Let $d = \bar{d}_H(x,\mathcal{S})$, $d' = \bar{d}_H(x',\mathcal{S})$, $\sigma = x[i]$, and $\sigma' = x'[i]$. Then, the following statements hold:*

$$\begin{aligned} d' = d-1 \text{ iff } &\Big(\forall j \in \mathcal{I}_d(x) : \varphi_1^{i,j}(\sigma')\Big) \wedge \\ &\Big(\forall j \in \mathcal{I}_{d-1}(x) : \varphi_4^{i,j}(\sigma)\Big), \end{aligned} \tag{9}$$

$$d' \leq d \text{ iff } \forall j \in \mathcal{I}_d(x) : \varphi_2^{i,j}(\sigma) \vee \varphi_3^{i,j}(\sigma'). \tag{10}$$

Proof. Follows from Theorem 8 and Definition 9. □

Taking this into consideration, we describe our final implementation of PMSprune.

Algorithm PMSprune—third version.

1) For each $y \lhd_l s_1$:
   a) Traverse $\mathcal{T}(y)$ in a depth-first search fashion evaluating $\bar{d}_H(x,\mathcal{S})$ using the second version of algorithm UpdateDistance.
      i) If $\bar{d}_H(x,\mathcal{S}) \leq d$, output x.
      ii) If $\bar{d}_H(x,\mathcal{S}) - d > d - h$, prune all descendants from x.
      iii) If $\bar{d}_H(x,\mathcal{S}) - d = d - h$, consider only x' that satisfy (9).
      iv) If $\bar{d}_H(x,\mathcal{S}) - d = d - h - 1$, consider only x' that satisfy (10).

3.3 A Variant of the Problem

In the (l,d) motif problem, one assumes that, in every sequence, there should be at least one approximate occurrence of the motif. However, in the biological realm, this might not be the case, so we address a more general version of the problem.

Definition 10. *Given a set of strings $\mathcal{S} := \{s_i\}_{i=1}^n$ over an alphabet Σ with $|s_i| = m$ and l, d, and q with $0 \leq d < l < m$ and $0 < q \leq n$, we define the (l,d) motif search problem with q instances as that of finding a string x with $|x| = l$ such that there exists q different i such that, for each i, s_i has a substring x_i of length l such that x differs from x_i in at most d places for $i = 1,\ldots,n$.*

We will call x an (l,d) motif with q instances for $\mathcal{S}$.

In order to solve this version of the problem, we need to introduce the following notation.

Definition 11. *Given a string $x = x[1]\ldots x[l]$ and a set of strings $\mathcal{S} := \{s_1 \ldots s_n\}$ with $|s_i| = m$ for $i = 1,\ldots,n$ and $l < m$ and q with $1 \leq q \leq n$, we denote by $\bar{d}_H^{(q)}(x,\mathcal{S})$ the value of the qth quantile of the set of values $\{\bar{d}_H(x,s_1),\ldots,\bar{d}_H(x,s_n)\}$. It is clear that $\bar{d}_H^{(n)}(x,\mathcal{S}) = \bar{d}_H(x,\mathcal{S})$.*

It is clear that the following are equivalent ways of stating that x is an (l,d) motif with q instances for $\mathcal{S} := \{s_1,\ldots,s_n\}$.

1. $\bar{d}_H^{(q)}(x,\mathcal{S}) \leq d$.
2. $$\begin{aligned} &\exists i \in [1, n-q+1] : \exists y \lhd_l s_i : \\ &x \in B_d(y) \wedge \bar{d}_H^{(q)}(x,\mathcal{S}) \leq d. \end{aligned}$$

It should be pointed out that Theorem 5, that is, the pruning strategy, also works for the case of $d^{(q)}(\cdot,\mathcal{S})$. This is due to the fact that if x and x' are two l-mers with $d_H^{(q)}(x,x') = 1$, then $\bar{d}^{(q)}(x',\mathcal{S}) = \bar{d}^{(q)}(x,\mathcal{S}) + \delta$, where $\delta \in \{-1,0,1\}$. Notice also that Theorem 6 also holds for the case of updating this new distance.

This observation gives rise to the following algorithm:

Algorithm qPMSprune.

1) For $i = 1,\ldots,n-q+1$
   a) For each $y \lhd_l s_i$:

TABLE 1
Time Comparison in Challenging Problem Instances

Algorithm	(11,3)	(13,4)	(15,5)	(17,6)	(19,7)
PMSprune	5s	53s	9m	69m	9.2 h
PMSP	6.9s	152s	35m	12h	–
Voting	8.6s	108s	22m	–	–
RISOTTO	54s	600s	100m	12h	–
PMSi	111s	18m	–	–	–
PMS1	45s	–	–	–	–

i) Traverse $\mathcal{T}(y)$ in a depth-first search way evaluating $\bar{d}_H^{(q)}(x, \mathcal{S})(x \in \mathcal{T}(y))$.
   A) If $\bar{d}_H^{(q)}(x, \mathcal{S}) \leq d$, output x.
   B) If $\bar{d}_H^{(q)}(x, \mathcal{S}) - d > d - h$, prune all the descendants from x.
   C) If $\bar{d}_H^{(q)}(x, \mathcal{S}) - d = d - h$, consider only x' such that x' is a child of x and $\bar{d}_H^{(q)}(x', \mathcal{S}) = \bar{d}_H^{(q)}(x, \mathcal{S}) - 1$.
   D) If $\bar{d}_H^{(q)}(x, \mathcal{S}) - d = d - h - 1$, consider only x' such that x' is a child of x and $\bar{d}_H^{(q)}(x', \mathcal{S}) \leq \bar{d}_H^{(q)}(x, \mathcal{S})$.

3.4 Time Complexity Results

Theorem 10 estimates the runtime of the algorithm PMSprune. It is noteworthy that it shows PMSprune outperforms PMSP by a factor of $\frac{l}{w}$.

Theorem 10. *Algorithm PMSprune can be implemented in $O(nm^2\mathcal{N}(l,d))$ time and in $O(nm^2)$ space.*

Proof. This follows easily from the fact that $|B_d(x)| = \mathcal{N}(l,d) = \sum_{i=1}^{d} \binom{l}{i}(|\Sigma|-1)^d$ and from the fact that we calculate $\bar{d}_H(\cdot, \mathcal{S})$ in an incremental fashion in accordance with Theorem 6. □

In a similar way, we have the following theorem for the case of qPMSprune.

Theorem 11. *Algorithm qPMSprune can be implemented in $O((n-q+1)nm^2\mathcal{N}(l,d))$ time and in $O(nm^2)$ space.*

We would like to estimate the time PMSprune takes on the average. To do so, we assume the model of Section 2.4.

Using (4), it is possible to estimate, for fixed values of n, m, and l, a value d' such that $\Pr\{\bar{d}_H(x, \mathcal{S}) \geq d'\}$ is close to 1.

This implies that, when we are evaluating $\bar{d}_H(\cdot, \mathcal{S})$ at $x \in \mathcal{T}(y)$, we will be getting the value d' in most of the cases. Hence, if x is at level h and $h > 2d - d'$, we will be pruning the descendants of x. This means that we will only go up to $d < 2d - d' + 1$ levels in most of the cases; hence, we have roughly that the expected time complexity of PMSprune is

$$O(nm(m + p_{2d}m\mathcal{N}(l, 2d - d' + 1))). \quad (11)$$

Let n, m, and l be fixed and suppose $(l, \tilde{d})$ is a challenging instance. Then, (11) implies that the time required by PMSprune to solve the (l, d') motif problem with $d' < \tilde{d}$ reduces by a factor of l^2 when compared with the time required to solve the (l, d) problem on the average.

TABLE 2
Time Comparison of PMSprune for Different (l, d) Instances

l	d	Time	d	Time	d	Time
15	5	9m	4	4s	3	0.2s
17	6	69m	5	33s	4	0.5s
19	7	9.2h	6	5m	5	1.1s

For example, when using $n = 600$, $m = 20$, and $l = 15$, we have that $d' = 6$. Thus, on the average, the solution of (15, 5) will take time proportional to l^5mn, but the solution of (15, 4) will take time proportional to l^3mn.

4 Experimental Results

4.1 Experiments on Artificial Data

As described before, we follow the experimental setting described in [14] and [2]. In particular, we fix $n = 20$ and $m = 600$ 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.

PMS1, PMSi, PMSP, and PMSprune are coded in C and we have run these programs on the same Linux machine with a Pentium4 2.40 GHz processor and a core memory size of 1 Gbyte. The results of Voting are the ones reported in [4] on a machine with the same processor and core memory size of 512 Mbytes. The RISOTTO program was coded in C and was run on the same Linux machine as the other programs.

We compare the results of PMSprune, PMSP, PMS1, PMSPi, RISOTTO, and Voting on some challenging instances in Table 1. It should be pointed out that the implementations of PMS1, PMSi, and PMSP work in the slightly easier version of the problem where one assumes that the instances of the motif occur at exactly distance d. The implementations PMSprune, RISOTTO, and Voting work for the problem as in Definition 1. When we write "–", it means that the algorithm uses too much memory (more than 1 Gbyte) in the instance, takes too long (more than 12 hours), or its time is not reported.

PMSi takes three 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. PMSprune consistently outperforms the other algorithms and the speedup is better for increasing values of d; for example, it outperforms PMSP on (17, 6) by a factor of more than 10.

In Table 2, we can see the behavior of Algorithm PMSprune for several values of (l, d). Notice that (as predicted at the end of Section 3.4) the time taken by the algorithm abruptly reduces for $d' < d$ such that (l, d) is a challenging instance, so, for example, solving (17, 5) takes 33 sec, whereas solving (17, 6) takes 69 min.

TABLE 3
Results on Biological Data

Gene	Detected Planted Motif	Published Motif	(l, d) used
preproinsulin	CAGCCTCAGCCCCTT	CCTCAGCCCC	$(15, 2)$
DHFR	ATTTCGTGGGCA	ATTTCnnGCCA	$(11, 2)$
metallothionein	CTC TGCACACGGCCC	TGCRCYCGG	$(15, 2)$
c-fos	CCAAATTTG	CCATATTAGAGACTCT	$(9, 2)$
Yeast ECB	CCCATTAAGGAAA	TTtCCcnntnaGGAAA	$(13, 2)$

4.2 Mining Transcription Factor-Binding Sites

We test our program with the test set discussed in [1], which corresponds to more realistic biologic data. This test set has been used extensively and existing algorithms [1], [2], [21] [17] are able to find known transcription regulatory elements in them by taking orthologous sequences from a variety of organisms. These data differ substantially from the random model employed to create synthetic data. We obtain the same results published in [17], which we summarize in Table 3. However, it should be pointed out that many of the existing algorithms are known to find solutions on this data set.

On the other hand, a more recent assessment has been described in [22]. The test sets available make use of Transcription Factor-Binding Sites, which vary in length in every sequence and can be absent in some of the sequences. In order to test our algorithm on this sort of data, one needs to fix the values of l, d, and q. This limitation can be overcome by trying different values of l between, say, 8 and 12, values of d between 1 and 4, and values of q larger than 50 percent of the number of sequences.

However, in such cases, we get a large number of spurious hits; hence, a scoring scheme needs to be used in order to classify the different motifs obtained. We use a simplified version of the scoring scheme used in Weeder [13], called sequence specificity, which basically calculates $\sum_{i=1}^{n} \log(\frac{1}{E_i(p,d_i)})$, where $E_i(p, d_i)$ is the expected number of occurrences of p in sequence s_i with up to d_i errors ($d_i \leq d$ and is the minimum value such that p occurs with up to d_i errors in s_i), assuming that every letter occurs independently in every string.

Given that Weeder obtains good results when used on the data set for yeast in [22], we tested qPMSprune on these data sets. In particular, we sorted all of the motifs found by qPMSprune in the order of their ranks. The solution provided by Weeder fell within the 10 percent best scoring solutions provided by PMSprune. To mention a specific example, on the data set **yst09g**, we obtain the motif TAGCCCGCC when tried with parameters $l = 8$, $d = 3$, and $q = \frac{n}{2}$, which is the motif reported by Weeder. It should be pointed out that one could eventually use these scoring systems directly in the pruning algorithm and not as a postprocessing step. This would take considerable effort and will be done in future work; however, the results obtained so far are encouraging and are worthy of consideration.

We should also point out that qPMSprune with this scoring scheme does not find the transcription factor-binding sites in most of the cases for data sets that come from the human or mouse genome. The reasons for this behavior may lie in the scoring itself, as was pointed out in [10]. Thus, in our future work, we plan to consider other types of scoring schemes.

In addition, note that the ability of any algorithm to find transcription factor-binding sites will crucially depend on the scoring scheme used. Our planted motif algorithms are guaranteed to come up with a set of solutions that will contain the correct solution. If researchers come up with general scoring schemes that are capable of discerning the right site, they can be immediately applied on our algorithms to identify the correct sites. In other words, there are two phases in finding transcription factor-binding sites. The first phase corresponds to the planted motif problem and the second phase corresponds to applying an appropriate scoring scheme to eliminate unwanted motifs.

5 Conclusions

We presented algorithms PMSi, PMSP, and PMSPrune that are based upon ideas used in PMS1. These algorithms are more space efficient than PMS1 and improve the runtime of PMS1 in several practical cases, even though their time complexity in the worst case could be higher than that of PMS1. PMSP and PMSprune are able to tackle challenging instances such as (17, 6) and PMSprune is even able to solve (19, 7) in a matter of hours. On the experimental setting, PMSprune outperforms the already existing algorithms on challenging instances and has very good behavior for instances closer to the challenging one.

Acknowledgments

This research was supported in part by US National Science Foundation Grant ITR-0326155. The authors would like to thank the referees for their valuable input and suggestions. A preliminary version of this paper appeared in the *Proceedings of the Second International Workshop on Bioinformatics Research and Applications* [6].

References

[1] M. Blanchette, "Algorithms for Phylogenetic Footprinting," *Proc. Fifth Ann. Int'l Conf. Computational Molecular Biology (RECOMB '01)*, Apr. 2001.

[2] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," *Proc. Fifth Ann. Int'l Conf. Computational Molecular Biology (RECOMB '01),* Apr. 2001.
[3] A.M. Carvalho, A.T. Freitas, A.L. Oliveira, and M.-F. Sagot, "A Highly Scalable Algorithm for the Extraction of CIS-Regulatory Regions," *Proc. Third Asia Pacific Bioinformatics Conf.,* Y.-P.P. Chen and L. Wong, eds., vol. 1, pp. 273-282, 2005.
[4] F.Y.L. Chin and C.M. Leung, "Voting Algorithms for Discovering Long Motifs," *Proc. Third Asia-Pacific Bioinformatics Conf. (APBC '05),* pp. 261-271, Jan. 2005.
[5] F.Y.L. Chin and C.M. Leung, "An Efficient Algorithm for String Motif Discovery," *Proc. Fourth Asia-Pacific Bioinformatics Conf. (APBC '06),* pp. 79-88, Feb. 2006.
[6] J. Davila, S. Balla, and S. Rajasekaran, "Space and Time Efficient Algorithms for Planted Motif Search," *Proc. Second Int'l Workshop Bioinformatics Research and Applications (IWBRA '06),* May 2006.
[7] E. Eskin and P. Pevzner, "Finding Composite Regulatory Patterns in DNA Sequences," *Bioinformatics,* vol. S1, pp. 354-363, 2002.
[8] P.A. Evans and A. Smith, "Toward Optimal Motif Enumeration," *Proc. Eighth Int'l Workshop Algorithms and Data Structures (WADS '03),* pp. 47-58, 2003.
[9] P.A. Evans, A. Smith, and H.T. Wareham, "On the Complexity of Finding Common Approximate Substrings," *Theoretical Computer Science,* vol. 306, pp. 407-430, 2003.
[10] N. Li and M. Tompa, "Analysis of Computational Approaches for Motif Discovery," *Algorithms for Molecular Biology,* vol. 1, no. 8, May 2006.
[11] M. Li, B. Ma, and L. Wang, "On the Closest String and Substring Problems," *J. ACM,* vol. 49, no. 2, pp. 157-171, Mar. 2002.
[12] L. Marsan and M.-F. Sagot, "Extracting Structured Motifs Using a Suffix-Tree—Algorithms and Application to Promoter Consensus Identification," *Proc. Int'l Conf. Computational Molecular Biology (RECOMB '00).* 2000.
[13] G. Pavesi, G. Mauri, and G. Pesole, "An Algorithm for Finding Signals of Unknown Length in DNA Sequences," *Bioinformatics,* vol. 17, supplement 1, pp. 207-214, 2001.
[14] P. Pevzner and S.-H. Sze, "Combinatorial Approaches to Finding Subtle Signals in DNA Sequences," *Proc. Eighth Int'l Conf. Intelligent Systems for Molecular Biology,* pp. 269-278, 2000.
[15] N. Pisanti, A.M. Carvalho, L. Marsan, and M.-F. Sagot, "RISOTTO: Fast Extraction of Motifs with Mismatches," *Proc. Seventh Latin Am. Theoretical Informatics Symp.,* J.R. Correa, A. Hevia, and M. Kiwi, eds., pp. 757-768, 2006.
[16] A. Price, S. Ramabhadran, and P. Pevzner, "Finding Subtle Motifs by Branching from Sample Strings," *Proc. Second European Conf. Computational Biology (ECCB '03), Bioinformatics,* supplementary ed., 2003.
[17] S. Rajasekaran, S. Balla, and C.-H. Huang, "Exact Algorithms for the Planted Motif Problem," *J. Computational Biology,* vol. 12, no. 8, pp. 1117-1128, Oct. 2005.
[18] S. Rajasekaran, "Motif Search Algorithms," *Handbook of Computational Molecular Biology,* CRC Press, 2005.
[19] M.F. Sagot, "Spelling Approximate Repeated or Common Motifs Using a Suffix Tree," *Proc. Latin American Symp. Theoretical Informatics (LATIN '98),* C.L. Lucchesi and A.V. Moura, eds., pp. 111-127, 1998.
[20] S. Sinha and M. Tompa, "A Statistical Method for Finding Transcription Factor Binding Sites," *Proc. Eighth Int'l Conf. Intelligent Systems for Molecular Biology,* pp. 344-354, Aug. 2000.
[21] S. Sze, S. Lu, and J. Chen, "Integrating Sample-Driven and Pattern-Driven Approaches in Motif Finding," *Proc. Fourth Int'l Workshop Algorithms in Bioinformatics (WABI '04),* 2004.
[22] M. Tompa, N. Li, T.L. Bailey, G.M. Church, B. De Moor, E. Eskin, A.V. Favorov, M.C. Frith, Y. Fu, W.J. Kent, V.J. Makeev, A.A. Mironov, W.S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu, "Assessing Computational Tools for the Discovery of Transcription Factor Binding Sites," *Nature Biotechnology,* vol. 23, no. 1, pp. 137-144, Jan. 2005.

Jaime Davila received the BS degree in mathematics and computer science and the MSc degree in mathematics from the Universidad de los Andes, Bogota, Colombia. He is currently working toward the PhD degree in the Computer Science and Engineering Department at the University of Connecticut, Storrs, under the guidance of Dr. Sanguthevar Rajasekaran, working on efficient algorithms for motif discovery in biosequences.

Sudha Balla is currently working toward the PhD degree in computer science and engineering at the University of Connecticut, working on efficient algorithms for motif discovery in biosequences.

Sanguthevar Rajasekaran received the ME degree in automation from the Indian Institute of Science, Bangalore, in 1983 and the PhD degree in computer science from Harvard University in 1988. Currently, he is the UTC Chair Professor of Computer Science and Engineering at the University of Connecticut (UConn) and the director of the Booth Engineering Center for Advanced Technologies (BECAT). Before joining UConn, he has served as a faculty member in the Computer and Information Science and Engineering (CISE) Department at the University of Florida and in the Computer and Information Science (CIS) Department at the University of Pennsylvania. During 2000-2002, he was the chief scientist for Arcot Systems. His research interests include parallel algorithms, bioinformatics, data mining, randomized computing, computer simulations, and combinatorial optimization. He has published more than 150 articles in journals and conferences. He has coauthored two texts on algorithms and coedited four books on algorithms and related topics. He is a senior member of the IEEE.